xs1_api_client Documentation

Release 1.0.0

Markus Ressel

Nov 23, 2017

Contents

A python 3.4+ library for accessing actuator and sensor data on the the EZcontrol® XS1 Gateway using its HTTP API.

CHAPTER 1

Build Status

Master	Beta	Dev

CHAPTER 2

How to use

2.1 Installation

```
pip install xs1-api-client
```

2.2 Usage

For a basic example have a look at the [example.py] file. If you need more info have a look at the [documentation] which should help.

2.3 Basic Example

2.3.1 Create the API Object

The basic way of creating an API object is by providing connection info directly when creating it:

```
from xs1_api_client import api as xs1api
from xs1_api_client import api_constants

# Create an api object with private configuration
api = xs1api.XS1('192.168.2.20', "Username", "Password")
```

This will automatically try to connect to the gateway with the given credentials and retrieve basic gateway information which you can output like this:

```
print("Gateway Hostname: " + api.get_gateway_name())
print("Gateway MAC: " + api.get_gateway_mac())
print("Gateway Hardware Version: " + api.get_gateway_hardware_version())
print("Gateway Bootloader Version: " + api.get_gateway_bootloader_version())
```

```
print("Gateway Firmware Version: " + api.get_gateway_firmware_version())
print("Gateway uptime: " + str(api.get_gateway_uptime()) + " seconds")
```

Now that you have a connection to your gateway we can retrieve its configuration and set or retrieve values of the devices.

2.3.2 Devices

All devices that you have configured in your XS1 are implemented using the `XS1Device` base class which can be found at `/device/base.py`. This class provides basic functionality for every device like getting the **id**, **name**, **type** and other values.

2.3.3 Retrieve Actuators

To retrieve a list of all actuators that are configured use the following call:

```
actuators = api.get_all_actuators()
```

This will return a list of `XS1Actuator` objects which is another base class for all actuators. You can use something like this to print all your actuators:

```
for actuator in actuators:
    print("Actuator " + str(actuator.id()) + ": " + actuator.name() + " (" +␣
↪str(actuator.type()) + ")")
```

There is also an integrated `__str__` method to print out most of the useful properties just like this:

```
for actuator in actuators:
    print(actuator)
```

You can also filter the elements by `enabled` and `disabled` state using:

```
enabled_actuators = api.get_all_actuators(True)
```

Retrieve a single actuator simply by using:

```
actuator_1 = api.get_actuator(1)
```

2.3.4 Retrieve an Actuator Value

To retrieve the current value of an actuator just call:

```
current_value = actuator.value()
```

2.3.5 Set a new Actuator value

To set a new value to this actuator use:

```
actuator.set_value(100)
```

This will send the required request to the XS1 and set the `new_value` property to your value. Most of the time this value is set instantaneously is in sync with the `value` property. However if this value is different from the standard `value` the XS1 gateway is still trying to update the value on the remote device. For some devices this can take up to a couple of minutes (f.ex. FHT 80B heating).

2.3.6 Updating Actuator Information

Currently there is **no callback** when the value is finally updated so **you have to update the device information manually** if you want to get an update on its current state:

```
actuator.update()
```

After that the usual methods like `actuator.value()` will respond with the updated state.

2.3.7 Executing Actuator Functions

If you have defined function presets for a device you can get a list of all functions using:

```
functions = actuator.get_functions()
```

and print them like this:

```
for function in functions:
    print(function)
```

to execute one of the functions type:

```
function.execute()
```

This will (like set_value) update the device state immediately with the gateways response. Remember though that there can be a delay for sending this value to the actual remote device like mentioned above.

2.3.8 Retrieve a List of Sensors

To retrieve a list of all sensors that are configured use the following call:

```
sensors = api.get_all_sensors()
```

Just like with actuators you can filter the elements by `enabled` and `disabled` state using:

```
enabled_sensors = api.get_all_sensors(True)
```

This will return a list of `XS1Sensor` objects which is the base class for all sensors.
You can print basic information about them like this:

```
for sensor in sensors:
    print("Sensor " + str(sensor.id()) + ": " + sensor.name() + " (" + str(sensor.
↪value()) + ")")
```

Just like mentioned above you can also use:

```
for sensor in sensors:
    print(sensor)
```

or:

```
sensor_1 = api.get_sensor(1)
```

to retrieve a specific sensor.

