AS3 Ninja

Release 0.6.1

Simon Kowallik

Feb 18, 2023

CONTENTS:

AS3 Ninja

CHAPTER

ONE

AS3 NINJA

AS3 Ninja

- Free software: ISC license
- Documentation: https://as3ninja.readthedocs.io
- Works with Python 3.8 and up

1.1 What is AS3 Ninja and what can it do for you?

AS3 Ninja is a templating engine as well as a validator for AS3 declarations. It offers a CLI for local usage, as well as a OpenAPI/Swagger based REST API.

AS3 Ninja empowers you to create AS3 declarations in a DevOps way by embracing the ideas of GitOps and CI/CD.

It separates Configuration from Code (Templates) as far as YOU wish.

AS3 Ninja let's you decide to scale between declarative and imperative paradigms to fit your needs.

What AS3 Ninja doesn't do:

- It does not provide you with a UI to create configurations
- It does not deploy AS3 configurations

1.2 Features

- Validate your AS3 Declarations against the AS3 Schema (via API, eg. for CI/CD) and AS3 specific formats
- Create AS3 Declarations from templates using the full power of Jinja2 (CLI and API)
  - reads your JSON or YAML configurations to generate AS3 Declarations
  - carefully crafted Jinja2 *as3ninja.jinja2.filters*, *as3ninja.jinja2.functions* and *as3ninja.jinja2.filterfunctions* further enhance the templating capabilities
- Use Git(hub) to pull template configurations and declaration templates

- HashiCorp Vault integration to retrieve secrets
- AS3 Ninja provides a simple CLI..
- ..and a REST API including a Swagger/OpenAPI interface at */api/docs* and */api/redoc* (openapi.json @ */api/openapi.json*)

1.3 AS3 Ninja Interface

Some impressions from the AS3 Ninja interfaces:

1.3.1 the Command Line

1.3.2 the API UI

ReDoc and Swagger UI:

Swagger UI demo:

1.4 Disclaimer and Security Note

AS3 Ninja is not a commercial product and *is not covered by any form of support, there is no contract nor SLA!*. Please read, understand and adhere to the license before use.

AS3 Ninja's focus is flexibility in templating and features, it is not hardened to run in un-trusted environments.

- It comes with a large set of dependencies, all of them might introduce security issues
- Jinja2 is not using a Sandboxed Environment and the *readfile* filter allows arbitrary file includes.
- The API is unauthenticated

> **Danger:** Only use AS3 Ninja in a secure environment with restricted access and trusted input.

1.5 Where to start?

Read the Docs and then Try it out! :-)

CHAPTER

TWO

CONCEPTS

To get started with AS3 Ninja first let's look at the concept.

The objective is to generate *AS3 Declarations* from templates where the parameterization of the template is done using a configuration.

AS3 Ninja uses the following main components to achieve this:

- Templates (Jinja2)
- Configurations (YAML and/or JSON)

The templates are also referred to as *Declaration Templates*, the configuration is referred to as *Template Configuration*.

AS3 Ninja doesn't force you into a declarative or imperative paradigm. You can just "fill the blanks" (declarative) or implement excessive logic within the *Declaration Templates* (imperative).

2.1 Workflow

The workflow to generate a deployable *AS3 Declaration* is a follows:

1. Get the *Declaration Template* and *Template Configuration* from the local filesystem or *Git*
2. Load, de-serialize and merge all *Template Configurations*
3. Feed the *Declaration Template* and *Template Configuration* to Jinja2
4. Render the *AS3 Declaration* using jinja2 ("transform the *Declaration Template* using the *Template Configuration*")
5. Validate the *AS3 Declaration* against the *AS3 Schema* (optional)

