



Input & Output design of plugin

Jerry Chae & Irene Cho



Contents

- **Module Context (Main Class of plugin)** Pg.3
- **Input Design** Pg.4
- **Input Design and STU property (examples)** Pg.5 ~ Pg.7
- **Output design** Pg.8
- **Plugin return** Pg.9
- **PAM & plugin** Pg.10
- **Output Design (examples)** Pg.11 ~ Pg.12
- **Argparse** Pg.13
- **Parameters of add_argument** Pg.14 ~ Pg.20



Module Context (Main Class of plugin)

- Inherited from argparse.ArgumentParser module
- keyword parameters
 - owner : define owner
 - group : which group, usually second level of package
 - version : (just info, real python module version is from setup.yaml file)
 - platform : supported OS platform ('windows', 'darwin' for mac, 'linux')
 - output_type : reserved
 - display_name : showed label in STU

```
with ModuleContext(  
    owner='ARGOS-LABS-DEMO',  
    group='Roo man',  
    version='1.0',  
    platform=['windows', 'darwin', 'linux'],  
    output_type='text',  
    display_name='Whatsapp Roo man',  
    icon_path=get_icon_path(__file__),  
    description='send message whatsapp friends',  
) as mcxt:  
    # ##### for app dependent parameters  
    mcxt.add_argument('msg',  
                      display_name='message',  
                      help='type message to send')  
    mcxt.add_argument('friends', nargs='+',  
                      display_name='Friends Name',  
                      help='type your friends name')  
    # ##### for app dependent options  
    mcxt.add_argument('--driver',  
                      choices=['Chrome', 'Firefox', 'MSIE'],  
                      display_name='Chromedriver path',  
                      help='type Chromedriver path')  
    mcxt.add_argument('--filepath',  
                      display_name='send file',  
                      help='give path of img/video')  
argspec = mcxt.parse_args(args)  
return whatsapp(mcxt, argspec)
```



Input Design

- All user input is added by `add_argument()` method
- Mandatory argument is parameter (not starting with `'-'`)
- Optional arguments are options (starting with `'-'`)

```
with ModuleContext(  
    owner='ARGOS-LABS-DEMO',  
    group='Roooman',  
    version='1.0',  
    platform=['windows', 'darwin', 'linux'],  
    output_type='text',  
    display_name='Whatsapp Roooman',  
    icon_path=get_icon_path(__file__),  
    description='send message whatsapp friends',  
) as mcxt:  
    # ##### for app dependent parameters  
    mcxt.add_argument('msg',  
                      display_name='message',  
                      help='type message to send')  
    mcxt.add_argument('friends', nargs='+',  
                      display_name='Friends Name',  
                      help='type your friends name')  
    # ##### for app dependent options  
    mcxt.add_argument('--driver',  
                      choices=['Chrome', 'Firefox', 'MSIE'],  
                      display_name='Chromedriver path',  
                      help='type Chromedriver path')  
    mcxt.add_argument('--filepath',  
                      display_name='send file',  
                      help='give path of img/video')  
argspec = mcxt.parse_args(args)  
return whatsapp(mcxt, argspec)
```



Input Design and STU property (example 1)

```
# ##### for app dependent options
mcxt.add_argument('--port',
                  display_name='Port',
                  type=int, default=22,
                  help='port number, default is [[22]]')
mcxt.add_argument('--password', show_default=True,
                  display_name='Password',
                  input_method='password',
                  help='user password')
mcxt.add_argument('--key-filename',
                  display_name='SSH keyfile',
                  help='SSH key filename instead of password')
mcxt.add_argument('--recursive', action='store_true',
                  display_name='Recursive',
                  help='If this flag is set get or put recursively '
                       'including all files and sub folders')
mcxt.add_argument('--preserve-times', action='store_true',
                  display_name='Preserve Times',
                  help='If this flag is set preserve file time attribute')
# ##### for app dependent parameters
mcxt.add_argument('host',
                  display_name='SSH Host',
                  help='hostname or ip address to connect')
mcxt.add_argument('user',
                  display_name='SSH User',
                  help='user id to connect')
mcxt.add_argument('op',
                  display_name='Operation',
                  choices=['Get', 'Put'],
                  help='One of Get or Put from remote file system')
mcxt.add_argument('remote',
                  display_name='Remote File/Folder',
                  help='Remote file or folder')
mcxt.add_argument('local',
                  display_name='Local File/Folder',
                  input_method='fileread',
                  help='Remote file or folder')
```