2.3.9 Updating Sensor Information

Just like with actuators there is no automatic updates for sensors either. To get a state update from the XS1 gateway for your sensor object call:

```
sensor.update()
```

After that the complete state of this sensor is updated.

2.3.10 Disabled Devices

The XS1 allows up to 64 actuator and 64 sensor configurations. These 128 device configurations are accessible via the HTTP API at any time - even when there is nothing configured for a specific device id/number.

To check if a device has been configured in the XS1 web interface call:

```
device.enabled()
```

for both actuators and sensors alike.

2.3.11 Get a device configuration

Since version 2.0 it is possible to get and set device configurations on the XS1 using this library.

Please have a look at the `example_config.py` file to get an idea of how to retrieve a device configuration.

2.3.12 Modify a device configuration

Before you proceed

Every configuration change will write to the internal flash memory of the XS1. Please keep in mind that that the use flash memory can and will probably degrade when written too often.

2.3.13 Copy a device configuration

There is a very detailed example in this project called `example_config_copy_actuator.py` that will show you how to copy a device configuration and also explains most of the important configuration parameters you will have to use to set a custom configuration. Keep in mind though that the configuration parameters can vary between device types and systems.

CHAPTER 3

Contributing

Github is for social coding: if you want to write code, I encourage contributions through pull requests from forks of this repository. Create Github tickets for bugs and new features and comment on the ones that you are interested in.

