Polyglot

version 0.0.2

Ryan M. Kraus

April 15, 2016

Contents

Polyglot Virtual Node Server Framework

Polyglot Virtual Node Server Framework is an application that makes it easy and quick to both develop and maintain virtual node servers for the ISY-994 home automation controller by Universal Devices Inc. Using virtual node servers, the ISY-994i is able to communicate with and control third-party devices to which the ISY-994i cannot natively connect.

Polyglot is written primarily with Python 2.7 and makes it easy to develop new Virtual Node Servers with Python 2.7. It should, however, by noted, that Virtual Node Servers may by developed using any language. Polyglot is intended to be run on a Raspberry Pi 2 Model B, but could potentially run on any ARM based machine running Linux with Python 2.7.

This document will document the usage of and development for Polyglot. For additional help, please reference the UDI Forum.

Usage

Installation

Polyglot ships in a compiled, system dependent container. To install, place this file in the desired directory on your system and launch.

```
./polyglot.pyz
```

This will launch Polyglot and create a directory titled *config* in the current directory. Polyglot will store all of its configuration and its log inside of this directory. You may specify a manual path for this directory using the command line flags.

The following are all of the available flags at the command line.

```
-h, --help            show this help message and exit
-c CONFIG_DIR, --config CONFIG_DIR
                      Polyglot configuration directory
-v, --verbose         Enable verbose logging
-vv                   Enable very verbose logging
```

While running in its default mode, Polyglot will log all warnings and errors. Verbose logging will include info messages. Very verbose mode adds debug messages that could be useful when developing a new node server.

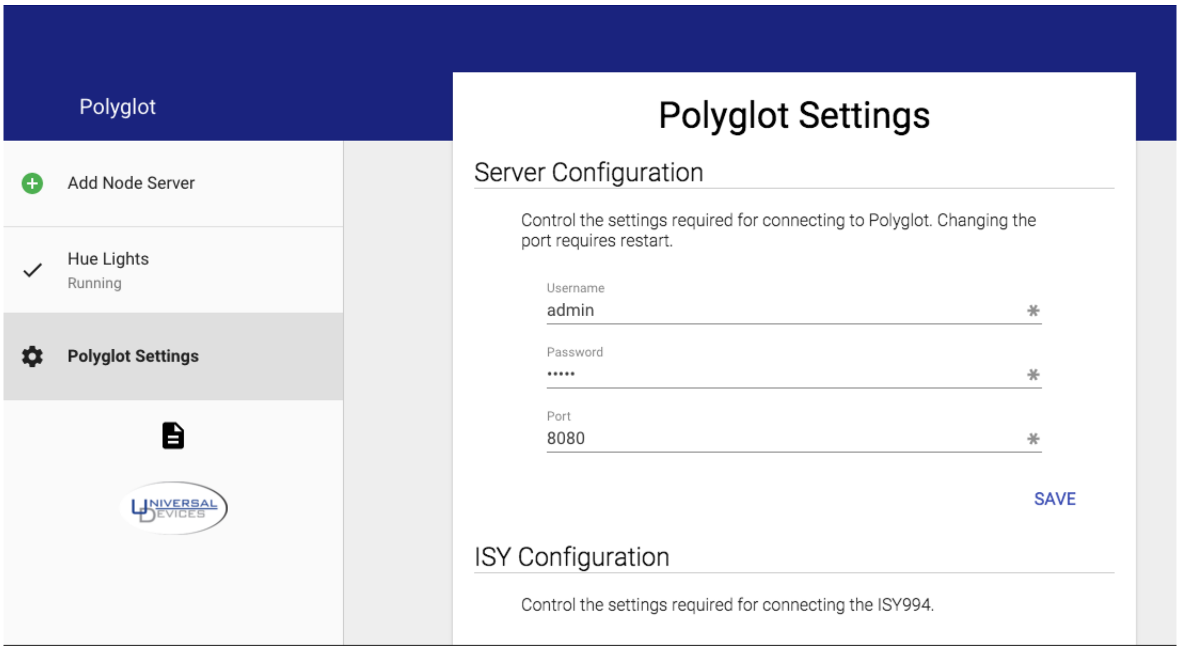
Once Polyglot is running, the user interface may be accessed by opening your favorite browser and navigating to:

```
http://localhost:8080
```

The default username and password are both *admin*.

If you are accessing the frontend from another machine, replace *localhost* with the IP Address or URL of the machine running Polyglot. If you are having trouble accessing the user interace from a remote machine, check your firewall settings.

User Interface



Polyglot Settings

Server Configuration

Control the settings required for connecting to Polyglot. Changing the port requires restart.

Username: admin
Password: •••••
Port: 8080

ISY Configuration

Control the settings required for connecting the ISY994.

The user interface is designed to be simple and intuitive to use. Pictured above is the settings page. Using the menu bar on the left, new node servers can be added and existing node servers may be monitored. The button on the bottom of the menu will open Polyglot's log in a new browser window.