The screenshot shows the ARGOS LABS interface for configuring a plugin. The top section displays the plugin details: "Plugin" dropdown set to "(P)SSH Copy", "Operation name" set to "Operation 1", and "Plugin version" set to "2.412.2130". Below this, the "Properties" section contains the following settings:

SSH Host	10.211.55.57
SSH User	toor
Operation	Put
Remote File/Fol...	/tmp/hosts.txt
Local File/Folder	V:\Bots\SCP\hosts.txt

At the bottom, there is a "Advanced" section with several checkboxes:

- Password
- Port (value: 22)
- SSH keyfile
- Recursive
- Preserve Times



Input Design and STU property (example 2)

```
# ##### for app dependent options
mcxt.add_argument('--data', action='append', display_name='Input data',
                  help='data to create or insert')
#
mcxt.add_argument('--jsonfile', display_name='Json file',
                  input_method='fileread',
                  help='Json inputfile')
#
mcxt.add_argument('--id', display_name='Data Id',
                  help='id for data')
#
mcxt.add_argument('--fieldnames', action='append',
                  display_name='Select fields',
                  help='select query fieldnames')
#
mcxt.add_argument('--value', display_name='Search value',
                  help='search specific value and return Id')

# ##### for app dependent parameters
mcxt.add_argument('op2',
                  display_name='Items', choices=SimpleAPI.OP_TYPE2,
                  help='Simple salesforce class type')
#
mcxt.add_argument('op',
                  display_name='Operations', choices=SimpleAPI.OP_TYPE,
                  help='Simple salesforce type of operation')
#
mcxt.add_argument('username', display_name='Username', help='username')
#
mcxt.add_argument('password', display_name='Password',
                  input_method='password', help='password')
#
mcxt.add_argument('security_token', display_name='Security token',
                  input_method='password', help='security_token')
```

The screenshot shows the ARGOS LABS interface for configuring a plugin operation. The top section displays the plugin details: **Plugin** (P)Simple-SFDC, **Operation name**: Operation 1, and **Plugin version**: 2.504.730. Below this, the **Properties** section is highlighted with a red box and contains fields for **Items**, **Operations**, **Username**, **Password**, and **Security token**. A blue box highlights the **Advanced** section, which lists five input types with corresponding input fields: **Input data**, **Json file**, **Data Id**, **Select fields**, and **Search value**.



Input Design and STU property (example 3)

```
# ##### for app dependent options
mcxt.add_argument('--port',
                  display_name='Port',
                  type=int, default=22,
                  help='port number, default is [[22]]')
mcxt.add_argument('--password',
                  display_name='Password',
                  input_method='password',
                  help='user password')
mcxt.add_argument('--key-filename',
                  display_name='SSH keyfile',
                  help='SSH key filename')
mcxt.add_argument('--connect-timeout',
                  display_name='Connect timeout',
                  type=int, default=10,
                  help='connection timeout, default is [[10]]')
mcxt.add_argument('--expect-timeout',
                  display_name='Prompt Expect timeout',
                  type=int, default=60,
                  help='expectation string waiting timeout, default is [[60]]')
mcxt.add_argument('--echo-display', action='store_true',
                  display_name='Echo On',
                  help='If this flag is set echo all command output')
mcxt.add_argument('--display-index', nargs='+', type=int,
                  display_name='Display index',
                  help='1-based index to display the output from command list')
# ##### for app dependent parameters
mcxt.add_argument('host',
                  display_name='SSH Host',
                  help='hostname or ip address to connect')
mcxt.add_argument('user',
                  display_name='SSH User',
                  help='user id to connect')
mcxt.add_argument('prompt',
                  display_name='Prompt RegExp',
                  help='prompt to expected')
mcxt.add_argument('command', nargs='+',
                  display_name='Commands at terminal',
                  help='one or more command to execute')
```

The screenshot shows the ARGOS LABS interface for configuring an SSH Command operation. The operation is named "Operation 3" and is using version 1.415.1930 of the plugin. The configuration includes:

- Properties:** SSH Host, SSH User, Prompt RegEx, Commands at t... (with a plus button to add more).
- Advanced:** Port (set to 22), Password, SSH keyfile, Connect timeout (set to 10), Prompt Expect ... (set to 60), Echo type, and Display index (set to 0).

