
TTR

Release 0.4.0

Denis Mulyalin

Jan 09, 2022

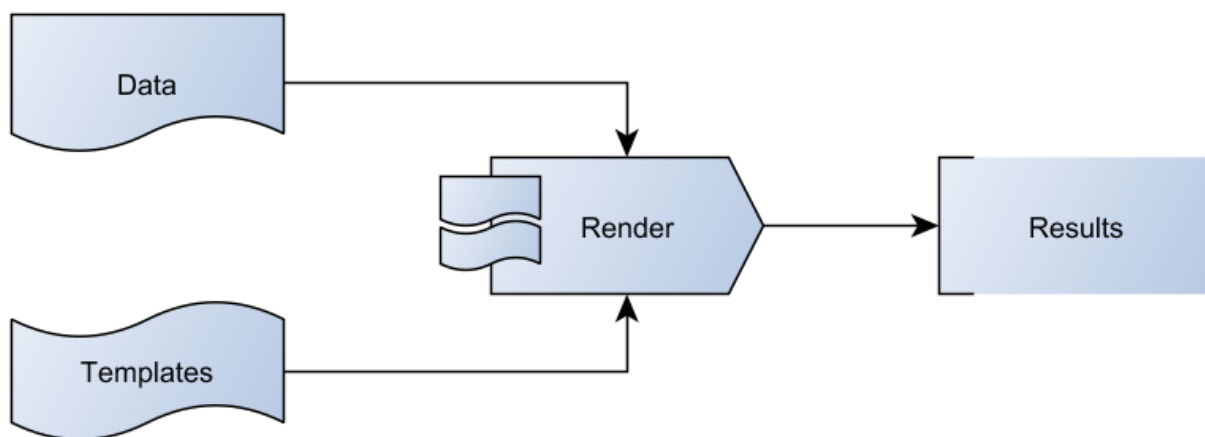
CONTENTS

1	Overview	1
2	Installation	3
3	Quick Start	5
4	Tutorials	11
4.1	Using TTR with Excel Tables	11
5	TTR CLI tool	19
6	Data Loader Plugins	21
6.1	XLSX Spreadsheets loader plugin	21
6.2	CSV Spreadsheets loader plugin	24
6.3	YAML loader	25
7	Processors Plugins	27
7.1	Multitemplate Processor	27
7.2	Templates Split Processor	28
7.3	Filtering Processor	29
8	Data Validation Plugins	31
8.1	Yangson Models Loader	31
8.2	Yangson Data Validation	33
9	Templates Loader Plugins	35
9.1	Base Template Loader	35
9.2	File Template Loader	36
9.3	Directory Template Loader	36
9.4	TTR Template Loader	36
9.5	XLSX Template Loader	37
10	Renderer Plugins	39
10.1	Jinja2 Renderer Plugin	39
11	Returner Plugins	41
11.1	Self Returner Plugin	41
11.2	Terminal Returner Plugin	41
11.3	File Returner Plugin	43
12	Jinja2 Templates Collection	45

13 API Reference	47
Python Module Index	51
Index	53

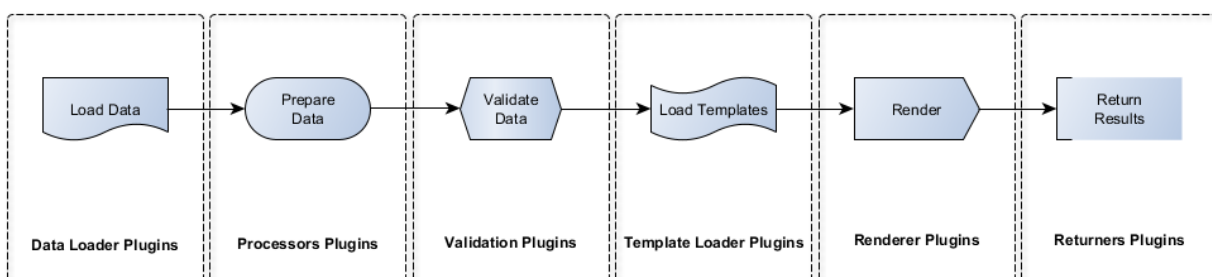
OVERVIEW

Module to produce text files using templates. TTR targets to implement common work flow:



Above approach is fairly simple but powerful enough to address various use cases where structured data need to be transformed in a textual form understandable by targeted system.

TTR uses plugins to load data and templates, render and return results.



Where:

- data plugins - load data from various format and transform it in a list of dictionaries
- processor plugins - optional step, but can be used to process data before rendering
- validation plugins - optional step, used to validate data before rendering it
- template loaders - retrieve template content from various sources (files, directories etc.)
- renderers - iterate over list of dictionaries data and render each item with template
- returners - return rendering results to various destinations, e.g. save to file system

In addition, TTR comes with a collection of Jinja2 templates to help with common use cases, such as generating configuration for network devices interfaces or BGP peers.

On the base level TTR takes list of dictionaries, renders each dictionary with template defined in `template_name_key` and saves rendered data in results dictionary keyed by `result_name_key`. Because of that each dictionary item must contain `template_name_key` and `result_name_key` keys.

Various plugins can be used to load data in a list of dictionaries with other plugins helping to process and validate it, render and save results.

INSTALLATION

From PyPi:

```
python3 -m pip install py-ttr
```

Make sure that TTR installed using Python version 3.6 or higher.

Or latest code from GitHub master branch:

```
python3 -m pip install git+https://github.com/dmulyalin/template-text-renderer
```

But for this to work need to have git installed on the system.

As part of installation TTR will automatically install these libraries:

```
PyYAML>=5.3.1  
openpyxl>=3.0.4  
Jinja2>=2.11.2
```

Alternatively, if you planning to use TTR as a CLI utility only, for Windows you can download `ttr.exe` file from [GitHub repository](#) *Executable* folder and use it as a CLI tool. **No Python required on the system in that case**, all dependencies packed within `ttr.exe` executable.

QUICK START

After installing TTR, simplest way to start using it is via CLI utility. For instance, this file with data at `/path/to/data.yaml`:

```
- interface: Gi1/1
  description: Customer A
  vid: 100
  ip: 10.0.0.1
  mask: 255.255.255.0
  vrf: cust_a
  template: interfaces.cisco_ios.txt
  device: rt-1
- interface: Gi1/2
  description: Customer C
  vid: 300
  ip: 10.0.3.1
  mask: 255.255.255.0
  vrf: cust_c
  template: interfaces.cisco_ios.txt
  device: rt-1
- interface: Gi1/2
  description: Customer B
  vid: 200
  ip: 10.0.2.1
  mask: 255.255.255.0
  vrf: cust_b
  template: interfaces.cisco_ios.txt
  device: rt-2
```

And this file with template at `/path/to/templates_folder/interfaces.cisco_ios.txt`:

```
interface {{ interface }}
{% if description is defined %}
  description {{ description }}
{% endif %}
{% if vid is defined %}
  encapsulation dot1q {{ vid }}
{% endif %}
{% if vrf is defined %}
  vrf forwarding {{ vrf }}
{% endif %}
{% if ip is defined and mask is defined %}
```

(continues on next page)

(continued from previous page)

```
ip address {{ ip }} {{ mask }}
{% endif %}
{% if ipv6 is defined and maskv6 is defined %}
  ipv6 address {{ ipv6 }}/{{ maskv6 }}
{% endif %}
exit
!
```

Could be combined using TTR by running this command:

```
ttr --data /path/to/data.yaml --template /path/to/templates_folder/ --print
```

Printing this output to terminal screen:

```
# -----
# rt-1 rendering results
# -----
interface Gi1/1
  description Customer A
  encapsulation dot1q 100
  vrf forwarding cust_a
  ip address 10.0.0.1 255.255.255.0
  exit
!
interface Gi1/2
  description Customer C
  encapsulation dot1q 300
  vrf forwarding cust_c
  ip address 10.0.3.1 255.255.255.0
  exit
!

# -----
# rt-2 rendering results
# -----
interface Gi1/2
  description Customer B
  encapsulation dot1q 200
  vrf forwarding cust_b
  ip address 10.0.2.1 255.255.255.0
  exit
!
```

Note: `--templates` argument should be a path to folder with templates files within that folder/subfolders or path to `.xlsx` spreadsheet file with templates or path to `.txt` file with single template content.

