domogik-plugin-notify

Release 0.1

Jun 26, 2018

Contents

CHAPTER 1

Plugin documentation

1.1 Last change

New instructions to apply changes from immediatly previous version.

- **0.3.0** [(07-09-2016) First published version for domogik 0.5]
  - **Version without xpl, work with 0MQ**
    - Update device_type, recreate domogik device with them.
- Previous change

1.2 Purpose

This Domogik plugin send notification message through services that could SMS (from web), Newtifry for Android.

Used with scenario could inform you of house event.

Offer a simply class developpement to add service.

1.3 Plugin configuration

1.3.1 Configuration

In Domogik administration section, go to client plugin-notify details page.

Key	Default value	Description
startup-plugin	false	Automatically start plugin at Domogik startup
msg_header	Domogik notification	Define header of all messages.
send_at_start	false	Send a notification message to all notify clients at starting plugin

1.3.2 Creating devices for SMS Client

In clients page of admin UI, go to **plugin-notify-<your_host_domogik>**, select tab **"Devices"**, **"New"** to create your devices.

Chose one way creation by product or device type.

device_types : notify.smsweb

Key	Example	Description
Device	My phone id	The display name for this device.
Description	What you want	A short description for this device.
Reference	What you want	A reference for this device.
Global to	My Phone number	Phone number to send SMS.
Global operator	Freemobile service	**Choice :** • Bouygues service • Freemobile service. • Orange service. • SFR service.
Global login	My user login	User login service.
Global pwd	My password	User password service.

Instance-type : notify.newtifry

Configure your Newtifry acces

CHAPTER 2

Development informations

2.1 How it work ?

The plugin use clients who send message through operator service. Operator could be any web service or hardware `(in future stuff)` Clients service are register by plugin and communicat with they by internal code. Header defined in plugin configuration is added at the top of message.

2.2 xPL messages

No xPL message handle, use only 0MQ

2.3 Existing clients service.

- **Sending SMS WEB:** Send SMS using web service phone operator, actually there is 4 french operators handled : Can use just optional value `title`
  - **Bouygues Telecom** [file `bin/smsweb_bouygues.py` who use web service on] http://www.mobile.service.bbox.bouyguestelecom.fr/services/SMSIHD/sendSMS.phtm
  - **Free mobile** [file `bin/smsweb_freemobile.py` who use web service on] https://smsapi.free-mobile.fr/sendmsg?
  - **Orange** [file `bin/smsweb_orange.py` who use web service on] http://smsmms1.orange.fr/C/Sms/sms_write.php
  - **SFR** [file `bin/smsweb_sfr.py` who use web service on] http://www.sfr.fr/xmscomposer/index.html?todo=compose
- **Sending Newtifry for Android:** Send notification using web service phone Newtifry API Can use all optional value `title`, `priority`, `url`, `image`

2.4 Developing new notification service

Plugin offer way to add new client service.

3 step are needed.

2.4.1 1 - Editin JSON file plugin

Some adding must be necessary to do in info.json file:

- For an SMS web service just add your operator id in `parameters`, `operator`, key `choices` of `device_types`, instance `notify.smsweb`:

```
......
"notify.smsweb" : {
        "description" : "Send SMS using web service phone operator.",
        "id" : "notify.smsweb",
        "name" : "SMS operator web service",
        "commands" : ["send_msg"],
        "sensors" : ["msg_status", "error_send"],
        "parameters" : [{
                "key" : "to",
                                    "xpl": false,
                "description" : "Phone number to send SMS.",
                "type" : "string"
            },{
                "key" : "operator",
                                    "xpl": false,
                "description" : "Operator service.",
                "type" : "enum",
                "choices" : {
                    "Bouygues_sms-web" : "Bouygues service",
                    "Freemobile_sms-web" : "Freemobile service",
                    "Orange_sms-web" : "Orange service",
                    "SFR_sms-web" : "SFR service", # add coma
                    "My_NEW_OPERATOR" : "NEW service"  # add ligne
                    }
            },{
                "key" : "login",
                                    "xpl": false,
                "description" : "User login service.",
                "type" : "string"
            },{
                "key" : "pwd",
                                    "xpl": false,
                "description" : "User password service.",
                "type" : "string"
            }
        ]
    },
    .....
```

- For a new service or hardware, create your own device_types

```
"notify.mynewservice" : {
        "description" : "A simple description of service.",
```

(continues on next page)

(continued from previous page)

```
        "id" : "notify.mynewservice",
        "name" : "Name service",
        "commands" : ["send_extendmsg or send_msg"], # chose type of message
        "sensors" : ["msg_status", "error_send"],
        "parameters" : [{
                "key" : "to",                       # keep this key,␣
↪identification of recipient
                                    "xpl": false,                       # set to␣
↪false for don't use in xPL message
                "description" : "A simple description of recipient.",
                "type" : "string"
            },{
                "key" : "key1",                     # a optional key needed by␣
↪service, like user/login/....
                                    "xpl": false,                       # set to␣
↪false for don't use in xPL message
                "description" : "A simple description of function.",
                "type" : "string"
            },{
                "key" : "key2",                     # a optional key needed by␣
↪service, like password/....
                                    "xpl": false,                       # set to␣
↪false for don't use in xPL message
                "description" : "A simple description of function.",
                "type" : "string"
            },{
                .....
            }
        ]
    }
},
```