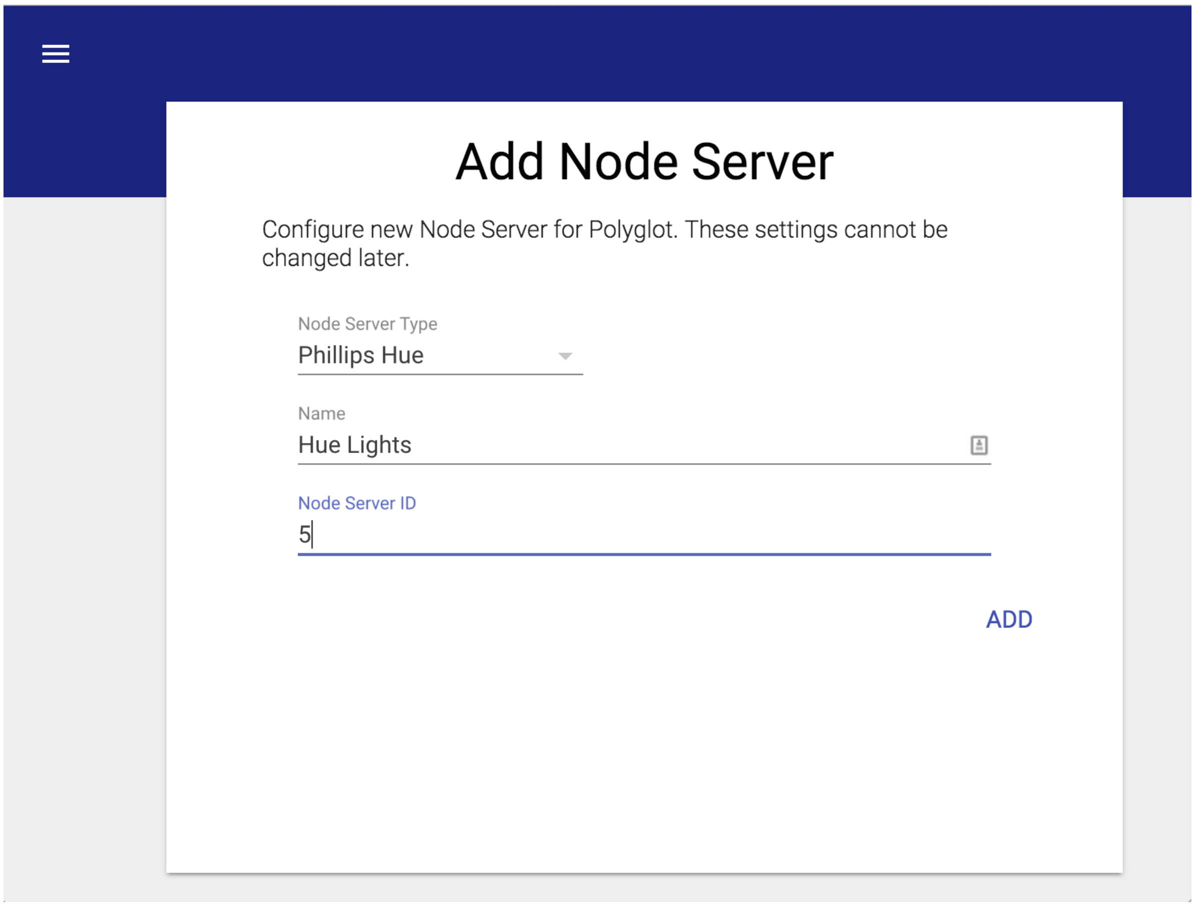
The user interface is fully compatible with both tablet and mobile devices.

Settings

The settings view allows the user to alter settings for Polyglot's HTTP server as well as Polyglot's connection to the ISY controller. It is recomended that the username and password are changed from the default. If a new different port is desired, it may be set in the *Server Configuration* block.

It is also necessary to set the username, password, host name, and port required for connecting to the ISY. These may be configured in the *ISY Configuration* block.

Adding Node Server



Add Node Server

Configure new Node Server for Polyglot. These settings cannot be changed later.

Node Server Type
Phillips Hue

Name
Hue Lights

Node Server ID
5

To add a node server, navigate to the *Add Node Server* view using the menu. This view is pictured above.

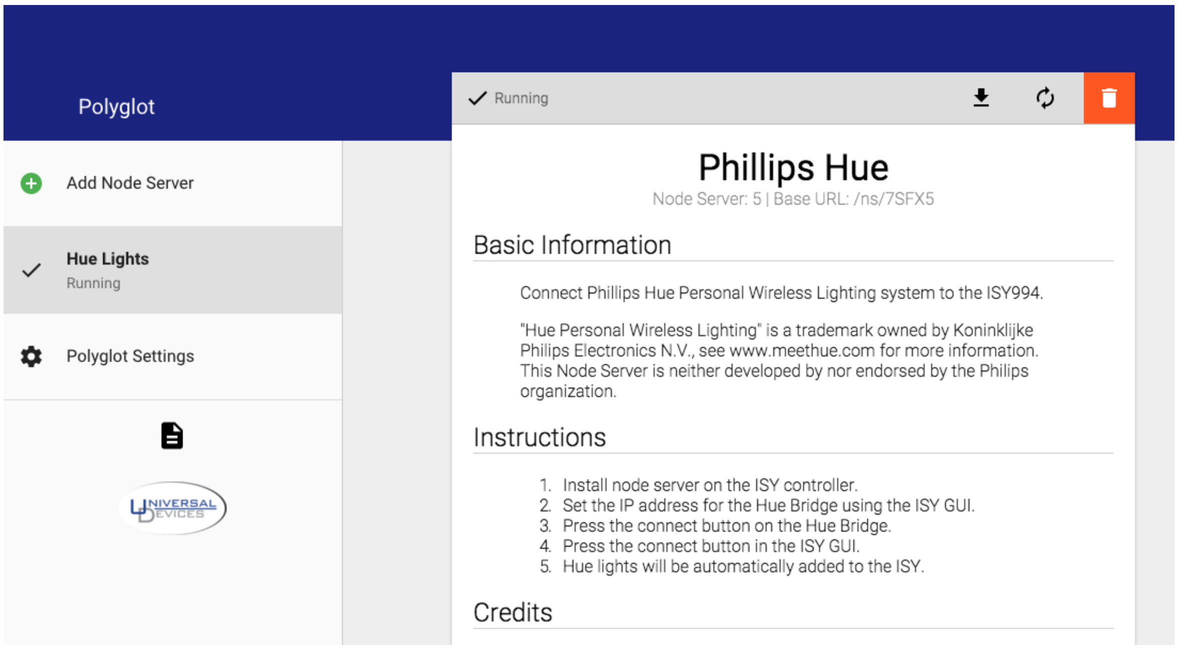
Populate this form with the details for the new node server. Select a type from all installed types using the drop down. Give the node server any name allows for easy recognition. Finally, populate the *Node Server ID* field with an ID that is available in the ISY. Press *ADD* when complete.

The node server will now be available in Polyglot. You may navigate to it using the menu. The node server view in Polyglot will show the Node Server ID, Base URL, and allow for the Profile to be downloaded.