TTR can be used as a module instantiating TTR object and supplying it with required attributes:

```
import pprint
from ttr import ttr
```

(continues on next page)

(continued from previous page)

```

templates = {
"interfaces.cisco_ios.txt": """
interface {{ interface }}
{% if description is defined %}
  description {{ description }}
{% endif %}
{% if vid is defined %}
  encapsulation dot1q {{ vid }}
{% endif %}
{% if vrf is defined %}
  vrf forwarding {{ vrf }}
{% endif %}
{% if ip is defined and mask is defined %}
  ip address {{ ip }} {{ mask }}
{% endif %}
{% if ipv6 is defined and maskv6 is defined %}
  ipv6 address {{ ipv6 }}/{{ maskv6 }}
{% endif %}
exit
!
"""
}

data = """
- interface: Gi1/1
  description: Customer A
  vid: 100
  ip: 10.0.0.1
  mask: 255.255.255.0
  vrf: cust_a
  template: interfaces.cisco_ios.txt
  device: rt-1
- interface: Gi1/2
  description: Customer C
  vid: 300
  ip: 10.0.3.1
  mask: 255.255.255.0
  vrf: cust_c
  template: interfaces.cisco_ios.txt
  device: rt-1
- interface: Gi1/2
  description: Customer B
  vid: 200
  ip: 10.0.2.1
  mask: 255.255.255.0
  vrf: cust_b
  template: interfaces.cisco_ios.txt
  device: rt-2
"""

gen = ttr(data=data, data_plugin="yaml", templates_dict=templates)
results = gen.run()

```

(continues on next page)

(continued from previous page)

```
pprint.pprint(results)

# prints:
#
# {'rt-1': '\n'
#      'interface Gi1/1\n'
#      ' description Customer A\n'
#      ' encapsulation dot1q 100\n'
#      ' vrf forwarding cust_a\n'
#      ' ip address 10.0.0.1 255.255.255.0\n'
#      ' exit\n'
#      '!\n'
#      '\n'
#      'interface Gi1/2\n'
#      ' description Customer C\n'
#      ' encapsulation dot1q 300\n'
#      ' vrf forwarding cust_c\n'
#      ' ip address 10.0.3.1 255.255.255.0\n'
#      ' exit\n'
#      '!',
# 'rt-2': '\n'
#      'interface Gi1/2\n'
#      ' description Customer B\n'
#      ' encapsulation dot1q 200\n'
#      ' vrf forwarding cust_b\n'
#      ' ip address 10.0.2.1 255.255.255.0\n'
#      ' exit\n'
#      '!'}

```

It is also possible to source templates and data from text files:

```
import pprint
from ttr import ttr

gen = ttr(
    data="./data/data.yaml",
    templates="./Templates/"
)

gen.run()
pprint.pprint(gen.results)

# prints:
#
# {'rt-1': 'interface Gi1/1\n'
#      ' description Customer A\n'
#      ' encapsulation dot1q 100\n'
#      ' vrf forwarding cust_a\n'
#      ' ip address 10.0.0.1 255.255.255.0\n'
#      ' exit\n'
#      '!\n'

```

(continues on next page)

(continued from previous page)

```
#      'interface Gi1/2\n'
#      ' description Customer C\n'
#      ' encapsulation dot1q 300\n'
#      ' vrf forwarding cust_c\n'
#      ' ip address 10.0.3.1 255.255.255.0\n'
#      ' exit\n'
#      '!',
# 'rt-2': 'interface Gi1/2\n'
#      ' description Customer B\n'
#      ' encapsulation dot1q 200\n'
#      ' vrf forwarding cust_b\n'
#      ' ip address 10.0.2.1 255.255.255.0\n'
#      ' exit\n'
#      '!'}
```

Data is the same as in previous example but stored in `./data/data.yaml` file, TTR picked up YAML loader based on data file extension. Directory `./Templates/` contains `interfaces.cisco_ios.txt` template file.

Notice that rendering results also accessible using TTR object `results` property.

TTR also can be invoked using context manager:

```
import pprint
from ttr import ttr

with ttr("./data/data.yaml") as gen:
    results = gen.run()

pprint.pprint(gen.results)
```

Above example produces same results as before, `templates_dir` used with default value which is `./Templates/`.

TUTORIALS

4.1 Using TTR with Excel Tables

TTR can source data and templates from Excel spreadsheets, this tutorial is going to show how.

First, make sure to install TTR:

```
pip install py-ttr
```

That should also install required libraries such as Jinja2, Openpyxl, PyYAML and Yangson.

After installation, TTR CLI utility should become available for use on your OS, try opening terminal window and type `ttr -h`, if all good, should get similar to below output:

```
C:\>ttr -h
usage: ttr [-h]

TTR CLI, version 0.4.0

optional arguments:
-h, --help  show this help message and exit

Template Text Renderer CLI utility

-d, --data          OS path to folder with data files or to data file, default ./Data/
-t, --templates     OS path to folder, .txt or .xlsx file with template(s), default ./
↳ Templates/
-o, --output        Output folder location, default ./Output/<current time><data file_
↳ name>/
-p, --print         Print results to terminal instead of saving to folder
-l, --logging       Set logging level - "DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL";
↳ default ERROR
-f, --filters       Comma separated list of glob patterns to use for filtering data to_
↳ render
```

As you can see, TTR version we using for this tutorial is 0.4.0, but it should work same with newer versions.

4.1.1 Defining Data

Next, need to define data, as promised, going to use Excel spreadsheets for that. Create file named `table_ttr_with_excel_tutorial.xlsx`. Going to populate 4 tabs in it.

First tab called “devices” and used to fill in data for device’s base parameters such as loopback0 and management IP addresses, BGP ASN number, location, platform etc. Tab content is:

	A	B	C	D	E	F	G	H	I
1	device	Lo0	mgmt_ip	bgp_asn	platform	location	device_role	snmp_ro	template
2	R1	10.0.0.1	10.1.1.23	65101	ASR1006-X	12 Street, City-A	CORE	ReadONLY	cisco.ios.core_base
3	R2	10.0.0.2	10.1.2.201	65101	ASR1001-X	35 Road, City-B	CORE	ReadONLY	cisco.ios.core_base
4	R3	10.0.0.3	10.1.3.77	65101	ISR4492	7 Avenue, City-C	CORE	ReadONLY	cisco.ios.core_base
5	R4	10.0.0.4	10.1.4.81	65101	ASR1002-X	88 Boulevard, City-D	CORE	ReadONLY	cisco.ios.core_base
6	R5	10.0.0.5	10.1.5.98	65101	ASR9001	53 Route, City-E	CORE	ReadONLY	cisco.ios_xr.core_base
7									
8									

devices ip_links templates_ios templates_ios_xr

Important columns are - “device” and “template”, first used as a name to identify results and, for example, used by file returner to name text files when saving results to file system, second column refers to templates to render data with. Except for “template” column, all the columns form actual data used to render templates.