2.4.2 2 - Editing clients_devices.py file

File `lib/clients_devices.py` contains a `BaseClientService` python class who handle basic methodes. You create a new class inherit this base class and overwriting some methodes to get acces of this new service or hardware. There is 3 thinks to do in this file :

- **1st** [Add your new operator id in OPERATORS_SERVICE global variable.] This id is the same than JSON file

```
OPERATORS_SERVICE =  ['SFR_sms-web', 'Orange_sms-web', 'Bouygues_sms-web',
↪'Freemobile_sms-web', 'Newtifry', My_NEW_OPERATOR]
```

- 2nd : You have to add new client device class reference in `CreateOperator` methode.

```
def CreateOperator(params, log = None):
    """ Create a device depending of operator, use instance class to get␣
↪parameters.
        - Developer : add your python class derived from DeviceBase class."""
    ....
    elif params['operator'] == 'My_new_service' :
        from domogik_packages.plugin_notify.lib.mynewcodefile import MyNewClient
    newOperator = MyNewClient(params,  log)
    ....
```

- 3rd : If you have ceated a new instance_type, you have to add new client getting specifique parameters in `GetDeviceParams` methode.

```
def GetDeviceParams(Plugin, device):
    """ Return all internal parameters depending of instance_type.
        - Developer : add your instance_type proper parameters
            @param Plugin : Plugin base class reference for "get_parameter" methods access.
                type : object class Plugin
            @param device :  domogik device data.
                type : dict
            @return : parameters for creating or update ClientService object.
                    Value must contain at least keys :
                        - 'operator' = Selected from OPERATORS_SERVICE
                        - 'to' = Client reference, the same as global parameter 'to'
                type : dict
    """
    ...
    elif device['device_type_id'] == 'newtifry.device':
        operator = 'My_NEW_OPERATOR'
        id = device['parameters']['to']['value']
              # keep this line
        valueKey1 = device['parameters']['Key1']['value']
              # Get your new device paramaters
        valuekey2 = device['parameters']['Key2']['value']
              # Get your new device paramaters
        valuekey3 = Plugin.get_parameter(device, 'defaulttitle')
              # Get a Plugin config paramaters if needed
        if operator and device["name"] and id and valueKey1 and valuekey3:
              # check for none optional parameters
                # Build a dict with parameters
            params = {'name': device["name"] , 'operator' : operator, 'to' : id, 'key1' : valueKey1, 'key2' : valuekey2, 'key3' : valuekey3}
            return params
                # return your own parameters
    ...
```

2.4.3 Creating new client class

In a new python file place in lib :

- Import BaseClientService from plugin lib

```
from client_devices import BaseClientService
```

- Create your python class

```
class My_New_Client(BaseClientService):  # simply new class declaration
    def send(self, message):  # overwrite methodes class
    ......
```

2.4.4 Overwrite `update` methode

This method is called by plugin to update proper parameters, so set your own parameters in:

```
def update(self, params):
    """ Create or update internal data, must be overwrited if others params needed.
        @param params :  parameters from GetDeviceParams() of domogik device'.
            type : dict
    """
    BaseClientService.update(self, params)                    # keep this line to␣
↪call basic method
    self.key1 = params['key1']                                # your parameter
    self.key2 = params['key2'] if 'key2' in params else None  # your parameter
```

2.4.5 Overwrite `send` methode

This method is called by plugin for sending message on service, She must return a dict with format : use self._log to log message in plugin log.

```
{
    'status' : <A simple text to say sended or not>,
    'error' : <Error cause or empty ('') if no error>
}
```

```
def send(self, message):
    """ Send message
        @param message : message dict data contain at least keys:
            - 'to' : recipient of the message
            - 'header' : header for message set in plugin configuration
            - 'body" : message
            - extra key defined in 'command' json declaration like 'title', priority',
↪ ....
        @return : dict = {'status' : <Status info>, 'error' : <Error Message>}
    """
    ...
    # write your own code for format message and send it.
    # using try / except should avoid failling plugin
    try :
      ...
    except :
        result = {'status': u"Message not sended", 'error':  u"Bad message format :
↪{0}".format(message)}
        self._log.warning(u"{0} Message <{1}> not sended. {2}".format(self.to,␣
↪message, traceback.format_exc()))  # use traceback in lo
    ...
    if message_Is_Sended : return {'status': 'Message sended :)', 'error': ''}
    else return {'status': 'Message not sended :(', 'error': 'Server not response ....
↪.'}
```

CHAPTER 3

0.3.2

- Freemobile sms service : - Fix UrlLib2 ssl CERTIFICATE_VERIFY_FAILED error 590, due to python lib update (2018 june) - Improve Error connection service.
- Fix #9 Add send at stop configuration option
- Fix #8 Unicode for device name (log)

CHAPTER 4

0.3.1

- Fix Unicode for device name

CHAPTER 5

0.3.0

- domogik 0.5 upgrade
- Version without xpl, work with 0MQ
- Update device_type, recreate domogik device with them.
- Add disconnect notification
- Live domogik device create/update, no need to restrat plugin
- Update user and develop doc.

CHAPTER 6

0.2a0

- Fusion of sms and apushnot plugin
- domogik 0.4 upgrade, add freemobile operator and Newtifry for android.

CHAPTER 7

0.1.2

- Initial version from domogik 0.3 plugin