In order to access the node server from the ISY, it must be added to the ISY. To do this, inside of the ISY console, navigate to Node Servers then Configure then the Node Server ID that was set while creating the node server. This will open a dialog that accepts all the information from the node server view. Populate this with the Profile Name and Base URL from the node server view. The User ID, Passsword, Host Name, and Port here must be the values used for connecting to Polyglot. Timeout may be left as 0, and the Isy User should be set to the appropriate user ID that was configured in Polyglot. If you are unsure, use 0.

Click the *Upload Profile* button and navigate to the zip file obtained from Polyglot's node server view. Once this has been uploaded, click *Ok* and restart the ISY controller. Once the ISY has fully rebooted, restart the node server in Polyglot using the node server view.

Managing Node Servers



Phillips Hue

Node Server: 5 | Base URL: /ns/7SFX5

Basic Information

Connect Phillips Hue Personal Wireless Lighting system to the ISY994.

"Hue Personal Wireless Lighting" is a trademark owned by Koninklijke Philips Electronics N.V., see www.meethue.com for more information. This Node Server is neither developed by nor endorsed by the Philips organization.

Instructions

1. Install node server on the ISY controller.
2. Set the IP address for the Hue Bridge using the ISY GUI.
3. Press the connect button on the Hue Bridge.
4. Press the connect button in the ISY GUI.
5. Hue lights will be automatically added to the ISY.

Credits

Clicking a Node Server in the menu will activate the node server view. In this view, there is a menu bar at the top. This menu bar will indicate is the node server is Running or Stopped. It also provides buttons to download the profile, restart the node server, or delete the node server.

Also in this view are instructions for using this node server. Different node servers may have their own instructions on how to use them in the ISY. Any open-source, third party libraries that were used for the development of the node server are also credited here.

If the node server were to crash, a red X will appear next to it in the menu and it will be indicated in the menu bar on the top of the node server view. If this happens, it is best to save the log for debugging and then restart the node server using the button in the menu bar.

Viewing Polyglot Log

There is a file icon below all the main menu items. Clicking this icon will open Polyglot's log in a new browser window. This log file is critical for debugging issues with Polyglot.

Node Server Development

Background

Node servers in Polyglot are nothing more than stand alone processes that are managed by Polyglot. Polyglot communicates with the node servers by reading the STDOUT and STDERR streams as well as writing to the STDIN stream. STDIN and STDOUT messages are JSON formatted commands that are documented in *Polyglot Node Server API*.

File Structure

Node servers are defined in self contained folders. The name given to this folder will be the node server ID and must be unique form all other node servers. New node servers can be stored in Polyglot's configuration directory in the folder titled *node_servers*. Inside of this folder, at least the following three files must exist.

- *profile.zip* is the profile that must be uploaded to the ISY describing the node server. This file is documented in the ISY Node Server API documentation.
- *instructions.txt* should be a file containing instructions on the use of the node server documented using markdown. The contents of this file will be formatted and displayed on the frontend.
- *server.json* is the metadata used by Polyglot to identify the node server. This file is documented in the next section.

The rest of the node server's folder should contain the code required to execute the node server and all necessary libraries with the exception of those explicitly included as part of the Polyglot distribution.

Node servers are executed in special directories in the user's configuration directory. Each node server type is assigned its own directory. Any required inforation may be written to this directory. Keep in mind, that all running node servers of the same type will share the same directory.

Server Metadata

The *server.json* file in the node server source directory is a JSON formatted file that informs Polyglot of how the node server is executed as well as other important details about the node server. The file contains a dictionary formatted object with specific fields. A sample *server.json* is included below. It has been extracted from the Philips Hue node server.

```
{
    "name": "Phillips Hue",
    "docs": "https://www.universal-devices.com/",
    "type": "python",
    "executable": "hue.py",
    "description": "Connect Phillips Hue Personal Wireless Lighting system to the ISY994.",
    "notice": "\"Hue Personal Wireless Lighting\" is a trademark owned by Koninklijke Philip
    "credits": [
        {
            "title": "phue: A Python library for Philips Hue",
            "author": "Nathanaël Lécaudé (studioimaginaire)",
            "version": "0.9",
            "date": "May 18, 2015",
            "source": "https://github.com/studioimaginaire/phue/tree/c48845992b476f4b1de9549
            "license": "https://raw.githubusercontent.com/studioimaginaire/phue/c48845992b47
        }
    ]
}
```

Below is a description of the required fields:

- *name* is the name of the node server type as it will be displayed to the user.
- *docs* is a link to an appropriate website about the node server. This value is not currently displayed anywhere.
- *type* is the node server executable type. This instructs Polyglot as to how the node server should be launched. Currently, only *python* is accepted.
- *executable* is the file that Polyglot should execute to start the node server process.
- *description* is a short description of the node server that will be displayed to the user on the frontend.
- *notice* contains any important notices the user might need to know.
- *credits* is a list of dictionaries indicating all third party library used in the node server. Some open source projects require that they be credited some where in the project. Others do not. Either way, it is nice to give credit here. When including a third party library in your node server, ensure that it is licensed for commercial use.

In the credits list:

- *title* is the title of the third party library.
- *author* is the author of the third party library.
- *version* is the appropriate versioning tag used to identify the third party library.
- *date* is the date the third party library was either released or obtained.
- *source* is a link to the library's source code.
- *license* is a link to the library's license file. Ensure that this is a static link whose contents cannot be changed. Linking to a specific GitHub commit is handy for this.

It can be a good idea to check the formatting of this file with a JSON linter before attempting to load the node server in Polyglot. If this file cannot be read, for whatever reason, the node server will not appear in the Polyglot frontend and an error will be logged.

Python Development

A Python 2.7 compatible implimentation of the API is provided with Polyglot to assist in Node Server development. It may be easily imported as shown below. In the future, more libraries may be made available and more languages may be supported.

```
from polyglot import nodeserver_api
```

The provided Node Server Library exposes all of the ISY controller's Node Server RESTful API as is. Data recieved by Polyglot's web server is parsed and directed immediately to the node server process via this library. The library will also send messages back up to Polyglot to be transmitted directly to the ISY. The only exception to this rule is that node ID's will not have the node server ID prefix prepended to them. It will also be expected that the node server will not prepend these prefixes. Polyglot will handle the node ID prefixes on behalf of the node servers.