Second tab “ip_links” holds information for point to point links such as device names, interfaces, IP addresses, descriptions and MTU:

	A	B	C	D	E	F	G	H	I	J	K	L
1	device:a	interface:a	ip:a	description:a	device:b	interface:b	ip:b	description:b	mask	mtu	template:a	template:b
2	R1	Gi1/1	10.1.0.1	R2 - Eth1	R2	Eth1	10.1.0.2	R1 - Gi1/1	255.255.255.252	1500	cisco.ios.ip_intf	cisco.ios.ip_intf
3	R1	Gi2/3	10.1.0.5	R3 - Eth2	R3	Eth2	10.1.0.6	R1 - Gi2/3	255.255.255.252	2048	cisco.ios.ip_intf	cisco.ios.ip_intf
4	R3	Eth5	10.1.0.9	R4 - Gi1/0/1	R4	Gi1/0/1	10.1.0.10	R3 - Eth5	255.255.255.252	1500	cisco.ios.ip_intf	cisco.ios.ip_intf
5	R3	Eth6	10.1.0.13	R5 - Te0/0/0/0.147	R5	Te0/0/0/0.147	10.1.0.14	R3 - Eth6	255.255.255.252	9600	cisco.ios.ip_intf	cisco.ios_xr.ip_intf
6	R4	Gi1/0/14	10.1.0.17	R5 - Te0/0/0/0.631	R5	Te0/0/0/0.631	10.1.0.18	R4 - Gi1/0/14	255.255.255.252	9600	cisco.ios.ip_intf	cisco.ios_xr.ip_intf
7												
8												
9												

devices ip_links templates_ios templates_ios_xr

This tab makes use of [multiple templates suffix separation](#) to simplify links definition in a way that single link requires only one table row to define it.

Above two tabs form data component, additional rows can be added as required, more tabs can be included as well to describe network functionality.

4.1.2 Defining Templates

Now is the turn to define templates to render above data. TTR supports sourcing templates content from excel spreadsheets - any tab with name that starts with “templates” considered to contain templates and used by TTR to load templates content.

Templates in this example separated across two tabs for demonstration purposes. However, it is an excel spreadsheet and nothing stopping us from putting all templates on one tab or adding more tabs if we think it better suite our design goals.

Here is the content of “templates_ios” tab:

	A
1	template:cisco.ios.core base
2	hostname {{ device }}
3	!
4	snmp-server community {{ snmp_ro }} RO
5	snmp-server location {{ location }} [Device {{ device }}, {{ platform }}, {{ device_role }}]
6	!
7	interface loopback0
8	ip address {{ Lo0 }} 255.255.255.255
9	description RID Loopback
10	!
11	interface Management1
12	ip address {{ mgmt_ip }} 255.255.255.128
13	no shutdown
14	!
15	router bgp {{ bgp_asn }}
16	bgp router-id {{ Lo0 }}
17	address-family ipv4 unicast
18	address-family ipv6 unicast
19	!
20	template:cisco.ios.ip intf
21	interface {{ interface }}
22	ip address {{ ip }} {{ mask }}
23	description {{ description }}
24	mtu {{ mtu }}
25	no shutdown
26	!
27	
28	
	devices ip_links templates_ios templates_ios_xr +

And this is the content of “templates_ios_xr” tab:

	A
1	template:cisco.ios_xr.core_base
2	hostname {{ device }}
3	!
4	snmp-server community {{ snmp_ro }} RO
5	snmp-server location {{ location }} [Device {{ device }}, {{ platform }}, {{ device_role }}]
6	!
7	interface loopback0
8	ipv4 address {{ Lo0 }} 255.255.255.255
9	description RID Loopback
10	!
11	interface mgmt0
12	ipv4 address {{ mgmt_ip }} 255.255.255.128
13	no shutdown
14	!
15	router bgp {{ bgp_asn }}
16	bgp router-id {{ Lo0 }}
17	address-family ipv4 unicast
18	address-family ipv6 unicast
19	!
20	template:cisco.ios_xr.ip_intf
21	interface {{ interface }}
22	ipv4 address {{ ip }} {{ mask }}
23	description {{ description }}
24	mtu {{ mtu }}
25	no shutdown
26	!
27	
28	

As demonstrated above, templates are in fact Jinja2 templates. Each template definition starts with a row `template:<template name>` where `template name` referred by data tabs' `template` columns.

Each `template name` must be unique across all templates, naming convention for templates' names is arbitrary from TTR perspectives and can be defined based on user preference.

4.1.3 Generating Text Results

The simplest way to produce results is to use TTR CLI utility, execute below command from folder with `table_ttr_with_excel_tutorial.xlsx` file:

```
ttr -d table_ttr_with_excel_tutorial.xlsx -p
```

and TTR should print rendering results to terminal screen:

```

C:\TTR_test>ttr -d table_ttr_with_excel_tutorial.xlsx -p

# -----
# R1 rendering results
# -----
hostname R1
!
snmp-server community ReadONLY RO
snmp-server location 12 Street, City-A [Device R1, ASR1006-X, CORE]
!
interface loopback0
 ip address 10.0.0.1 255.255.255.255
 description RID Loopback
!
interface Management1
 ip address 10.1.1.23 255.255.255.128
 no shutdown
!
router bgp 65101
 bgp router-id 10.0.0.1
 address-family ipv4 unicast
 address-family ipv6 unicast
!
interface Gi1/1
 ip address 10.1.0.1 255.255.255.252
 description R2 - Eth1
 mtu 1500
 no shutdown
!
interface Gi2/3
 ip address 10.1.0.5 255.255.255.252
 description R3 - Eth2
 mtu 2048
 no shutdown
!

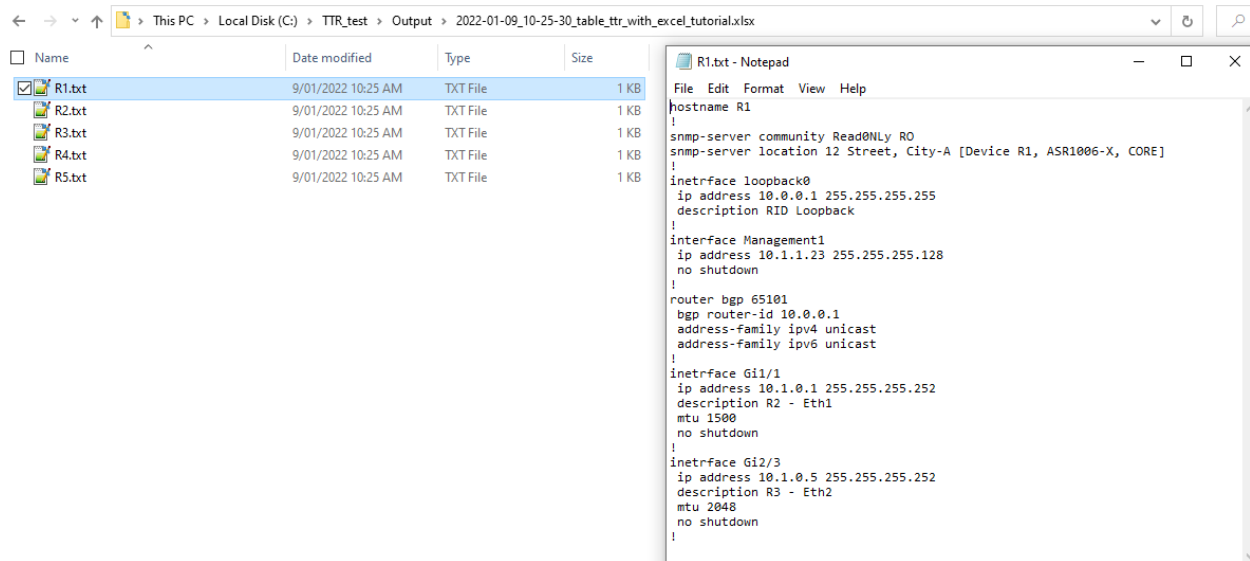
# -----
# R2 rendering results
# -----
hostname R2
!
snmp-server community ReadONLY RO
snmp-server location 35 Road, City-B [Device R2, ASR1001-X, CORE]

```

Outputting result to terminal is good if amount of output is fairly small or for debugging purposes, better way is to let TTR output result in a folder on the local file system, for that need to run same command as above but without `-p` argument:

```
ttr -d table_ttr_with_excel_tutorial.xlsx
```

If all is well, TTR should create Output folder within current directory with subdirectory containing per-device rendering results:



4.1.4 Next Steps

Ideas for improvement.