CHAPTER 4

License

```
xs1-api-client by Markus Ressel
Copyright (C) 2017  Markus Ressel

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

CHAPTER 5

Content:

5.1 API

5.1.1 xs1_api_client package

Subpackages

xs1_api_client.device package

Subpackages

xs1_api_client.device.actuator package

Submodules

xs1_api_client.device.actuator.switch module

class xs1_api_client.device.actuator.switch.**XS1Switch**(*state*, *api*)
Bases: *xs1_api_client.device.actuator.XS1Actuator*

Represents a XS1 Switch.

turn_off() → None
Turns off the switch.

turn_on() → None
Turns on the switch.

xs1_api_client.device.actuator.thermostat module

class xs1_api_client.device.actuator.thermostat.**XS1Thermostat**(*state*, *api*)
Bases: *xs1_api_client.device.actuator.XS1Actuator*

Represents a basic XS1 Actuator, there may be special variants for some types.

set_temperature(*temp: float*) → None
Sets the new target temperature of this thermostat

Parameters **temp** – new target temperature

Module contents

class xs1_api_client.device.actuator.**XS1Actuator**(*state*, *api*)
Bases: *xs1_api_client.device.XS1Device*

Represents a basic XS1 Actuator, there may be special variants for some types.

call_function(*xs1_function*)
Calls the specified function by id and saves the api response as the new state

Parameters **xs1_function** – XS1Function object

get_function_by_id(*func_id*)
Get a function by it's id :param func_id: function id :return: XS1Function or None

get_function_by_type(*func_type: xs1_api_client.api_constants.FunctionType*)
Get a function by it's type :param func_type: function type :return: XS1Function or None

get_functions() → []

Returns a list of functions that can be executed using the call_function() method

set_name(*name: str*)
Sets a new name for this device. Keep in mind that there are some limitations for a device name.

Parameters **name** – the new name to set

Returns the new name of the actuator

set_value(*value*) → None
Sets a new value for this actuator

Parameters **value** – new value to set

update() → None
Updates the state of this actuator

class xs1_api_client.device.actuator.**XS1Function**(*actuator: xs1_api_client.device.actuator.XS1Actuator*, *function_id: int*, *function_type: xs1_api_client.api_constants.FunctionType*, *description: str*)
Bases: object

Represents a function of a XS1Actuator.

description() → str

Returns a description for this function

execute() → None
Executes this function and sets the response as the new actuator value

id() → int

Returns the id of this function (note that this id is only unique for a single actuator!)

type() → xs1_api_client.api_constants.FunctionType

Returns the type of this function

xs1_api_client.device.sensor package

Module contents

class xs1_api_client.device.sensor.**XS1Sensor**(*state*, *api*)
Bases: *xs1_api_client.device.XS1Device*

Represents a XS1 Sensor

set_name(*name: str*)
Sets a new name for this device. Keep in mind that there are some limitations for a device name.

Parameters **name** – the new name to set

Returns the new name of the sensor

set_value(*value*) → None
Sets a value for this sensor This should only be used for debugging purpose! :param value: new value to set

update() → None
Updates the state of this sensor

Module contents

class xs1_api_client.device.**XS1Device**(*state: dict*, *api*) → None
Bases: object

This is a generic XS1 device, all other objetcs inherit from this.

NAME_PATTERN_VALID = re.compile('^[a-zA-Z0-9_]+$', re.IGNORECASE)

enabled() → bool

Returns Returns if this device is enabled.

id() → int

Returns id of this device

last_update() → int

Returns the time when this device's value was updated last

name() → str

Returns the name of this device

new_value()
Returns the new value to set for this device. If this value differs from the currrent value the gateway is still trying to update the value on the device. If it does not differ the value has already been set.

Returns the new value to set for this device

set_name(*name: str*)
Sets a new name for this device. Keep in mind that there are some limitations for a device name.

Parameters **name** – the new name to set

Returns the new name of the actuator

set_state(*new_state: dict*) → None
Sets a new state for this device. If there is an existing state, new and old values will be merged to retain any information that was missing from api responses.