There also exists, in the Python library, some abstract classes that may be used to ease the development of a new node server. Except in rare cases where it may not be appropriate, it is recomended that these be used.

When Python is used to develop node server, the Polyglot environment is loaded into the Python path. This environment includes the Requests library.

Python Polyglot Library

Summary

This library consists of four classes and one function to assist with node server development. The classes `polyglot.nodeserver_api.NodeServer` and

`polyglot.nodeserver_api.SimpleNodeServer` and basic structures for creating a node server. The class **`polyglot.nodeserver_api.Node`** is used as an abstract class to crate custom nodes for node servers. The class **`polyglot.nodeserver_api.PolyglotConnector`** is a bottom level implimentation of the API used to communicate between Polyglot and your node server. Finally, included in this library is a method decorator, **`polyglot.nodeserver_api.auto_request_report()`**, that wraps functions and methods to automatically handle report requests from the ISY.

Custom Node Types

When creating a new node server, each node type that will be controlled by the server must be defined. This abstract class may be used as a skeleton for each node type. When inheriting this class, a new method should be defined for each command that the node can perform. Additionally, the _drivers and _commands attributes should be overwritten to define the drivers and commands relevant to the node.

class `polyglot.nodeserver_api.`**`Node`** (`parent`, `address`, name, `primary=True`, `manifest=None`)

Abstract class for representing a node in a node server.

Parameters:

- **parent** (*polyglot.nodeserver_api.NodeServer*) -- The node server that controls the node
- **address** (*str*) -- The address of the node in the ISY without the node server ID prefix
- **name** (*str*) -- The name of the node
- **primary** -- The primary node for the device this node belongs to,

: or True if it's the primary. :type primary: polyglot.nodeserver_api.Node or True if this node is the primary. :param manifest: The node manifest saved by the node server :type manifest: dict or None

`_drivers` = *{}*

The drivers controlled by this node. This is a dictionary of lists. The key's are the driver names as defined by the ISY documentation. Each list contains three values: the initial value, the UOM identifier, and a function that will properly format the value before assignment.

Insteon Dimmer Example:

```
_drivers = {
    'ST': [0, 51, int],
    'OL': [100, 51, int],
    'RR': [0, 32, int]
}
```

`_commands` = *{}*

A dictionary of the commands that the node can perform. The keys of this dictionary are the names of the command. The values are functions that must be defined in the node object that perform the necessary actions and return a boolean indicating the success or failure of the command.

`add_node ()`

Adds node to the ISY

Returns boolean: Indicates success or failure of node addition

`get_driver` (`driver=None`)

Gets a driver's value

Parameters: **driver** (*str or None*) -- The driver to return the value for

Returns: The current value of the driver

`manifest`

The node's manifest entry. Indicates the current value of each of the drivers. This is called by the node server to create the full manifest.

Type: dict

`node_def_id` = ''

The node's definition ID defined in the node server's profile

`query ()`

Abstractly queries the node. This method should generally be overwritten in development.

Returns boolean: Indicates success or failure of node query

`report_driver` `(driver=None)`

Reports a driver's current value to ISY

Parameters: **driver** (*str or None*) -- The name of the driver to report. If None, all drivers are reported.

Returns boolean: Indicates success or failure to report driver value

`run_cmd` `(command, **kwargs)`

Runs one of the node's commands.

Parameters:

- **command** (*str*) -- The name of the command
- **kwargs** (*dict*) -- The parameters specified by the ISY in the incoming request. See the ISY Node Server documentation for more information.

Returns boolean: Indicates success or failure of command

`set_driver` `(driver, value, uom=None, report=True)`

Updates the value of one of the node's drivers. This will pass the given value through the driver's formatter before assignment.

Parameters:

- **driver** (*str*) -- The name of the driver
- **value** -- The new value for the driver
- **uom** (*int or None*) -- The given values unit of measurement. This should correspond to the UOM IDs used by the ISY. Refer to the ISY documentation for more information.
- **report** (*boolean*) -- Indicates if the value change should be reported to the ISY. If False, the value is changed silently.

Returns boolean: Indicates success or failure to set new value

Polyglot API Implimentation

This class impliments the Polyglot API and calls registered functions when the API is invoked. This class is a singleton and will not allow itself to be initiated more than once. This class binds itself to the STDIN stream to accept commands from Polyglot.

To create a connection in your node server to Polyglot, use something similar to the following. This creates the connection, connect to Polyglot, and then waits for the Node Server's configuration to be received. The configuration will be the first command received from Polyglot and will never be sent again after the first transmission.:

```
poly = PolyglotConnector()
poly.connect()
poly.wait_for_config()
```

Then, commands can be sent upstream to Polyglot or to the ISY by using the connector's methods.:

```
poly.send_error('This is an error message. It will be in Polyglot\'s log.')
poly.add_node('node_id_1', 'NODE_DEFINITION', 'node_id_0', 'New Node')
poly.report_status('node_id_1', 'ST', value=55, uom=51)
poly.remove_node('node_id_1')
```

To respond to commands received from Polyglot and the ISY, handlers must be registered for events. The handlers arguments will be the parameters specified in the API for that event. This will look something like the following.:

```
def status_handler(node_address, request_id=None):
    print('Status Event Handler Called')

poly.listen('status', status_handler)
```

Now, when the ISY requests a status update from Polyglot, this function will be called. Handlers will not be called in the node server's main thread.

class `polyglot.nodeserver_api.`**`PolyglotConnector`**

Polyglot API implimentation. Connects to Polyglot and handles node server IO.

Raises: RuntimeError

`LOGGER` = *None*

Commands that may be invoked by Polyglot

`add_node` **(**`node_address, node_def_id, primary, name`**)**

Adds a node to the ISY. To make this node the primary, set primary to the same value as node_address.

Parameters:

- **node_address** (*str*) -- The full address of the node (e.g. 'dimmer_1')
- **node_def_id** (*str*) -- The id of the node definition to use for this node
- **primary** (*str*) -- The primary node for the device this node belongs to
- **name** (*str*) -- The name of the node

`change_node` **(**`node_address, node_def_id`**)**

Changes the node definition to use for an existing node. An example of this is may be to change a thermostat node from Fahrenheit to Celsius.