Keep Data and Templates separate

Data and templates does not have to be defined inside the same Excel document, in fact it is better to keep them separate. For example, if we move templates to `templates.xlsx` spreadsheet we can use this command to generate results:

```
ttr -d table_ttr_with_excel_tutorial.xlsx -t templates.xlsx
```

Keeping templates and actual data in separate files will allow to add more data spreadsheets where all of them can reference templates from single source.

Moreover, templates can be defined within, for instance, `Templates` folder using text files, where each file name must be same as the value used in data tables' `template` columns. The benefit of using text files is capability to simplify version control using for example GIT.

Validate Data

Data can contain errors, excel features can help to spot and fix some of them, but additional layer of protection can be provided by YANG models. YANG models can define what is mandatory or optional, can check values types - is it an integer or a string, is it matching regular expression or integer value falls within certain range etc. That will help to make sure that templates supplied with verified data, lowering the probability of having errors in your devices' configurations.

4.1.5 Conclusion

In a few seconds with a single command TTR can produce network devices configurations out of excel table, that can be very powerful tool to streamline the process of building consistent configuration using a collection of tested and verified templates. Introducing fixes becomes easier, adding new devices becomes a matter of populating excel spreadsheet. Excel features can help to simplify editing, deleting, sorting and searching through your data.

Hope this tutorial was useful, thank you for reading.

`table_ttr_with_excel_tutorial.xlsx` can be downloaded from [TTR GitHub repository](#)

TTR CLI TOOL

TTR CLI is a tool built using TTR library.

This tool can be used to render data from various sources and either print results to screen or save them in output folder.

Supported arguments:

-d, --data	OS path to folder with data files or to data file, default ./Data/
-t, --templates	OS path to folder, .txt or .xlsx file with template(s), default ./
↪ Templates/	
-o, --output	Output folder location, default ./Output/<current time><data file_
↪ name>/	
-p, --print	Print results to terminal instead of saving to folder
-l, --logging	Set logging level - "DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL";
↪ default ERROR	
-f, --filters	Comma separated list of glob patterns to use for filtering data to_
↪ render	

Note: --templates argument should be a path to folder with templates files within that folder/subfolders or path to .xlsx spreadsheet file with templates or path to .txt file with single template content.

In general case TTR CLI utility takes data file and templates location references and saves produced results in a subfolder within ./Output/ directory, where subfolder name has this format ./Output/<current time><data file name>/.

Sample invocation:

```
ttr -d ./data/data.yaml
ttr -d ./data/data.yaml -t ./templates_folder/
ttr -d ./data/data.yaml -t ./templates/templates_spreadsheet_file.xlsx
```

Alternatively a path to directory can be provided instead of data file, in that case TTR will scan that path and prompt user to select file to work with:

```
ttr -d ./data/
=====
Files found in './data/' directory
0: csv_data_1.csv
1: data.yaml
2: table_data_1.xlsx
Choose data file to work with (number): 1
```


DATA LOADER PLUGINS

Data plugins responsible for loading data to render from various formats such as `YAML` structured text or `xlsx` spreadsheets.

By default TTR uses file extension to choose plugin for loading data, alternatively TTR object `data_plugin` attribute can be used to specify default plugin to use.

For instance, to load `data.csv` file TTR will use `csv` plugin, to load `data.yaml` file, `yaml` plugin will be used, etc.

Data plugins load data in a list of dictionaries, where each item rendered using template as per `template_name_key` attribute.

6.1 XLSX Spreadsheets loader plugin

Plugin reference name: `xlsx`

This plugin supports loading data from multiple sheets, combining them for rendering.

Prerequisites:

- Requires `openpyxl` $\geq 3.0.0$ library

Restrictions and guidelines

Spreadsheets must contain a column or multiple columns with headers starting with `template_name_key` argument string, default is `template`. Values of `template(s)` columns either names of the template to use for rendering or OS path string to template file relative to `template_dir` argument supplied to TTR object on instantiation.

In addition, table must contain column with `result_name_key` values, default is `device`, they used to combine results, i.e. rendering results for identical `result_name_key` combined in a single string. `result_name_key` used further by returners to return results.

Spreadsheet tabs with names starting with `#` are skipped, useful to comment out tabs that no need to render.

First row in the spreadsheet must contain headers, otherwise spreadsheet not loaded.

Note: empty cells loaded with value of `None`

Sample spreadsheet table that contains details for interfaces configuration:

device	interface	vid	vrf	ip	mask	template
rt1	Gi1/1.100	100	MGMT	10.0.0.1	24	ttr://simple/interface.cisco_ios.txt
rt1	Gi2/3		CUST	10.3.0.1	30	ttr://simple/interface.cisco_ios.txt
sw23	Vlan21		MGMT	10.0.0.2	24	ttr://simple/interface.cisco_ios.txt

where:

- device column contains result_name_key values
- template column contains template_name_key values
- ttr://simple/interface.cisco_ios.txt - is a template included in TTR templates collection

Above table loaded into this list of dictionaries:

```
[{'device': 'rt1',  
  'interface': 'Gi1/1.100',  
  'ip': '10.0.0.1',  
  'mask': 24,  
  'template': 'ttr://simple/interface.cisco_ios.txt',  
  'vid': 100,  
  'vrf': 'MGMT'},  
{'device': 'rt1',  
  'interface': 'Gi2/3',  
  'ip': '10.3.0.1',  
  'mask': 30,  
  'template': 'ttr://simple/interface.cisco_ios.txt',  
  'vid': None,  
  'vrf': 'CUST'},  
{'device': 'sw23',  
  'interface': 'Vlan21',  
  'ip': '10.0.0.2',  
  'mask': 24,  
  'template': 'ttr://simple/interface.cisco_ios.txt',  
  'vid': None,  
  'vrf': 'MGMT'}]
```

Combined with ttr://simple/interface.cisco_ios.txt it will produce these results:

```
ttr -d /path_to_table.xlsx/ -p  
  
# -----  
# rt1 rendering results  
# -----  
interface Gi1/1.100  
  encapsulation dot1q 100  
  vrf forwarding MGMT  
  ip address 10.0.0.1 24  
  exit  
!  
interface Gi2/3  
  encapsulation dot1q None  
  vrf forwarding CUST  
  ip address 10.3.0.1 30  
  exit  
!  
  
# -----  
# sw23 rendering results  
# -----  
interface Vlan21
```

(continues on next page)

(continued from previous page)

```
encapsulation dot1q None
vrf forwarding MGMT
ip address 10.0.0.2 24
exit
!
```

6.1.1 Multiple Templates suffix separation

Using multitemplate processor it is possible to define multiple template columns within same spreadsheet tab using suffixes. Columns with headers with same suffixes considered part of same datum and combined together. Headers without suffixes shared across all datums.

