

Models

Overview

Models in FLUX enable you to perform data transformations using different processing classes. Each model class requires specific formatting in the **Model data** field to define how data should be parsed and transformed.

You can validate your model's behavior by entering sample input in the **Test data** field and running a test to preview the output results.

Model Classes

CLI

Execute command-line instructions on a device.

Model data format: A command string that will be executed.

Structured Text

Process text containing variables in a structured format.

Model data format: Structured text with embedded variables.

Model data:

```
1 <group name='vlans*' method='table'>
2 # NAME MTU ARP VLAN-ID INTERFACE{{ ignore }}
3 {{ id }} R {{ name }} {{ MTU }} {{ ARP }} {{ VID }} {{ Interface }}
4 </group>
```

Test data:

```
1 lags: R - RUNNING
2 Columns: NAME, MTU, ARP, VLAN-ID, INTERFACE
3 # NAME MTU ARP VLAN-ID INTERFACE
4 0 R vlan10 1500 enabled 10 ether2
5 1 R vlan12 1500 enabled 11 ether2
6 2 R vlan13 1500 enabled 1 ether2
7 3 R vlan14 1500 enabled 14 ether2
```

Python

Execute Python code to transform data.

Model data format: Python code that uses a `data` variable, which will be populated with the content from the **Test data** input.

Model data:

```
1 lines = data.split("\n")
2
3 output = {}
4
5 for linea in lines:
6     partes = linea.split(": ")
7     nombre_variable = partes[0]
8     valor_variable = int(partes[1])
9     output[nombre_variable] = valor_variable
```

Test data:

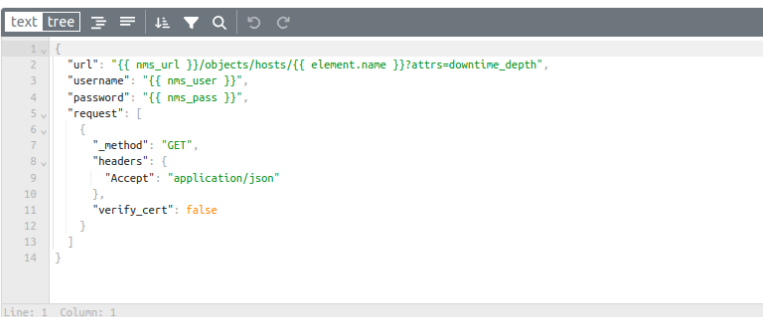
```
1 var 1: 2
2 var 2: 2
3 var 3: 3
```

JSON & XML

Parse and transform JSON or XML data structures.

Model data format: JSON structure with variables.

Model data:



```
1 {
2   "url": "{{ nns_url }}/objects/hosts/{{ element.name }}?attrs=downtime_depth",
3   "username": "{{ nns_user }}",
4   "password": "{{ nns_pass }}",
5   "request": [
6     {
7       "_method": "GET",
8       "headers": {
9         "Accept": "application/json"
10      },
11      "verify_cert": false
12    }
13  ]
14 }
```

The model data extracted from the specified file (when defined) will be showed here. If no model file is specified the model data in the corresponding format can be pasted here.

Test data:



```
1 {
2   "url": "jij/objects/hosts/ssss?attrs=downtime_depth",
3   "username": "cxz",
4   "password": "dsa",
5   "request": [
6     {
7       "_method": "GET",
8       "headers": {
9         "Accept": "application/json"
10      },
11      "verify_cert": false
12    }
13  ]
14 }
```

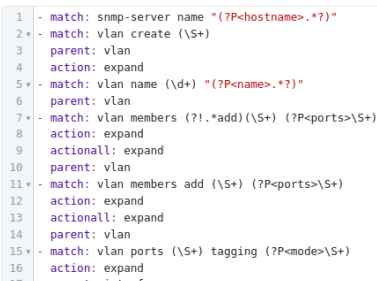
Set data that will be used to test this model.

Block

Parse block-formatted files using custom or predefined rules.

Model data format: YAML parser configuration with multiple parsing rules.