Parameters:

- **node_address** (*str*) -- The full address of the node (e.g. 'dimmer_1')
- **node_def_id** (*str*) -- The id of the node definition to use for this node

`connect ()`

Connects to Polyglot if not currently connected

`connected`

Indicates if the object is connected to Polyglot. Can be set to control connection with Polyglot.

Type: boolean

`disconnect ()`

Disconnects from Polyglot. Blocks the thread until IO stream is clear

`exit` **(**`*args, **kwargs`**)**

Tells Polyglot that this Node Server is done.

`get_params` **(**`**kwargs`**)**

Get the params from nodeserver and makes them available to the nodeserver api

install (*args, **kwargs)

Abstract method to install the node server in the ISY. This has not been implimented yet and running it will raise an error.

Raises: NotImplementedError

listen (event, handler)

Register an event handler. Returns True on success. Event names are defined in *commands*. Handlers must be callable.

Parameters:

- **event** (*str*) -- Then event name to listen for.
- **handler** (*callable*) -- The callable event handler.

pong (*args, **kwargs)

Sends pong reply to Polyglot's ping request. This verifies that the communication between the Node Server and Polyglot is functioning.

remove_node (node_address)

Removes a node from the ISY. A node cannot be removed if it is the primary node for at least one other node.

Parameters: **node_address** (*str*) -- The full address of the node (e.g. 'dimmer_1')

report_command (node_address, command, value=None, uom=None, **kwargs)

Sends a command to the ISY that may be used in programs and/or scenes. A common use of this is a physical switch that somebody turns on or off. Each time the switch is used, a command should be reported to the ISY. These are used for scenes and control conditions in ISY programs.

Parameters:

- **node_address** (*str*) -- The full address of the node (e.g. 'switch_1)
- **command** (*str*) -- The command to perform (e.g. 'DON', 'CLISPH', etc.)
- **value** (*str, int, or float*) -- Optional unnamed value the command used
- **uom** (*int or str*) -- Optional units of measurement of value
- **<pN>.<uomN>** (*optional*) -- Nth Parameter name (e.g. 'level') . Unit of measure of the Nth parameter (e.g. 'seconds', 'uom58')

report_request_status (request_id, success)

When the ISY sends a request to the node server, the request may contain a 'requestId' field. This indicates to the node server that when the request is completed, it must send a fail or success report for that request. This allows the ISY to in effect, have the node server synchronously perform tasks. This message must be sent after all other messages related to the task have been sent.
For example, if the ISY sends a request to query a node, all the results of the query must be sent to the ISY before a fail/success report is sent.

Parameters:

- **request_id** (*str*) -- The request ID the ISY supplied on a request to the node server.
- **success** (*bool*) -- Indicates if the request was sucessful

report_status (node_address, driver_control, value, uom)

Updates the ISY with the current value of a driver control (e.g. the current temperature, light level, etc.)

Parameters:

- **node_address** (*str*) -- The full address of the node (e.g. 'dimmer_1')
- **driver_control** (*str*) -- The name of the status value (e.g. 'ST', 'CLIHUM', etc.)
- **value** (*str, float, or int*) -- The numeric status value (e.g. '80.5')
- **uom** (*int or str*) -- Unit of measure of the status value

`send_config` **(**`config_data`**)**

Update the configuration in Polyglot.

Parameters: **config_data** (*dict*) -- Dictionary of updated configuration

Raises: ValueError

`send_error` **(**`err_str`**)**

Enqueue an error to be sent back to Polyglot.

Parameters: **err_str** (*str*) -- Error text to be sent to Polyglot log

`uptime`

The number of sections the connection with Polyglot has been alive

Type: float

`wait_for_config` **()**

Blocks the thread until the configuration is received

Node Server Classes

class `polyglot.nodeserver_api.`**`NodeServer`** (`poly`, `shortpoll=1`, `longpoll=30`)

It is generally desireable to not be required to bind to each event. For this reason, the NodeServer abstract class is available. This class should be abstracted. It binds appropriate handlers to the API events and contains a suitable run loop. It should serve as a basic structure for any node server.

Parameters:

- **poly** (*polyglot.nodeserver_api.PolyglotConnector*) -- The connected Polyglot connection
- **optional shortpoll** (*int*) -- The seconds between poll events
- **optional longpoll** (*int*) -- The second between longpoll events

`add_node` **(**`node`**)**

Add this node to the polyglot

Returns bool: True on success

`long_poll` **()**

Called every longpoll seconds for less important polling.

`on_add_all` **(**`request_id=None`**)**

Received add all command from ISY

Parameters: **optional request_id** (*str*) -- Status request id

Returns bool: True on success

`on_added` **(**`node_address, node_def_id, primary_node_address, name`**)**

Received node added report from ISY

Parameters:

- **node_address** (*str*) -- The address of the node to act on
- **node_def_id** (*str*) -- The node definition id
- **primary_node_address** (*str*) -- The node server's primary node address
- **name** (*str*) -- The node's friendly name
- **optional request_id** (*str*) -- Status request id

Returns bool: True on success

on_cmd (`node_address, command, value=None, uom=None, request_id=None, **kwargs`)

Received run command from ISY

Parameters:

- **node_address** (*str*) -- The address of the node to act on
- **command** (*str*) -- The command to run
- **value** (*optional*) -- The value of the command's unnamed parameter
- **uom** (*optional*) -- The units of measurement for the unnamed parameter
- **optional request_id** (*str*) -- Status request id
- **<pN>.<uomN>** (*optional*) -- The value of parameter pN with units uomN

Returns bool: True on success

on_config (`**data`)

Received configuration data from Polyglot

Parameters: **data** (*dict*) -- Configuration data

Returns bool: True on success

on_disabled (`node_address`)

Received node disabled report from ISY

Parameters: **node_address** (*str*) -- The address of the node to act on

Returns bool: True on success

on_enabled (`node_address`)

Received node enabled report from ISY

Parameters: **node_address** (*str*) -- The address of the node to act on

Returns bool: True on success

on_exit (`*args, **kwargs`)

Polyglot has triggered a clean shutdown. Generally, this method does not need to be orwritten.