For example, this table uses :a and :b suffixes to denote relationship with certain templates:

de- vice:a	inter- face:a	ip:a	mask	de- vice:b	inter- face:b	ip:b	template:a	template:b
rt1	Gi1/1	10.0.0.1	30	rt2	Gi1	10.0.0.2	ttr://simple/interface.cisco_	ttr://simple/interface.cisco_
rt3	Gi2/3	10.3.0.1	30	rt4	Gi3	10.3.0.2	ttr://simple/interface.cisco_	ttr://simple/interface.cisco_

where:

- device columns contains result_name_key values
- template columns contains template_name_key values
- ttr://simple/interface.cisco_ios.txt - is a template included in TTR templates collection
- ttr://simple/interface.cisco_nxos.txt - is a template included in TTR templates collection

Above table data, after passing through multitemplate processor loaded into this list of dictionaries:

```
import pprint
from ttr import ttr

gen = ttr("./path/to/table.xlsx", processors=["multitemplate"])

pprint.pprint(gen.data_loaded)

# prints:
# [{ 'device': 'rt1',
#   'interface': 'Gi1/1',
#   'ip': '10.0.0.1',
#   'mask': 30,
#   'template': 'ttr://simple/interface.cisco_ios.txt'},
#  { 'device': 'rt2',
#   'interface': 'Gi1',
#   'ip': '10.0.0.2',
#   'mask': 30,
#   'template': 'ttr://simple/interface.cisco_nxos.txt'},
#  { 'device': 'rt3',
#   'interface': 'Gi2/3',
#   'ip': '10.3.0.1',
#   'mask': 30,
```

(continues on next page)

(continued from previous page)

```
# 'template': 'ttr://simple/interface.cisco_ios.txt'},  
# {'device': 'rt4',  
# 'interface': 'Gi3',  
# 'ip': '10.3.0.2',  
# 'mask': 30,  
# 'template': 'ttr://simple/interface.cisco_nxos.txt'}]
```

That technique allows to simplify definition of “paired” configurations, e.g. device A and device B configurations or forward and rollback configurations etc.

`ttr.plugins.data.xlsx_loader.load(data, templates_dict, template_name_key, **kwargs)`

Function to load XLSX spreadsheet. Takes OS path to `.xlsx` file and returns list of dictionaries, where keys equal to headers of spreadsheets' tabs.

Parameters

- **data** – string, OS path to `.xlsx` file
- **templates_dict** – dictionary to load templates from spreadsheet
- **template_name_key** – string, templates column header prefix

6.2 CSV Spreadsheets loader plugin

Plugin Name: `csv`

Support loading data from CSV text file.

Spreadsheet must contain a column or multiple columns with headers starting with `template_name_key` argument string. Values of template(s) columns either names of the template to use for rendering or OS path string to template file relative to `template_dir` argument supplied to TTR object on instantiation.

In addition, table must contain column with `result_name_key` values, they used to combine results, i.e. rendering results for identical `result_name_key` combined in a single string. `result_name_key` used further by returners to return results.

`ttr.plugins.data.csv_loader.load(data, templates_dict=None, template_name_key=None, **kwargs)`

Function to load CSV spreadsheet.

Parameters

- **data** – OS path to CSV text file
- **templates_dict** – (dict) dictionary to load templates from spreadsheet, not supported by csv loader
- **template_name_key** – (str) templates column header prefix, not supported by csv loader
- **kwargs** – (dict) any additional arguments to pass on to `csv.DictReader` object instantiation

6.3 YAML loader

Plugin Name: yaml

Prerequisites:

- Requires PyYAML library

Plugin to load data to render from YAML structured text.

```
ttr.plugins.data.yaml_loader.load(data, templates_dict=None, template_name_key=None, **kwargs)
```

Function to load YAML data from text file or from string. Text file should have `.yaml` or `.yml` extension to properly detect loader.

Parameters

- **data** – string, OS path to text file or YAML structured text
- **templates_dict** – (dict) dictionary to load templates from spreadsheet, not supported by yaml loader
- **template_name_key** – (str) templates column header prefix, not supported by yaml loader
- **kwargs** – (dict) any additional arguments are ignored

PROCESSORS PLUGINS

Processors used to process loaded data before its rendered.

7.1 Multitemplate Processor

Plugin reference name: multitemplate

Processor to extract multiple template dictionaries from each data item based on suffix values.

Takes a list of dictionaries:

```
[{'device': 'r1',
  'hostname': 'r1',
  'lo0_ip': '1.1.1.1',
  'lo0_ip_rollback': '1.1.1.11',
  'template': 'test_path/device_base',
  'template_rollback': 'test_path/device_base_rollback'},
 {'device:a': 'r1',
  'device:b': 'r2',
  'interface:a': 'Eth1',
  'interface:b': 'Eth1',
  'ip:a': '10.0.0.1',
  'ip:b': '10.0.0.2',
  'mask': 24,
  'template:a': 'test_path/interf_cfg',
  'template:b': 'test_path/interf_cfg_b'}]
```

Returns:

```
[{'device': 'r1',
  'hostname': 'r1',
  'lo0_ip': '1.1.1.11',
  'template': 'test_path/device_base_rollback'},
 {'device': 'r1',
  'hostname': 'r1',
  'lo0_ip': '1.1.1.1',
  'template': 'test_path/device_base'},
 {'device': 'r1',
  'interface': 'Eth1',
  'ip': '10.0.0.1',
  'mask': 24,
```

(continues on next page)

(continued from previous page)

```
'template': 'test_path/interf_cfg'},
{'device': 'r2',
'interface': 'Eth1',
'ip': '10.0.0.2',
'mask': 24,
'template': 'test_path/interf_cfg_b'}}
```

Where `template_name_key` is `template`.

Multitemplate processor detects suffixes/endings, `:a` and `:b` in this case, and uses them to split dictionaries apart, populating them with values corresponding to certain suffixes.

Key names without suffixes considered as common values and shared across all dictionaries.

`ttr.plugins.processors.multitemplate_processor.process(data, template_name_key, **kwargs)`

Function to process multitemplate data items.

Parameters

- **data** – (list), data to process - list of dictionaries
- **template_name_key** – string, name of the template key
- **kwargs** – (dict) any additional arguments ignored

7.2 Templates Split Processor

Plugin reference name: `templates_split`

Processor to support definition of several templates separated by delimiter (default - ;) consequentially splitting data into several items with dedicated template.

Takes a list of dictionaries, for example:

```
[{'device': 'r1',
'hostname': 'r1',
'lo0_ip': '1.1.1.1',
'lo0_ip_rollback': '1.1.1.11',
'template': 'device_base; isis_base; bgp_base'},
{'device': 'r2',
'hostname': 'r2',
'lo0_ip': '1.1.1.2',
'lo0_ip_rollback': '1.1.1.22',
'template': 'device_base; bgp_base'}]
```

After splitting templates `templates_split` processor returns:

```
[{'device': 'r1',
'hostname': 'r1',
'lo0_ip': '1.1.1.1',
'lo0_ip_rollback': '1.1.1.11',
'template': 'device_base'},
{'device': 'r1',
'hostname': 'r1',
'lo0_ip': '1.1.1.1',
```

(continues on next page)

(continued from previous page)

```

    'lo0_ip_rollback': '1.1.1.11',
    'template': 'isis_base'},
{'device': 'r1',
 'hostname': 'r1',
 'lo0_ip': '1.1.1.1',
 'lo0_ip_rollback': '1.1.1.11',
 'template': 'bgp_base'},
{'device': 'r2',
 'hostname': 'r2',
 'lo0_ip': '1.1.1.2',
 'lo0_ip_rollback': '1.1.1.22',
 'template': 'device_base'},
{'device': 'r2',
 'hostname': 'r2',
 'lo0_ip': '1.1.1.2',
 'lo0_ip_rollback': '1.1.1.22',
 'template': 'bgp_base'}}]

```

ttr.plugins.processors.templates_split.process(*data*, *template_name_key*, *split_char*=';', ***kwargs*)

Function to split templates. e.g. if *template_name_key* value contains several templates, this processor will split them using *split_char* and produce data item for each template coping data accordingly.