A blue oval highlights the "Properties" section, and a red oval highlights the "Advanced" section.



Output design

- Output for plugin is the output stream to stdout
 - `print('abc')`
 - `sys.stdout.write('d,e,f')`
- If plugin has error or exception output stream to stderr
 - `sys.stderr.write('Error:...')`
- PAM interprets the output stream as one of these
 - String into variable
 - CSV with variable group
 - File path to save the result and variable for file

The image displays three separate configuration dialog boxes, likely from a software interface, illustrating different ways PAM can interpret output streams:

- String into variable:** Shows "Result type" set to "String" and "Variable name" set to "{{my.a}}".
- CSV with variable group:** Shows "Result type" set to "CSV" and "Variable group name" set to "group". There is also a checked checkbox for "First row contains column header".
- File path to save the result and variable for file:** Shows "Result type" set to "File" and "File Name" set to "V:\Bots\SCP\hosts.txt". It also shows "Variable for file" set to "{{my.b}}".



Plugin return

- Return 0 for success
- Return non 0 for failure

```
#####
@func_log
def xml_extract(mcxt, argspec):
    """
    plugin job function
    :param mcxt: module context
    :param argspec: argument spec
    :return: True
    """
    mcxt.logger.info('>>>starting...')

    try:
        r = get_xpath_data(argspec.xml, argspec.xpath,
                           include_header=argspec.header)
        return 0 if r > 0 else 2
    except Exception as err:
        msg = str(err)
        mcxt.logger.error(msg)
        sys.stderr.write('%s%s' % (msg, os.linesep))
        return 1
    finally:
        sys.stdout.flush()
        mcxt.logger.info('>>>end...')
```



PAM & plugin

- When PAM executes plugins in a different way from the native operations
- PAM executes plugins as a subprocess with user properties and advanced properties (same concept as CLI)
- PAM checks if the result of plugin is 0 or not
 - if 0 then stdout output stream will be processed one of String, CSV, File
 - if not 0 then stop the bot and stderr error stream will be showed up as error result (Error message must be kind plain English)



Output Design (example 1)

```
print(os.path.abspath(argspec.out_file), 'end='')
```

The screenshot shows the ARGOS LABS interface for configuring a plugin operation. The configuration panel includes the following fields:

- Plugin: (P)pandas I
- Operation name: Operation 1
- Plugin version: 2.429.1740
- In file: V:\Bots\PANDAS\bar.csv
- Out file: V:\Bots\PANDAS\bar.out.csv
- Filters: df['Units'].str.startswith('I')
df['Value'] > 1000000
- Select Range, e...: 1:[2,5,8]
- Advanced
- Return value:
 - Result type: String
 - Variable name: {{my.a}}



Output Design (example 2)

```
# =====
def sql_select(self, sql):
    self.logger.debug('MySQL.sql_select: sql=<%s>' % sql)
    with self.conn.cursor() as cursor:
        cursor.execute(sql)
        # num_fields = len(cursor.description)
        field_names = [x[0] for x in cursor.description]
        # noinspection PyUnusedLocal
        field_types = [x[1] for x in cursor.description]
        print('%s' % ', '.join(field_names))
        cwr = csv.writer(sys.stdout, lineterminator='\n')
        rows = cursor.fetchall()
        for row in rows:
            cwr.writerow(row)
```

The screenshot shows the configuration interface for a MySQL plugin operation. The 'Plugin' dropdown is set to '(P)SQL'. The 'Operation name' is 'Operation 1'. The 'Plugin version' is '2.228.1800'. Under 'Properties', the 'RDB type' is 'mysql', 'DB Host' is '192.168.99.249', 'DB Port' is '0', 'DB User' is 'root', 'DB Password' is masked, 'dbname' is 'mysql', and 'SQL string' is 'select * from mytable'. There are also options for 'SQL file' and 'Advanced'. A red box highlights the 'Return value' section, which includes 'Result type' (set to 'CSV'), 'Variable group name' ('cols'), and a checked checkbox for 'First row contains column header'. The 'Advanced' section is partially visible below the 'Return value' section.



