Task

Task Class

The task class determines the action that will be executed when the task runs. FLUX supports the following task classes:

1. **CLI Operations**: Establishes a connection with a device and sends model commands via console
2. **REST/HTTP**: Sends model data (typically JSON format) via HTTP
3. **SOAP/HTTP**: Sends model data via HTTP
4. **Notifications**: Sends a custom notification with the model as the message body

Management Script

The Management Script adds extra data to the execution context that can be referenced within models.

For detailed code documentation, see Visual Workflow Designer Scripts.

Special Variables by Task Class

Notifications Task

Use these context variables to configure notification parameters:

```
context['email_destination'] = 'devops@zequenze.com'
context['email_subject'] = 'Flow notification'
```

Simulating Automation Model Output

You can set the model output by retrieving the last configuration:

```
context['automation_custom_output'] = config.get_last_config()
```

Automation Model

The Automation Model is used to build the messages or commands that will be sent to the target element.

Processing Model

The Processing Model is used to process and parse the element's response to the executed message or command.

Processing Script

The Processing Script adds extra data to the results of command or message execution.

For detailed code documentation, see Visual Workflow Designer Scripts.

Special Variables by Task Class

REST/HTTP Task

In REST/HTTP tasks, you can access the response data using the `data` variable:

```
if data.get('status_code') == 200:
    settings['nms_downtime.status'] = 'false'
```

All Task Classes

All task classes can save specific configuration data for the target element. See PublicElementConfig for object reference documentation.

Example configuration and audit checks:

```
config.version = '1.7'
config.section = 'core'
config.required_approvement_percentage = 85

settings['audit.interfaces'] = config.check_key_occurrences("interface.*", ">=", 90,
required_for_audit=True)
settings['audit.valid_domain'] = config.compare_string_value('domain', 'startswith',
```

```
'zequenze', required_for_audit=True)
settings['audit.virtual_interface'] = config.complex_compare('interface.x', 'comienza con
Virtual', required_for_audit=True)
settings['audit.custom_hop'] = config.complex_compare('route.x', 'contiene un nexthop a la ip
172.16.254.1', required_for_audit=True)
settings['audit.route'] = config.check_key_matches('route.0.0.0.0/0', required_for_audit=True)
settings['audit.valid_hostname'] = config.compare_string_value('hostname', 'contains', '01',
required_for_audit=True)
settings['audit.has_logging_interface'] = config.check_key_matches("interface.*.logging",
required_for_audit=True)
settings['audit.has_vrf_interface'] = config.check_key_matches("interface.*.vrf",
required_for_audit=True)
settings['audit.has_gigabit_interface'] =
config.check_key_matches("interface.GigabitEthernet1.*", required_for_audit=True)
settings['audit.has_gigabit_two_interface'] =
config.check_key_matches("interface.GigabitEthernet2.*", required_for_audit=True)
settings['audit.result'] = config.audit_value
```

Revision #2
Created 2026-02-13 23:10:52 UTC by ipena@zequenze.com
Updated 2026-04-09 03:28:05 UTC by mauro@zequenze.com