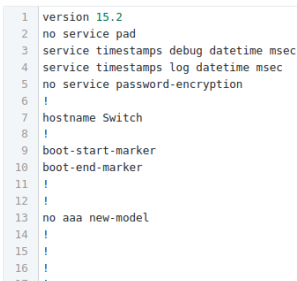
Model data:



```
1 - match: snmp-server name "(?P<hostname>.*?)"
2 - match: vlan create (\S+)
3   parent: vlan
4   action: expand
5 - match: vlan name (\d+) "(?P<name>.*?)"
6   parent: vlan
7 - match: vlan members (?!. *add) (\S+) (?P<ports>\S+)
8   action: expand
9   actionall: expand
10  parent: vlan
11 - match: vlan members add (\S+) (?P<ports>\S+)
12  action: expand
13  actionall: expand
14  parent: vlan
15 - match: vlan ports (\S+) tagging (?P<mode>\S+)
16  action: expand
```

The model data extracted from the specified file (when defined) will be showed here. If no model file is specified the model data in the corresponding format can be pasted here.

Test data:



```
1 version 15.2
2 no service pad
3 service timestamps debug datetime msec
4 service timestamps log datetime msec
5 no service password-encryption
6 !
7 hostname Switch
8 !
9 boot-start-marker
10 boot-end-marker
11 !
12 !
13 no aaa new-model
14 !
15 !
16 !
17 !
```

Set data that will be used to test this model.

Using predefined rules: Leave the **Model data** field empty to automatically detect the block type and parse it using FLUX's built-in rules.

Block Parser - Autodissector@1.0 Clase: [Block configuration](#) | ID: 527

Exportar Histórico Refrescar

Creado: 3 de Mayo de 2023 a las 18:50 | Last change: 3 de Mayo de 2023 a las 18:50

Settings Test model

Model data:

```
1 |
```

Test data:

```
1 version 15.2
2 no service pad
3 service timestamps debug datetime msec
4 service timestamps log datetime msec
5 no service password-encryption
6 !
7 hostname Switch
8 !
9 boot-start-marker
10 boot-end-marker
11 !
12 !
13 no aaa new-model
14 !
15 !
16 !
```

Testing Models

Basic Model Testing

1. Navigate to the **Test model** tab
2. Fill the **Model data** input with the parsing configuration
3. Fill the **Test data** input with sample data
4. Run `test` in the CLI to perform the transformation

Example:

test@1.0 Clase: [Python management](#) | ID: 519

Exportar Histórico Refrescar

Creado: 6 de Abril de 2023 a las 22:23 | Last change: 17 de Julio de 2023 a las 14:33

Settings Test model

Model data:

```
1 lines = data.split("\n")
2
3 output = {}
4
5 for linea in lines:
6     partes = linea.split(": ")
7     nombre_variable = partes[0]
8     valor_variable = int(partes[1])
9     output[nombre_variable] = valor_variable
```

Test data:

```
1 var 1: 2
2 var 2: 2
3 var 3: 3
```

The model data extracted from the specified file (when defined) will be showed here. If no model file is specified the model data in the corresponding format can be pasted here.

Set data that will be used to test this model.

```
test@1.0 > CLI terminal
Type help to list the available commands.
S test
```

Testing Against Element Context

You can test a model using the context of a specific element by entering the following in the **Test data** input:

```
id={x}
```

Replace `{x}` with the ID of the element whose context you want to use for testing.

Creating Model Templates with OpenAI

FLUX can generate model templates automatically using OpenAI based on your test data.

Prerequisites

1. Create an empty model with your desired class
2. Create an OpenAI helper element with the naming pattern: `openai-helper-{class}` (where `{class}` matches your model class)

Steps

1. Navigate to the **Test model** tab
2. Fill the **Test data** input with your sample data
3. Leave the **Model data** input empty
4. Run `template.create` in the CLI terminal

The system will call the `openai-helper-{class}` element, query the OpenAI API, print the generated template, and automatically evaluate it against your test data.

Dynamic Parameters

Models support dynamic variable substitution using element context data. Variables can be loaded in steps prior to model execution, allowing the model to replace them during transformation.

Example configuration:

```
Model data: 1 sudo cat {{env_path}}
```

Example usage:

Management script:

```
1 context['env_path'] = "/opt/zequenze/gate/etc/.env"
```

Please configure the script that will be executed after the connection with the managed element is established and before applying the data is transformed with the configured data models. This scripts can be used to perform specific data transformation and pre-processing actions, and to execute execute ad-hoc management commands in the managed element. Additional documentation of the processing scripts can be found on this [link](#)

Automation model:

Extract ENV file@1.0

x



Revision #2

Created 2026-02-13 23:07:12 UTC by ipena@zequenze.com

Updated 2026-03-05 22:30:35 UTC by ipena@zequenze.com