Parameters **new_state** – new representation of this device (api response)

set_value(*value*) → None
Sets a new value for this device. This method should be implemented by inheriting classes.

Parameters **value** – the new value to set

type() → str

Returns the type of this device

unit() → str

Returns the unit that is used for the value

update() → None
Updates the current value of this device. This method should be implemented by inheriting classes.

value()

Returns the current value of this device

Submodules

xs1_api_client.api module

This is the main xs1_api_client api which contains the XS1 object to interact with the gateway.

Example usage can be found in the example.py file

class xs1_api_client.api.**XS1**(*host: str = None*, *user: str = None*, *password: str = None*) → None
Bases: object

This class is the main api interface that handles all communication with the XS1 gateway.

RETRY_STRATEGY = Retry(total=5, connect=None, read=None, redirect=None, status=None)

call_actuator_function(*actuator_id*, *function*) → dict
Executes a function on the specified actuator and sets the response on the passed in actuator.

Parameters

- **actuator_id** – actuator id to execute the function on and set response value
- **function** – id of the function to execute

Returns the api response

get_actuator(*actuator_id: int*) → xs1_api_client.device.actuator.XS1Actuator
Get an actuator with a specific id :param actuator_id: the id of the actuator :return: XS1Actuator

get_all_actuators(*enabled: bool = None*) → [<class 'xs1_api_client.device.actuator.XS1Actuator'>]
Requests the list of enabled actuators from the gateway.

Parameters **enabled** –

Returns a list of XS1Actuator objects

get_all_sensors(*enabled: bool = None*) → [<class 'xs1_api_client.device.sensor.XS1Sensor'>]
Requests the list of enabled sensors from the gateway.

Returns list of XS1Sensor objects

get_config_actuator(*actuator_id: int*) → dict

Returns the configuration of a specific actuator

get_config_main() → dict

Returns main configuration of the XS1

get_config_sensor(*sensor_id: int*) → dict

Returns the configuration of a specific sensor

get_gateway_bootloader_version() → str

Returns the bootloader version number of the gateway

get_gateway_firmware_version() → str

Returns the firmware version number of the gateway

get_gateway_hardware_version() → str

Returns the hardware version number of the gateway

get_gateway_mac() → str

Returns the mac address of the gateway

get_gateway_name() → str

Returns the hostname of the gateway

get_gateway_uptime() → str

Returns the uptime of the gateway in seconds

get_list_functions() → list

Returns a list of available functions / actions for actuators

get_list_systems() → list

Returns a list of currently compatible systems

get_protocol_info() → str
Retrieves the protocol version that is used by the gateway

Returns protocol version number

get_sensor(*sensor_id: int*) → xs1_api_client.device.sensor.XS1Sensor
Get a sensor with a specific id :param sensor_id: the id of the actuator :return: XS1Sensor

get_state_actuator(*actuator_id*) → dict
Gets the current state of the specified actuator.

Parameters **actuator_id** – actuator id

Returns the api response as a dict

get_state_sensor(*sensor_id*) → dict
Gets the current state of the specified sensor.

Parameters **sensor_id** – sensor id

Returns the api response as a dict

get_types_actuators() → list

Returns a list of compatible actuators

get_types_sensors() → list

Returns a list of compatible sensors

send_request(*command: xs1_api_client.api_constants.Command*, *parameters: dict = None*) → dict
Sends a GET request to the XS1 Gateway and returns the response as a JSON object.

Parameters

- **command** – command parameter for the URL (see api_constants)
- **parameters** – additional parameters needed for the specified command like 'number=3' passed in as a dictionary

Returns the api response as a json object

set_actuator_value(*actuator_id*, *value*) → dict
Sets a new value for the specified actuator.

Parameters

- **actuator_id** – actuator id to set the new value on
- **value** – the new value to set on the specified actuator

Returns the api response

set_config_actuator(*actuator_id: int*, *configuration: dict*) → dict

Returns the configuration of a specific actuator

set_config_sensor(*sensor_id: int*, *configuration: dict*) → dict