Returns bool: True on success

on_install (`profile_number`)

Received install command from ISY

Parameters: **profile_number** (*int*) -- Noder Server's profile number

Returns bool: True on success

on_query (`node_address, request_id=None`)

Received query command from ISY

Parameters:
- **node_address** (*str*) -- The address of the node to act on
- **optional request_id** (*str*) -- Status request id

Returns bool: True on success

on_removed (`node_address`)
Received node removed report from ISY

Parameters: **node_address** (*str*) -- The address of the node to act on

Returns bool: True on success

on_renamed (`node_address, name`)
Received node renamed report from ISY

Parameters:
- **node_address** (*str*) -- The address of the node to act on
- **name** (*str*) -- The node's friendly name

Returns bool: True on success

on_status (`node_address, request_id=None`)
Received status command from ISY

Parameters:
- **node_address** (*str*) -- The address of the node to act on
- **optional request_id** (*str*) -- Status request id

Returns bool: True on success

poll ()
Called every shortpoll seconds to allow for updating nodes.

poly = *None*
The Polyglot Connection

Type: polyglot.nodeserver_api.PolyglotConnector

run ()
Run the Node Server. Exit when triggered. Generally, this method should not be overwritten.

class `polyglot.nodeserver_api.`**SimpleNodeServer** (`poly, shortpoll=1, longpoll=30`)
Simple Node Server with basic functionality built-in. This class inherits from **`polyglot.nodeserver_api.NodeServer`** and is the best starting point when developing a new node server. This class impliments the idea of manifests which are dictionaries that contain the relevant information about all of the nodes. The manifest gets sent to Polyglot to be saved as part of the configuration. This allows the node server to automatically recall its last known values when it is restarted.

add_node (`*args, **kwargs`)
Add node to the Polyglot and the nodes dictionary.

Parameters: **node** (*polyglot.nodeserver_api.Node*) -- The node to add

Returns boolean: Indicates success or failure of node addition

exist_node (`address`)
Check if a node exists by it's address.

Parameters: **address** (*str*) -- The node address

Returns boolean:

get_node (address)

Get a node by it's address.

Parameters: **address** (*str*) -- The node address

Returns polyglot.nodeserver_api.Node: on success, otherwise False.

nodes = *OrderedDict()*

Nodes registered with this node server. All nodes are automatically added by the add_node method. The keys are the node IDs while the values are instances of `polyglot.nodeserver_api.Node`. Classes inheriting can access this directly, but the prefered method is by using get_node or exist_node methods.

on_add_all (*args, **kwargs)

Adds all nodes to the ISY. Also sends requests reponses when necessary.

Parameters: **optional request_id** (*str*) -- Status request id

Returns bool: True on success

on_added (node_address, node_def_id, primary_node_address, name)

Internally indicates that the specified node has been added to the ISY.

Parameters:

- **node_address** (*str*) -- The address of the node to act on
- **node_def_id** (*str*) -- The node definition id
- **primary_node_address** (*str*) -- The node server's primary node address
- **name** (*str*) -- The node's friendly name
- **optional request_id** (*str*) -- Status request id

Returns bool: True on success

on_cmd (*args, **kwargs)

Runs the specified command on the specified node. Also sends requests reponses when necessary.

Parameters:

- **node_address** (*str*) -- The address of the node to act on
- **command** (*str*) -- The command to run
- **value** (*optional*) -- The value of the command's unnamed parameter
- **uom** (*optional*) -- The units of measurement for the unnamed parameter
- **optional request_id** (*str*) -- Status request id
- **<pN>.<uomN>** (*optional*) -- The value of parameter pN with units uomN

Returns bool: True on success

on_exit (*args, **kwargs)

Triggers a clean shut down of the node server by saving the manifest, clearing the IO, and stopping.

Returns bool: True on success

on_query (*args, **kwargs)

Queries each node and reports all control values to the ISY. Also responds to report requests if necessary.

Parameters:
- **node_address** (*str*) -- The address of the node to act on
- **optional request_id** (*str*) -- Status request id

Returns bool: True on success

on_removed (node_address)

Internally indicates that a node has been removed from the ISY.

Parameters: **node_address** (*str*) -- The address of the node to act on

Returns bool: True on success

on_renamed (node_address, name)

Changes the node name internally to match the ISY.

Parameters:
- **node_address** (*str*) -- The address of the node to act on
- **name** (*str*) -- The node's friendly name

Returns bool: True on success

on_status (*args, **kwargs)

Reports the requested node's control values to the ISY without forcing a query. Also sends requests reponses when necessary.

Parameters:
- **node_address** (*str*) -- The address of the node to act on
- **optional request_id** (*str*) -- Status request id

Returns bool: True on success

update_config ()

Updates the configuration with new node manifests and sends the configuration to Polyglot to be saved.

Helper Functions

polyglot.nodeserver_api.**auto_request_report** (fun)

Python decorator to automate request reporting. Decorated functions must return a boolean value indicating their success or failure. It the argument *request_id* is passed to the decorated function, a response will be sent to the ISY. This decorator is implimented in the SimpleNodeServer.

Python Node Server Example

The following is a brief example of some implemented node servers written in Python. The examples included are pulled from the Philips Hue Node Server and may not be current with the actual code used in that node server and is redacted a bit for clarity, but will serve as a solid jumping off point for defining the process by which a new node server can be developed.

Node Type Definition

Some may find it easiest to start by developing all the types of nodes that the node server may be controlling. As these are being defined in code, it may be best to also define them in the file that will eventually make up the *profile.zip* file. Documentation for profile files is available in the ISY Virtual Node Server API documentation.