Parameters

- **data** – list of dictionaries to process
- **template_name_key** – string, name of the template key
- **split_char** – str, character to use to split template names
- **kwargs** – (dict) any additional arguments ignored

7.3 Filtering Processor

Plugin reference name: filtering

Processor to filter data using glob patterns. Filtering done against *result_name_key* values.

Takes a list of dictionaries, for example:

```

[{'device': 'rr21',
 'lo0_ip': '1.1.1.1',
 'lo0_ip_rollback': '1.1.1.11',
 'template': 'device_base'},
{'device': 'core-1',
 'lo0_ip': '1.1.1.2',
 'lo0_ip_rollback': '1.1.1.22',
 'template': 'device_base'}]

```

If filter pattern is *core-** and *result_name_key* is *device*, filtering processor will return:

```

[{'device': 'core-1',
 'lo0_ip': '1.1.1.2',

```

(continues on next page)

(continued from previous page)

```
'lo@ip_rollback': '1.1.1.22',  
'template': 'device_base}]
```

`ttr.plugins.processors.filtering.process(data, result_name_key, filters=None, **kwargs)`

Function to filter data using glob patterns.

Parameters

- **data** – list of dictionaries to process
- **filters** – list, list of glob patterns to use for filtering. Filtering successful if at list one pattern matches
- **result_name_key** – (str) name of the key in data items value of which should be used as a key in results dictionary, default device. Filtering done against values defined under result_name_key
- **kwargs** – (dict) any additional arguments ignored

DATA VALIDATION PLUGINS

Data validation plugins responsible for validating loaded data to make sure it adheres to model or schema requirements.

By default TTR uses `yangson` plugin for data validation, alternatively TTR object `validator` attribute can be used to specify plugin to use.

Validation step is optional, if no models provided, data not validated. However, if required data models can be used to make sure that correct data provided prior to performing rendering step.

There are two types of validation plugins:

1. Plugins to load models
2. Plugins to validate the actual data

8.1 Yangson Models Loader

Reference name `yangson`

This plugin loads YANG models into `yangson DataModel` objects.

YANG models must sit within their own directories, each such a directory used to create JSON library for Yangson to load models from.

Directory name, main YANG model file name and module name must be the same, directory name used as a reference name for the model.

For example, this is directory tree with YANG models inside:

```
|-- Models
  |-- interface
    |-- ietf-inet-types@2013-07-15.yang
    |-- interface.yang
  |-- vrf
    |-- vrf.yang
```

Above directory structure translated to two models named `interface` and `vrf`, these names can be used to reference models in data for validation, e.g.:

```
- interface: Gi1/1
  description: Customer A
  vid: 100
  ip: 10.0.0.1
  mask: 255.255.255.0
```

(continues on next page)

(continued from previous page)

```
vrf: cust_a
device: R1
template: interfaces.cisco_ios
model: interface # YANG model name to validate this data item
```

For reference, YANG model Models/interface/interface.yang file content is:

```
module interface {
  yang-version "1.1";

  namespace "http://ttr/test-1";

  import ietf-inet-types {
    prefix inet;
  }

  typedef ipmask {
    type string {
      pattern '([0-9]{1,3}.){3}[0-9]{1,3}';
    }
    description
      "Pattern to match strings like 255.255.255.0 or 255.0.0.0";
  }

  prefix "ttr";

  leaf interface {
    mandatory true;
    type string;
  }
  leaf template {
    mandatory true;
    type string;
  }
  leaf device {
    mandatory true;
    type string;
  }
  leaf description{
    type string;
  }
  leaf vid {
    type int32;
  }
  leaf ip {
    type inet:ipv4-address;
  }
  leaf mask {
    type ipmask;
  }
  leaf vrf {
    type string;
  }
}
```

(continues on next page)

(continued from previous page)

```
}
}
```

`ttr.plugins.models.yangson_model_loader.load(models_dict, models_dir, **kwargs)`

Creates JSON-encoded YANG library data [RFC7895] and instantiates data model object out of it.

Parameters

- **models_dir** – (str) OS path to directory with YANG models modules subdirectories, each subdirectory models loaded to form single DataModel and added to models_dict under directory name key.
- **models_dict** – (dict) dictionary to store loaded model object at
- **kwargs** – (dict) any additional arguments ignored
- **return** – None

8.2 Yangson Data Validation

Reference name yangson

This plugin relies on Yangson library for data instance validation using YANG models.

`ttr.plugins.validate.validate_yangson.validate(data, model_content, model_name, validation_scope='all', content_type='all', on_fail='raise')`

Validate data for compliance with YANG modules.

Parameters

- **data** – (dict) dictionary data to validate
- **model_content** – (obj) Fully instantiated Yangson DataModel object
- **model_name** – (str) name of the model
- **content_type** – (str) optional, content type as per <https://yangson.labs.nic.cz/enumerations.html> supported - all, config, nonconfig
- **validation_scope** – (str) optional, validation scope as per <https://yangson.labs.nic.cz/enumerations.html> supported - all, semantics, syntax
- **on_fail** – (str) action to do if validation fails - raise (default) or log

Returns:

- True if validation succeeded
- False if validation failed and on_fail is “log”
- Raises `RuntimeError` exception if validation failed and on_fail is “raise”

TEMPLATES LOADER PLUGINS

Template loaders responsible for loading template text from various sources.

9.1 Base Template Loader

Reference name base

Base loader loads templates content in `templates_dict` dictionary using other loader plugins following this order:

0. Check if template with given name already exists in `templates_dict`, use it if so
1. Check if template name starts with `ttr://`, load it using `ttr_template_loader`
2. If template name references file, load it using `file_template_loader`
3. If `templates` is a directory load template content using `dir_template_loader`
4. If `templates` referring to `.xlsx` file load all templates using `xlsx_template_loader`

On failure to load template file, base loader will log an error message and TTR will continue processing other data items.

`ttr.plugins.templates.base_template_loader.load(template_name, templates_dict, templates, **kwargs)`
Function to locate template file and return it's content

Attributes

Parameters

- **template_name** – (str) - name of template to load
- **templates** – (str) - location of templates
- **templates_dict** – (dict) - dictionary of to store template content in
- **kwargs** – (dict) any additional arguments ignored

Returns True on success and False on failure to load template

On success loads template content in `templates_dict` and returns True, on failure returns False.

9.2 File Template Loader

Reference name file

Loads template for rendering from file.

```
ttr.plugins.templates.file_template_loader.load(template_name, templates_dict, filepath=None,
                                                **kwargs)
```

Function to load template content from file path.

Parameters

- **template_name** – (str) name of template to load, should point to file if no **filepath** argument provided
- **templates_dict** – (str) dictionary to store template content in
- **filepath** – (str) optional, path to file to open
- **kwargs** – (dict) any additional arguments ignored

Returns True on success and False on failure to load template

9.3 Directory Template Loader

Reference name dir

Loads template for rendering from file in directory

```
ttr.plugins.templates.dir_template_loader.load(template_name, templates_dict, templates, **kwargs)
```

Function to load template content from file in directory.

Parameters

- **template_name** – (str) name of template to load, should point to file
- **templates_dict** – (str) dictionary to store template content in
- **templates** – (str) OS path to directory with template file
- **kwargs** – (dict) any additional arguments ignored

Returns True on success and False on failure to load template

9.4 TTR Template Loader

Reference name ttr

Loads templates for rendering from TTR package.

```
ttr.plugins.templates.ttr_template_loader.load(template_name, templates_dict, **kwargs)
```

Function to load template content from `ttr://...` path.