Returns the configuration of a specific actuator

set_connection_info(*host*, *user*, *password*) → None
Sets private connection info for this XS1 instance. This XS1 instance will also immediately use this connection info.

Parameters

- **host** – host address the gateway can be found at
- **user** – username for authentication
- **password** – password for authentication

set_sensor_value(*sensor_id*, *value*) → dict
Sets a new value for the specified sensor. WARNING: Only use this for "virtual" sensors or for debugging!

Parameters

- **sensor_id** – sensor id to set the new value on
- **value** – the new value to set on the specified sensor

Returns the api response

update_config_info() → None
Retrieves gateway specific (and immutable) configuration data

xs1_api_client.api_constants module

XS1 HTTP Web API constants used to create GET request URLs and parse the JSON answer.

class xs1_api_client.api_constants.**ActuatorType**
Bases: enum.Enum

Actuator types

BLIND = 'blind'

DIMMER = 'dimmer'

DISABLED = 'disabled'

DOOR = 'door'

SHUTTER = 'shutter'

SOUND = 'sound'

SUN_BLIND = 'sun-blind'

SWITCH = 'switch'

TEMPERATURE = 'temperature'

TIMERSWITCH = 'timerswitch'

WINDOW = 'window'

class xs1_api_client.api_constants.**Command**
Bases: enum.Enum

XS1 Web API (HTTP) commands

GET_CONFIG_ACTUATOR = 'get_config_actuator'
Command to get the configuration of an actuator

GET_CONFIG_INFO = 'get_config_info'
Command to get (final) configuration information about the XS1

GET_CONFIG_MAIN = 'get_config_main'
Command to get additional configuration information about the XS1

GET_CONFIG_SENSOR = 'get_config_sensor'
Command to get the configuration of a sensor

GET_LIST_ACTUATORS = 'get_list_actuators'
Command to get a list of all actuators

GET_LIST_FUNCTIONS = 'get_list_functions'
Returns a list of available functions / actions for actuators

GET_LIST_RFMODES = 'get_list_rfmodes'
Returns a list of currently active and compatible RF modes of the XS1

GET_LIST_SENSORS = 'get_list_sensors'
Command to get a list of all sensors

GET_LIST_SYSTEMS = 'get_list_systems'
Returns a list of currently compatible systems

GET_PROTOCOL_INFO = 'get_protocol_info'
Command to get information about the protocol version used by the XS1

GET_STATE_ACTUATOR = 'get_state_actuator'
Command to get the state of a specific actuator

GET_STATE_SENSOR = 'get_state_sensor'
Command to get the state of a specific sensor

GET_TYPES_ACTUATORS = 'get_types_actuators'
Retrieves the types of compatible actuators

GET_TYPES_SENSORS = 'get_types_sensors'
Retrieves the types of compatible sensors

SET_CONFIG_ACTUATOR = 'set_config_actuator'
Command to set the configuration of an actuator

SET_CONFIG_SENSOR = 'set_config_sensor'
Command to set the configuration of a sensor

SET_STATE_ACTUATOR = 'set_state_actuator'
Command to set a new value on an actuator

SET_STATE_SENSOR = 'set_state_sensor'
Command to set a new value on a sensor (for debugging)

class xs1_api_client.api_constants.**ErrorCode**
Bases: enum.Enum

Error codes

CMD_TYPE_MISSING = '02'
Error code for 'cmd type missing'

DUPLICATE = '04'
Error code for 'duplicate name'

INVALID_COMMAND = '01'
Error code for 'invalid command'

INVALID_DATE_TIME = '07'
Error code for 'invalid date/time'

INVALID_FUNCTION = '06'
Error code for 'invalid function'

INVALID_SYSTEM = '05'
Error code for 'invalid system'

INVALID_TIME_RANGE = '11'
Error code for 'error time range'

NOT_FOUND = '03'
Error code for 'number/name not found'

OBJECT_NOT_FOUND = '08'
Error code for 'object not found'

PROTOCOL_VERSION_MISMATCH = '12'
Error code for 'protocol version mismatch'

SYNTAX_ERROR = '10'
Error code for 'syntax error'

TYPE_NOT_VIRTUAL = '09'
Error code for 'type not virtual'

static get_message(*error_code*) → str

class xs1_api_client.api_constants.**FunctionType**

Bases: enum.Enum

Function types

ABSOLUT = 'absolut'

AUTO = 'auto'

BLIND_ABS = 'blind_abs'

BUTTON_LONG = 'button_long'

BUTTON_SHORT = 'button_short'

DIM_ABSOLUT = 'dim_absolut'

DIM_DOWN = 'dim_down'

DIM_UP = 'dim_up'

DISABLED = 'disabled'

IMPULS = 'impuls'

LEARN = 'learn'

LONG_OFF = 'long_off'

LONG_ON = 'long_on'

MANUAL = 'manual'

OFF = 'off'

OFF_WAIT_OFF = 'off_wait_off'

OFF_WAIT_ON = 'off_wait_on'

ON = 'on'

ON_WAIT_OFF = 'on_wait_off'

ON_WAIT_ON = 'on_wait_on'

RELATIVE = 'relative'

SPECIAL = 'special'

STOP = 'stop'