Below is the definition for a Hue color changing light.

```
from converters import RGB_2_xy, color_xy, color_names
from functools import partial
from polyglot.nodeserver_api import Node

def myint(value):
    """ round and convert to int """
    return int(round(float(value)))


def myfloat(value, prec=4):
    """ round and return float """
    return round(float(value), prec)

class HueColorLight(Node):
    """ Node representing Hue Color Light """

    def __init__(self, parent, address, name, lamp_id, manifest=None):
        super(HueColorLight, self).__init__(parent, address, name, manifest)
        self.lamp_id = int(lamp_id)

    def query(self):
        """ command called by ISY to query the node. """
        updates = self.parent.query_node(self.address)
        if updates:
            self.set_driver('GV1', updates[0], report=False)
            self.set_driver('GV2', updates[1], report=False)
            self.set_driver('ST', updates[2], report=False)
            self.report_driver()
            return True
        else:
            return False

    def _set_brightness(self, value=None, **kwargs):
        """ set node brightness """
        # pylint: disable=unused-argument
        if value is not None:
            value = int(value / 100. * 255)
            if value > 0:
                command = {'on': True, 'bri': value}
            else:
                command = {'on': False}
        else:
            command = {'on': True}
        return self._send_command(command)
    def _on(self, **kwargs):
        """ turn light on """
        status = kwargs.get("value")
        return self._set_brightness(value=status)

    def _off(self, **kwargs):
        """ turn light off """
        # pylint: disable=unused-argument
        return self._set_brightness(value=0)

    def _set_color_rgb(self, **kwargs):
        """ set light RGB color """
        color_r = kwargs.get('R.uom56', 0)
        color_g = kwargs.get('G.uom56', 0)
        color_b = kwargs.get('B.uom56', 0)
        (color_x, color_y) = RGB_2_xy(color_r, color_g, color_b)
        command = {'xy': [color_x, color_y], 'on': True}
        return self._send_command(command)

    def _set_color_xy(self, **kwargs):
        """ set light XY color """
        color_x = kwargs.get('X.uom56', 0)
        color_y = kwargs.get('Y.uom56', 0)
        command = {'xy': [color_x, color_y], 'on': True}
        return self._send_command(command)

    def _set_color(self, value=None, **_):
        """ set color from index """
        ind = int(value) - 1

        if ind >= len(color_names):
            return False

        cname = color_names[int(value) - 1]
        color = color_xy(cname)
        return self._set_color_xy(
            **{'X.uom56': color[0], 'Y.uom56': color[1]})

    def _send_command(self, command):
        """ generic method to send command to hue hub """
        responses = self.parent.hub.set_light(self.lamp_id, command)
        return all(
            [list(resp.keys())[0] == 'success' for resp in responses[0]])

    _drivers = {'GV1': [0, 56, myfloat], 'GV2': [0, 56, myfloat],
                'ST': [0, 51, myint]}
    """ Driver Details:
    GV1: Color X
    GV2: Color Y
    ST: Status / Brightness
    """
    _commands = {'DON': _on, 'DOF': _off,
                 'SET_COLOR_RGB': _set_color_rgb,
                 'SET_COLOR_XY': _set_color_xy,
                 'SET_COLOR': _set_color}
    node_def_id = 'COLOR_LIGHT'
```

As can be seen here, one method is defined for each of the commands that the node may run. The query method from the Node ABC is also overwritten to provide the desired functionality. An additional method called _send_command is also created. This is not called by the ISY directly, but is a helper used to send information to the Hue device. This method calls a method from a third party library that connects to the Hue lighting system.

Additionally, the _drivers, _command, and node_def_id properties are overwritten. This must be done by every node class as it instructs the node server classes on how to interact with this node. Custom formatters myint and myfloat are used to format the control values.

This process must be repeated for each type of node that is desired.

Node Server Creation

Once all the nodes are defined, the node server class can be created.

```python
from polyglot.nodeserver_api import SimpleNodeServer, PolyglotConnector
# ... additional imports are redacted for clarity

class HueNodeServer(SimpleNodeServer):
    """ Phillips Hue Node Server """

    hub = None

    def setup(self):
        """ Initial node setup. """
        # define nodes for settings
        manifest = self.config.get('manifest', {})
        self.nodes['hub'] = HubSettings(self, 'hub', 'Hue Hub', manifest)
        self.connect()
        self.update_config()

    def connect(self):
        """ Connect to Phillips Hue Hub """
        # get hub settings
        hub = self.nodes['hub']
        ip_addr = '{}.{}.{}.{}'.format(
            hub.get_driver('GV1')[0], hub.get_driver('GV2')[0],
            hub.get_driver('GV3')[0], hub.get_driver('GV4')[0])

        # ... Connects to the hub and validate connection. Redacted for clarity.

    def poll(self):
        """ Poll Hue for new lights/existing lights' statuses """

        # ... Connects to Hue Hub and gets current values for lights,
        #     stores in dictionary called lights. Redacted for clarity.

        for lamp_id, data in lights.items():
            address = id_2_addr(data['uniqueid'])
            name = data['name']

            if address not in self.nodes:
                # Add the light to the Node Server if it doesn't already
                # exist. This automatically adds the light to the ISY.
                self.nodes[address] = HueColorLight(
                    self, address, name, lamp_id, manifest)

            (color_x, color_y) = [round(val, 4)
                                  for val in data['state']['xy']]
            brightness = round(data['state']['bri'] / 255. * 100., 4)
            brightness = brightness if data['state']['on'] else 0
            self.nodes[address].set_driver('GV1', color_x)
            self.nodes[address].set_driver('GV2', color_y)
            self.nodes[address].set_driver('ST', brightness)
        return True

    def query_node(self, lkp_address):
        """ find specific node in api. """

        # ... Polls Hue Hub for current specified light values, and updates
        #     Node object with new values. Works very similarly to poll
        #     above. Redacted for clarity.

    def _get_api(self):
        """ get hue hub api data. """

        # ... Uses third party library to get updated Hue Hub information.
        #     Redacted for clarity.

    def long_poll(self):
        """ Save configuration every 30 seconds. """
        self.update_config()
        # In this example, the configuration is autoatically saved every
        # 30 seconds. Make sure your node server saves its configuration
        # at some point.
```

This example class contains four methods that are not part of the abstract class. They are setup, connect, query_node, and _get_api. These functions will probably not appear in all node servers and are very specific to this one.

However, the setup method is a good way to handle any node server setup that must be done that is specific to your node server. In this example, the primary node, the Hue Hub, is created and a connection is attempted.

This class also stores an object called hub as an attribute. This objet is an instance of a class from the third party library used. This object is the actual connection to the Hue Hub. It may be best to follow a similar method when creating node servers so that the code that handles the connection is differentiated from the code that organizes the nodes.