Parameters

- **template_name** – (str) name of template to load
- **templates_dict** – (str) dictionary to store template content in
- **kwargs** – (dict) any additional arguments ignored

Returns True on success and False on failure to load template

9.5 XLSX Template Loader

Plugin reference name: `xlsx`

Spreadsheet might contain multiple tabs with names starting with `template`, these tabs can contain templates to use for rendering data from other tabs. All the templates loaded line by line, `template:{{ template_name }}` lines used to identify end of previous and start of next template, where `template_name` used for referencing template.

Sample table that contains rendering templates, no headers required:

template:interface
interface {{ interface }}
description {{ description }}
template:logging
logging host {{ log_server }}

Above templates loaded in a dictionary:

```
{
  "interface": 'interface {{ interface }}\n'
               '  description {{ description }}',
  "logging": 'logging host {{ log_server }}'
}
```

In this case templates referenced in data using `interface` and `logging` template names

`ttr.plugins.templates.xlsx_template_loader.load(templates_dict, templates=None, sheet=None, **kwargs)`

Function to load **all** templates from `xlsx` spreadsheets.

Parameters

- **templates** – OS path to `.xlsx` file
- **sheet** – openpyxl sheet object
- **templates_dict** – (dict) dictionary of {template name: template content} to load templates in
- **kwargs** – (dict) any additional arguments ignored

Returns True on success and False on failure to load template

RENDERER PLUGINS

Renderers responsible for combining data with templates and producing text output.

10.1 Jinja2 Renderer Plugin

Prerequisites:

- Requires `jinja2` library

Warning: data keys or table headers must be valid Python variable names for Jinja2 engine to work correctly, e.g. a variable name must start with a letter or the underscore character.

This renderer uses Jinja2 templates to render data and produce text results.

For example, if this is a data to render expressed in YAML format:

```
- interface: Gi1/1
  description: Customer A
  dot1q: 100
  ip: 10.0.0.1
  mask: 255.255.255.0
  vrf: cust_a
  template: ttr://interfaces.cisco_ios
  device: rt-1
- interface: Gi1/2
  description: Customer B
  dot1q: 200
  ip: 10.0.2.1
  mask: 255.255.255.0
  vrf: cust_b
  template: ttr://interfaces.cisco_ios
  device: rt-2
```

`template_name_key` corresponds to `template` key in above data, `result_name_key` corresponds to `device` key in above data.

And this is the content of `ttr://interfaces.cisco_ios` template:

```
interface {{ interface }}
description {{ description }}
```

(continues on next page)

(continued from previous page)

```
encapsulation dot1q {{ vid }}
vrf forwarding  {{ vrf }}
ip address {{ ip }} {{ mask }}
ipv6 address {{ ipv6 }}/{{ maskv6 }}
!
```

This renderer will combine each item in above data with `ttr://interfaces.cisco_ios` template and return results for further processing.

`ttr.plugins.renderers.jinja2_renderer.render`(*data*, *template_name_key*, *templates*, *templates_dict*, *result_name_key*, ***renderer_kwargs*)

Render function takes data, templates and produces text output.

Parameters

- **data** – (list), list of dictionaries render
- **templates_dict** – (dict), dictionary keyed by template name with template content as a value
- **template_name_key** – (str), name of template key to use for data rendering, default - `template`
- **result_name_key** – (str), name of result key to use to combine rendering results, default - `device`
- **renderer_kwargs** – (dict), kwargs to pass on to `jinja2.Template(..., **kwargs)` object instantiation

By default `renderer_kwargs` will include:

```
{"trim_blocks": True, "lstrip_blocks": True}
```

RETURNER PLUGINS

Returners responsible for returning produced results to various destinations.

11.1 Self Returner Plugin

Plugin Name: self

This plugin does nothing with results, implementing behavior where results stored in TTR object for further programmatic consumption.

`ttr.plugins.returners.self_returner.dump(data_dict, **kwargs)`

This function applying no actions to results, implemented to keep plugins API consistent.

Parameters

- **data_dict** – (dict) dictionary keyed by `result_name_key` where values are rendered results string
- **kwargs** – (dict) any additional arguments ignored

11.2 Terminal Returner Plugin

Plugin Name: terminal

This plugin prints rendered result to terminal screen applying minimal formatting to improve readability.

For instance if these are rendering results:

```
{'rt-1': 'interface Gi1/1\n'
      ' description Customer A\n'
      ' encapsulation dot1q 100\n'
      ' vrf forwarding cust_a\n'
      ' ip address 10.0.0.1 255.255.255.0\n'
      ' exit\n'
      '!\n'
      'interface Gi1/2\n'
      ' description Customer C\n'
      ' encapsulation dot1q 300\n'
      ' vrf forwarding cust_c\n'
      ' ip address 10.0.3.1 255.255.255.0\n'
      ' exit\n'
      '!',
```

(continues on next page)

(continued from previous page)

```
'rt-2': 'interface Gi1/2\n'
      ' description Customer B\n'
      ' encapsulation dot1q 200\n'
      ' vrf forwarding cust_b\n'
      ' ip address 10.0.2.1 255.255.255.0\n'
      ' exit\n'
      '!'}
```

Terminal returner will print to screen:

```
# -----
# rt-1 rendering results
# -----
interface Gi1/1
description Customer A
encapsulation dot1q 100
vrf forwarding cust_a
ip address 10.0.0.1 255.255.255.0
exit
!
interface Gi1/2
description Customer C
encapsulation dot1q 300
vrf forwarding cust_c
ip address 10.0.3.1 255.255.255.0
exit
!
# -----
# rt-2 rendering results
# -----
interface Gi1/2
description Customer B
encapsulation dot1q 200
vrf forwarding cust_b
ip address 10.0.2.1 255.255.255.0
exit
!
```

This returner useful for debugging or, for instance, when it is easier to copy produced results from terminal screen.

`ttr.plugins.returners.terminal_returner.dump(data_dict, **kwargs)`

This function prints results to terminal screen

Parameters

- **data_dict** – (dict) dictionary keyed by `result_name_key` where values are rendered results string
- **kwargs** – (dict) any additional arguments ignored

11.3 File Returner Plugin

Plugin Name: file

This plugin responsible for saving results to text files iterating over results dictionary keyed by `result_name_key`.

For example, if results `data_dict` might look like this:

```
{
  "rt-1": "interface Gi1/1\n"
    " description Customer A\n"
    " encapsulation dot1q 100\n"
    " vrf forwarding cust_a\n"
    " ip address 10.0.0.1 255.255.255.0\n"
    " exit\n"
    "!\n"
    "interface Gi1/2\n"
    " description Customer C\n"
    " encapsulation dot1q 300\n"
    " vrf forwarding cust_c\n"
    " ip address 10.0.3.1 255.255.255.0\n"
    " exit\n"
    "!",
  "rt-2": "interface Gi1/2\n"
    " description Customer B\n"
    " encapsulation dot1q 200\n"
    " vrf forwarding cust_b\n"
    " ip address 10.0.2.1 255.255.255.0\n"
    " exit\n"
    "!"
}
```

If `result_dir` argument set to `./Output/`, file returner will iterate over `data_dict` using keys as filenames populating files with values at the end `./Output/` directory will contain two files named `rt-1.txt` and `rt-2.txt` with respective content.

`ttr.plugins.returners.file_returner.dump(data_dict, result_dir='./Output/', **kwargs)`

Function to save results in text files.

Parameters

- **data_dict** – (dict) dictionary keyed by `result_name_key` where values are strings to save in text files
- **result_dir** – (str) OS path to directory to save results in
- **kwargs** – (dict) any additional arguments ignored

JINJA2 TEMPLATES COLLECTION

TBD

API REFERENCE

TTR main class reference

```
class ttr.ttr(data=None, data_plugin=None, data_plugin_kwargs=None, renderer='jinja2',
              validator='yangson', renderer_kwargs=None, templates='./Templates/', models_dir='./Models/',
              template_name_key='template', model_name_key='model', returner='self',
              returner_kwargs=None, result_name_key='device', processors=None, processors_kwargs=None,
              templates_dict=None, models_dict=None, validator_kwargs=None)
```

Main class to instantiate TTR object.