TOGGLE = 'toggle'

UNKNOWN = 'unknown'

WAIT = 'wait'

class xs1_api_client.api_constants.**Node**

Bases: enum.Enum

JSON API nodes

ACTUATOR = 'actuator'
Node with an array of actuators

DEVICE_BOOTLOADER_VERSION = 'bootloader'
Bootloader version number

DEVICE_FIRMWARE_VERSION = 'firmware'
Firmware version number

DEVICE_HARDWARE_VERSION = 'hardware'
Hardware revision

DEVICE_MAC = 'mac'
MAC address

DEVICE_NAME = 'devicename'
Hostname

DEVICE_UPTIME = 'uptime'
Uptime in seconds

ERROR = 'error'
Node containing the error code

FUNCTION = 'function'
Node with an array of functions

INFO = 'info'
Node with gateway specific information

PARAM_DESCRIPTION = 'dsc'
Device description

PARAM_FUNCTION = 'function'
Array of functions

PARAM_ID = 'id'
Device id (only unique within actuators/sensors)

PARAM_NAME = 'name'
Device name

PARAM_NEW_VALUE = 'newvalue'
New value to set for the device

PARAM_NUMBER = 'number'
Alternative device id (only unique within actuators/sensors)

PARAM_TYPE = 'type'
Device type

PARAM_UNIT = 'unit'
Device value unit

PARAM_UTIME = 'utime'
Time this device was last updated

PARAM_VALUE = 'value'
Current device value

SENSOR = 'sensor'
Node with an array of sensors

SYSTEM = 'system'
Node with gateway main configuration

VERSION = 'version'
Node with protocol version info

class xs1_api_client.api_constants.**SensorType**

Bases: enum.Enum

Sensor types

AIR_QUALITY = 'air_quality'

ALARMMAT = 'alarmmat'

BAROMETER = 'barometer'

COUNTER = 'counter'

COUNTERDIFF = 'counterdiff'

DISABLED = 'disabled'

DOORBELL = 'doorbell'

DOOROPEN = 'dooropen'

FENCEDETECTOR = 'fencedetector'

GAS_BUTAN = 'gas_butan'

GAS_CO = 'gas_co'

GAS_CONSUMP = 'gas_consump'

GAS_METHAN = 'gas_methan'

GAS_PEAK = 'gas_peak'

GAS_PROPAN = 'gas_propan'

HEATDETECTOR = 'heatdetector'

HYGROMETER = 'hygrometer'

LEAFWETNESS = 'leafwetness'

LIGHT = 'light'

LIGHTBARRIER = 'lightbarrier'

MAIL = 'mail'

MOTION = 'motion'

OIL_CONSUMP = 'oil_consump'

OIL_PEAK = 'oil_peak'

OTHER = 'other'

PRESENCE = 'presence'

PWR_CONSUMP = 'pwr_consump'

PWR_PEAK = 'pwr_peak'

PYRANOMETER = 'pyranometer'

RAIN = 'rain'

RAININTENSITY = 'rainintensity'

RAIN_1H = 'rain_1h'

RAIN_24H = 'rain_24h'

REMOTECONTROL = 'remotecontrol'

SMOKEDETECTOR = 'smokedetector'

SOILMOISTURE = 'soilmoisture'

SOILTEMP = 'soiltemp'

TEMPERATURE = 'temperature'

UV_INDEX = 'uv_index'

WATERDETECTOR = 'waterdetector'

WATERLEVEL = 'waterlevel'

WINDDIRECTION = 'winddirection'

WINDGUST = 'windgust'

WINDOWBREAK = 'windowbreak'

WINDOWOPEN = 'windowopen'

WINDSPEED = 'windspeed'

WINDVARIANCE = 'windvariance'

WTR_CONSUMP = 'wtr_consump'

WTR_PEAK = 'wtr_peak'

class xs1_api_client.api_constants.**SystemType**

Bases: enum.Enum

An enumeration.

AB400 = 'ab400'

AB500 = 'ab500'

AB601 = 'ab601'

BGJ = 'bgj'

BSQUIGG = 'bsquigg'

EM = 'em'

FA20RF = 'fa20rf'

FC1 = 'fc1'

FHT = 'fht'

FS10 = 'fs10'

FS20 = 'fs20'

HE = 'he'

HMS = 'hms'

IT = 'it'

IT2 = 'it2'

MARMI = 'marmi'

OASEFM = 'oasefm'

REV = 'rev'

RGBLED1 = 'rgbled1'

RS200 = 'rs200'

RS862 = 'rs862'

RSL = 'rsl'

SCHALK = 'schalk'

VENT831 = 'vent831'

VIRTUAL = 'virtual'

WAREMA = 'warema'

WMR200 = 'wmr200'

WS300 = 'ws300'

WS433 = 'ws433'

xs1_api_client.api_constants.**UNIT_BOOLEAN = 'boolean'**
Boolean unit type

class xs1_api_client.api_constants.**UrlParam**
Bases: enum.Enum

URL parameters

ADDRESS = 'address'

COMMAND = 'cmd'
command parameter that specifies the method the api is queried with

FACTOR = 'factor'

FUNCTION = 'function'
parameter that specifies the function to execute (on an actuator)

FUNCTION1_DSC = 'function1.dsc'

FUNCTION1_TYPE = 'function1.type'

FUNCTION2_DSC = 'function2.dsc'

FUNCTION2_TYPE = 'function2.type'

FUNCTION3_DSC = 'function3.dsc'

FUNCTION3_TYPE = 'function3.type'

FUNCTION4_DSC = 'function4.dsc'

FUNCTION4_TYPE = 'function4.type'

INITVALUE = 'initvalue'

LOG = 'log'

NAME = 'name'

NUMBER = 'number'
number parameter that specifies the id of an actuator or sensor

OFFSET = 'offset'

PASSWORD = 'pwd'
'Password' parameter

SYSTEM = 'system'

TYPE = 'type'

USER = 'user'
'User' parameter

VALUE = 'value'
'value' parameter that specifies the new value to set an actuator (or sensor) to

xs1_api_client.api_constants.**VALUE_DISABLED = 'disabled'**
'Disabled' type

Module contents

Python Module Index

x

Index

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X