The poll and long_poll methods from the abstract class are used in this example. The Hue Hub sends no event stream, so it must be polled for updates. This is done in the poll method. The long_poll method is utilized to ensure the configuration data is saved consistently. These methods do not need to be manually called anywhere as they are automatically invoked from the run loop every (approximately) 1 second and 30 seconds respectively.

Starting the Node Server

Finally, your program must be able to initialize itself and begin running the node server. In Python, it will very nearly look like this.

```
def main():
    """ setup connection, node server, and nodes """
    poly = PolyglotConnector()
    nserver = HueNodeServer(poly)
    poly.connect()  # begin listening for Polyglot commands
    poly.wait_for_config()  # This is best practice to not start until
                            # Polyglot has begun communicating. This way,
                            # Polyglot will not miss messages sent from
                            # the node server.
    nserver.setup()  # setup method is specific to this example
    nserver.run()  # begin node server run loop


if __name__ == "__main__":
    main()
```

Installing the Node Server

Once all of this has been coded and all the appropriate files (documented in the last section) have been created, the node server directory can be placed in the configuration directory in a subfolder called *node_servers*. Polyglot should then be restarted to trigger the discovery of new node server types. If there is an issue with your node server, it will appear in the log.

Polyglot Node Server API

Documented here is the JSON API used for communication between Polyglot and the Node Server processes. This API will never be referenced directly by either by an end user and will rarely be referenced by a developer. It is documented here for continuity. Nearly each command and its arguments maps to a command and arguments specified in the ISY Virtual Node Server API documentation. The only exceptions are the additions of some commands necessary for Polyglot's operation.

General Format

In general, each API message is formatted as such:

```
{COMMAND: {ARG_NAME_1: ARG_VALUE_1, ..., ARG_NAME_N: ARG_VALUE_N}}
```

All of the arguments are named. Each message ends with a new line and will contain no new lines. Each message will contain only one command. Never will multiple command be sent in the same message.

Node Server STDIN - Polyglot to Node Server

The following messages may be sent from Polyglot to the Node Server to trigger an action inside of the Node Server.

- *{'config': {... arbitrary data saved by the node server ...}}*
  This command is the first one sent to the node server and is only sent once. The arguments dictionary will be of an arbitrary structure and will match what the Node Server had last saved.

- *{'install': {'profile_number': ...}}*
  Instructs the node server to install itself with the specified *profile_number*.

- *{'query': {'node_address': ..., 'request_id': ...}}*
  Instructs the node server to query a node. *request_id* is optional.

- *{'status': {'node_address': ..., 'request_id': ...}}*
  Requests the node server to send current node status to the ISY. *request_id* is optional.

- *{'add_all': {'request_id': ...}}*
  Requests that the node server add all its nodes to the ISY. *request_id* is optional.

- *{'added': {'node_address': ..., 'node_def_id': ..., 'primary_node_address': ..., 'name': ...}}*
  Indicates that the node has been added to the ISY.

- *{'removed': {'node_address': ...}}*
  Indicates that the node has been removed from the ISY.

- *{'renamed': {'node_address': ..., 'name': ...}}*
  Indicates that the node has been renamed in the ISY.

- *{'enabled': {'node_address': ...}}*
  Indicates that the node has been enabled in the ISY.

- *{'disabled': {'node_address': ...}}*
  Indicates that the node has been disabled in the ISY.

- *{'cmd': {'node_address': ..., 'command': ..., *'value': ...., *'uom': ..., *'<pn>.<uomn>': ..., *'request_id': ...}}*
  Instructs the node server to run the specified command on the specified node. *value* and *uom* are optional and described the unnamed parameter. They will always appear together. *<pn>.<uomn>* will be repeated as necessary to described the unnamed parameters. They are also optional. *request_id* is optional.

- *{'ping': {}}*
  This is a command from Polyglot requesting a Pong response. This is handled in the PolyglotConnector class.

- *{'exit': {}}*
  This command is Polyglot instructing the node server to cleanly shut down.

Node Server STDOUT - Node Server to Polyglot

The following messages are accepted by Polyglot from the Node Server and will typically instruct Polyglot to send a response upstream to the ISY.

- *{'config': {... arbitrary data saved by the node server ...}}*
  Sends configuration data to Polyglot to be saved. This data will be sent back to the Node Server, exactly as it has been sent to Polyglot, the next time the Node Server is started.

- *{'install': {}}*
  Install the node server on the ISY. This has not been implemented yet.

- *{'status': {'node_address': ..., 'driver_control': ..., 'value': ..., 'uom': ...}}*
  Reports a node's driver status.

- *{'command': {'node_address': ..., 'command', ..., 'value': ...., 'uom': ..., '<pn>.<uomn>': ...}}*
  Reports that a command has been run on a node. *value* and *uom* are optional and described the unnamed parameter. They will always appear together. *<pn>.<uomn>* will be repeated as necessary to described the unnamed parameters. They are also optional.

- *{'add': {'node_address': ..., 'node_def_id': ..., 'primary': ..., 'name': ...}}*
  Adds a node to the ISY.

- *{'change': {'node_address': ..., 'node_def_id': ...}}*
  Changes the node's definition in the ISY.

- *{'remove': {'node_address': ...}}*
  Instructs the ISY to remove a node.

- *{'request': {'request_id': ..., 'result': ...}}*
  Replies to the ISY indicating that a request has been finished either successfully or unsuccessfully. The result parameter must be a boolean indicating this.

- *{'pong': {}}*
  The proper response to a Ping command. Must be recieved within 30 seconds of a Ping command or Polyglot assumes the Node Server has stalled and kills it. This is handled automatically in the PolyglotConnector class.

- *{'exit': {}}*
  Indicates to Polyglot that the node server has exited and is now closing. This is the last message sent from a node server. All messages following this will be ignored. It is not guaranteed that the node server process will continue to run after this command is sent.

Node Server STDERR - Node Server to Polyglot

STDERR messages have no structured formatting, they are free flowing text. Anything recieved by Polyglot through this stream will not be processed and will be immediately logged as an error. Do not send personal information in error messages as they will always be logged regardless of the log verbosity.

Index

Python Module Index