Parameters

- **data** – (str) type depends on data plugin in use, but can be an OS path string referring to YAML structured text file or CSV spreadsheet
- **data_plugin** – (str) name of data plugin to use to load data
- **data_plugin_kwargs** – (dict) arguments to pass on to data plugin
- **renderer** – (str) name of renderer plugin to use, default `jinja2`
- **renderer_kwargs** – (dict) arguments to pass on to renderer plugin
- **templates** – (str) OS path to directory or excel spreadsheet file with templates, defaults to `./Templates/` folder
- **template_name_key** – (str) name of the key in data items that reference template to use to render that particular datum, default `template`
- **returner** – (str) name of returner plugin to use, default `self`
- **returner_kwargs** – (dict) arguments to pass on to returner plugin
- **result_name_key** – (str) name of the key in data items value of which should be used as a key in results dictionary, default `device`
- **processors** – (list) list of processor plugins names to pass loaded data through, default is empty list - no processors applied
- **templates_dict** – (dict) dictionary of {template_name: template_content}
- **models_dir** – (str) OS path to directory or with data models, defaults to `./Models/` folder
- **model_name_key** – (str) name of the key in data items that reference model to use to validate that particular datum, default `model`
- **models_dict** – (dict) dictionary of {model_name: model_content}
- **validator** – (str) validator plugin to use to validate provided data against models, default is `yangson`

- **validator_kwargs** – (dict) arguments to pass on to validator plugin

load_data(*data*, *data_plugin=None*)

Method to load data to render.

Parameters

- **data** – (str) data to load, either OS path to data file or text
- **data_plugin** – (str) name of data plugin to load data, by default will choose data loader plugin based on file extension e.g. `xlsx`, `csv`, `yaml/yml`

load_models(*models_dir=None*, *model_plugin=None*, ***kwargs*)

Function to load models content to models dictionary.

Parameters

- **models_dir** – (str) OS path to directory with models, defaults to `./Models/` directory
- **model_plugin** – (str) models loader plugin to use - `yangson` (default)
- **kwargs** – any additional ***kwargs* to pass on to `model_plugin` call

load_templates(*template_name=""*, *template_content=""*, *templates=""*, *templates_plugin=""*, ***kwargs*)

Function to load templates content in templates dictionary.

Parameters

- **template_name** – (str) name of template to load
- **template_content** – (str) template content to save in templates dictionary under `template_name`
- **templates** – (str) OS path to directory or file with templates, default `./Templates/`
- **templates_plugin** – (str) templates loader plugin to use - `base`, `xlsx`, `dir`, `file`, `ttr`
- **kwargs** – any additional ***kwargs* to pass on to `templates_plugin`

Decision logic:

1. If `template_content` provided add it to templates dictionary under `template_name` key
2. If valid `templates_plugin` name given use it to load template
3. Use base templates loader plugin to load template content

process_data(*data*)

Function to pass loaded data through a list of processor plugins.

Parameters **data** – (list) list of dictionaries data to process

Returns processed data

run()

Method to render templates with data and produce dictionary results keyed by `result_name_key`.

If `returner` set to `self`, will return results dictionary.

run_returner(*results=None*, *returner=None*, ***kwargs*)

Function to run returner to return results via plugin of choice.

Parameters

- **results** – (dict) results to run returner for
- **returner** – (str) returner plugin name e.g. `self`, `file`, `terminal`

- **kwargs** – (dict) additional arguments for returner plugin

validate_data(*data*)

Function to validate provided data

Parameters **data** – (list) list of dictionaries data to validate

Returns None

Running validation raises or logs error on validation failure depending on value of **on_fail** argument in **validator_kwargs**

PYTHON MODULE INDEX

t

- ttr.plugins.data.csv_loader, 24
- ttr.plugins.data.xlsx_loader, 21
- ttr.plugins.data.yaml_loader, 24
- ttr.plugins.models.yangson_model_loader, 31
- ttr.plugins.processors.filtering, 29
- ttr.plugins.processors.multitemplate_processor,
27
- ttr.plugins.processors.templates_split, 28
- ttr.plugins.renderers.jinja2_renderer, 39
- ttr.plugins.returners.file_returner, 42
- ttr.plugins.returners.self_returner, 41
- ttr.plugins.returners.terminal_returner, 41
- ttr.plugins.templates.base_template_loader,
35
- ttr.plugins.templates.dir_template_loader, 36
- ttr.plugins.templates.file_template_loader,
35
- ttr.plugins.templates.ttr_template_loader, 36
- ttr.plugins.templates.xlsx_template_loader,
37
- ttr.plugins.validate.validate_yangson, 33
- ttr.ttr, 1
- ttr.utils.cli, 17

INDEX

D

`dump()` (in module `ttr.plugins.returners.file_returner`), 43
`dump()` (in module `ttr.plugins.returners.self_returner`), 41
`dump()` (in module `ttr.plugins.returners.terminal_returner`), 42

L

`load()` (in module `ttr.plugins.data.csv_loader`), 24
`load()` (in module `ttr.plugins.data.xlsx_loader`), 24
`load()` (in module `ttr.plugins.data.yaml_loader`), 25
`load()` (in module `ttr.plugins.models.yangson_model_loader`), 33
`load()` (in module `ttr.plugins.templates.base_template_loader`), 35
`load()` (in module `ttr.plugins.templates.dir_template_loader`), 36
`load()` (in module `ttr.plugins.templates.file_template_loader`), 36
`load()` (in module `ttr.plugins.templates.ttr_template_loader`), 36
`load()` (in module `ttr.plugins.templates.xlsx_template_loader`), 37

M

module

`ttr.plugins.data.csv_loader`, 24
`ttr.plugins.data.xlsx_loader`, 21
`ttr.plugins.data.yaml_loader`, 24
`ttr.plugins.models.yangson_model_loader`, 31
`ttr.plugins.processors.filtering`, 29
`ttr.plugins.processors.multitemplate_processor`, 27
`ttr.plugins.processors.templates_split`, 28
`ttr.plugins.renderers.jinja2_renderer`, 39
`ttr.plugins.returners.file_returner`, 42
`ttr.plugins.returners.self_returner`, 41
`ttr.plugins.returners.terminal_returner`, 41
`ttr.plugins.templates.base_template_loader`, 35

`ttr.plugins.templates.dir_template_loader`, 36
`ttr.plugins.templates.file_template_loader`, 35
`ttr.plugins.templates.ttr_template_loader`, 36
`ttr.plugins.templates.xlsx_template_loader`, 37
`ttr.plugins.validate.validate_yangson`, 33
`ttr.ttr`, 1
`ttr.utils.cli`, 17

P

`process()` (in module `ttr.plugins.processors.filtering`), 30
`process()` (in module `ttr.plugins.processors.multitemplate_processor`), 28
`process()` (in module `ttr.plugins.processors.templates_split`), 29

R

`render()` (in module `ttr.plugins.renderers.jinja2_renderer`), 40

T

`ttr.plugins.data.csv_loader` module, 24
`ttr.plugins.data.xlsx_loader` module, 21
`ttr.plugins.data.yaml_loader` module, 24
`ttr.plugins.models.yangson_model_loader` module, 31
`ttr.plugins.processors.filtering` module, 29
`ttr.plugins.processors.multitemplate_processor` module, 27
`ttr.plugins.processors.templates_split` module, 28
`ttr.plugins.renderers.jinja2_renderer` module, 39