Argparse

- What is Argparse?
 - Argparse is a python module that gets command line arguments into one's program
 - Get the required spec using add_argument method
- The add_argument() method
 - Define how a single command-line argument should be parsed
 - Distinguished by an optional argument, like -f or -foo, or a positional argument, also called dependent options, like a list of filenames
 - Positional arguments are shown as default parameters in STU and optional argument are located under 'Advanced' section



Parameters of add_argument

Parameters	Type	Usage	Example
name or flags	str	A name of a list of option strings	foo (positional argument) --foo (optional argument)
action	Store	Store the argument's value (default action)	parser.add_argument('--foo')
	Append	Store a list and appends each argument value to a list	parser.add_argument('--foo', action='append')
action	Count	Count the number of times a keyword argument occurs	parser.add_argument('--verbose', '-v', action='count', default=0)
	Store_true	Special case of 'store_const' used for storing	parser.add_argument('--foo', action='store_true')
	Store_false	the values True or False	



Parameters of add_argument

Parameters	Type	Usage	Example
	Default	Relate a different number of command-line arguments to a single action	
nargs	N (an integer)	Gather the N arguments into a list	<code>parser.add_argument('--foo', nargs=2)</code>
	?	<ul style="list-style-type: none">Consume one argument and produce a single itemPrint value from ‘default’ if there is no argumentIn optional argument, the value from ‘const’ will be printed if there is no argument	<code>parser.add_argument('--foo', nargs='?', const='c', default='d')</code>
	*	Gather all command-line arguments present into a list (recommend one positional argument)	<code>parser.add_argument('-bar', nargs='*)</code>
	+	Same as '*' but it will give an error message if there wasn't at least one argument	<code>parser.add_argument('foo', nargs='+')</code>



Parameters of add_argument

Parameters	Type	Usage	Example
const	value	Hold constant values that are not read from the command line, but required for the various actions	<code>parser.add_argument('--foo', const=2)</code>
default	value	Specify a value if the command-line argument is not present	<code>parser.add_argument('--foo', default=2)</code>
type	Str	Allow any necessary type-checking and type conversions to be performed	
	Float	Float	
	Int	Int	<code>parser.add_argument('--foo', type=str)</code>



Parameters of add_argument

Parameters	Type	Usage	Example
choices	list of str	Pass a container object and allows the arguments to be selected from a restricted set of values	<code>parser.add_argument('foo', choices=['rock', 'peer', 'scissors'])</code>
help	str	A string which contains a brief description of the argument	<code>parser.add_argument('--foo', help='foo the bars before frobbing')</code>
dest	str	Determine the name of the attribute added by ArgumentParser actions	<code>>parser.add_argument('bar')</code> <code>>parser.add_argument('bar')</code>



Parameters of add_argument

Parameters	Type	Usage	Example
input_method	password	The way for the user to securely enter a password in STU	parser.add_argument('foo', input_method=password)
	fileread	Select the file which stored on the user's computer in STU	parser.add_argument('foo', input_method=fileread)
display_name	str	Display the name of a parameter in STU	parser.add_argument('foo', display_name='Foo')
show_default	True/False	If show_default=True, the parameter will always be shown as a default value	parser.add_argument('-- foo', show_default=True)



Parameters of add_argument

Parameters	Type	Usage	Example
min_value or greater_eq	value	Check if value is greater than or equal otherwise error	parser.add_argument('foo', type=int, min_value=1)
max_value or less_eq	value	Check if value is less than or equal otherwise error	parser.add_argument('foo', max_value=100)
min_value_ni or greater	value	Check if value is greater than otherwise error	parser.add_argument('foo', greater=0)
max_value_ni or less	value	Check if value is less than otherwise error	parser.add_argument('foo', less='zz')



Parameters of add_argument

Parameters	Type	Usage	Example
equal	value	Check if value is equal otherwise error	<code>parser.add_argument('foo',equal='abc')</code>
not_equal	value	Check if value is not equal otherwise error	<code>parser.add_argument('foo',not_equal=55)</code>
re_match	str	Check if str is matching with regular expression (example: '123.4.56.1')	<code>parser.add_argument('foo', re_match=r'^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\$')</code>



Happy Automation!