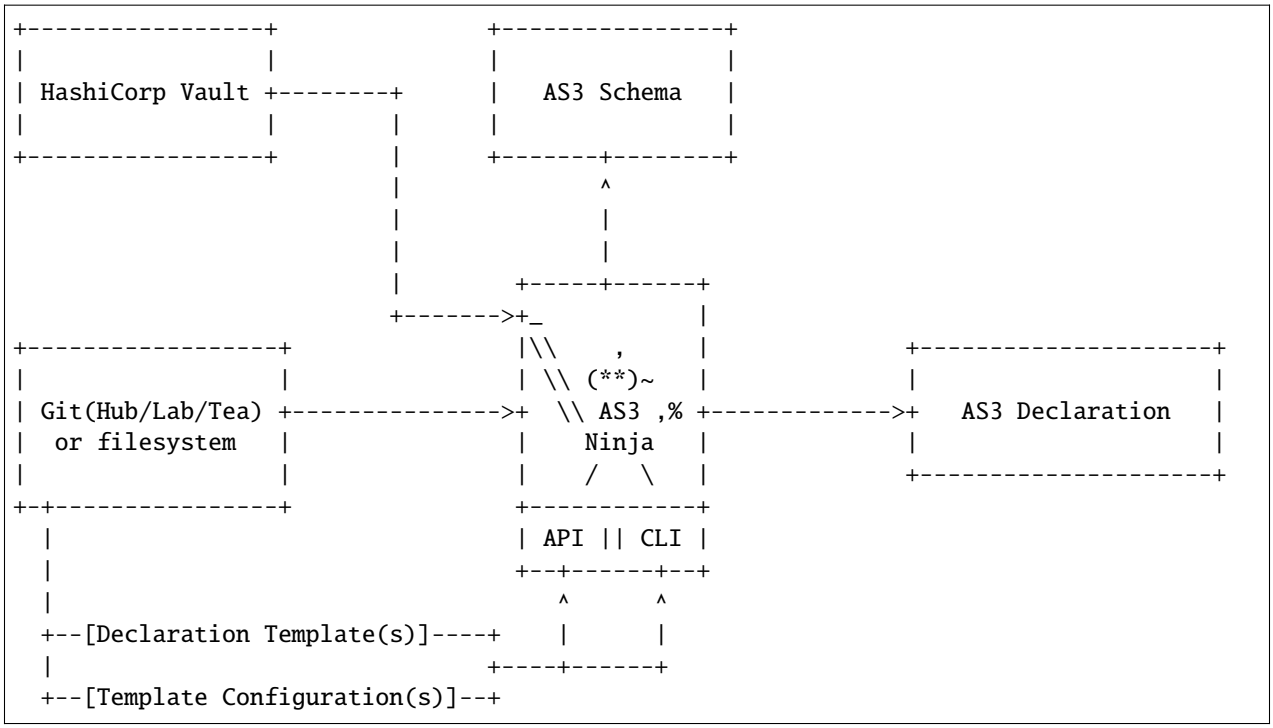
This workflow is also referred to as *transformation*.

2.2 Components

Let's look at the components at play.

Here is a diagram.

2.2.1 The AS3 Ninja ecosystem



2.2.2 AS3 Declaration

The *AS3 Declaration* is the JSON file ready to be pushed to the AS3 API. In DevOps terms it is an artifact (see also here). It contains all configuration elements needed for AS3 to create the configuration.

Note: *AS3 Declarations* often contain very sensitive data, like cryptographic keys or passwords.

2.2.3 Declaration Templates

Declaration Templates are Jinja2 templates, which can include further templates for specific *AS3 Declaration* components, e.g. for pools or profiles. Jinja2 offers a variety of imperative programming techniques like control structures.

The Jinja2 Template Designer Documentation is a highly recommended read.

Filters and Functions

Jinja2 offers filters and functions which can be used in templates.

AS3 Ninja comes with additional filters and functions which are specifically aimed at AS3.

See also:

- *as3ninja.jinja2.filterfunctions*
- *as3ninja.jinja2.filters*
- *as3ninja.jinja2.functions*

- *as3ninja.jinja2.tests*

2.2.4 Template Configuration

The *Template Configuration* are one or more YAML or JSON files. These define various variables, lists and in general contain data to be used in the *Declaration Template(s)*.

Multiple configuration files can be combined, where settings within the previous file are updated by all following files. This is quite powerful, as it allows to overwrite ("overlay") specific configuration parameters, for example for different environments (DEV/QA/PROD).

Note: It is recommended to avoid storing secrets within the Template Configuration.

2.2.5 AS3 Schema

Once the *AS3 Declaration* is generated from the *Declaration Template* using the *Template Configuration*, the resulting *artifact* can be validated against the *AS3 Schema*, which is available on the GitHub AS3 Repository.

Note: AS3 Ninja doesn't need to generate the *AS3 Declaration* to validate it. Any other declaration can be validated against the *AS3 Schema* using the API.

2.2.6 Git

Git has not only conquered the world of version control systems but is also very handy when you need to save, version, track and rollback any kind of configuration files. Therefore *Git* is a perfect place to store *Declaration Template(s)* as well as *Template Configuration(s)*.

AS3 Ninja can fetch from *Git* and automatically generate an *AS3 Declaration* for you.

2.2.7 Vault

AS3 Declarations often contain very sensitive data, these are commonly called *secrets* in the DevOps context. Hashicorp's Vault is a well established platform to manage any kind of secret and AS3 Ninja uses hvac to interface with vault.

AS3 Ninja retrieves relevant secrets during the transformation of the *AS3 Declaration*. The *Declaration Template* contains functions / filters which communicate to vault based on the settings within the template as well as the *Template Configuration*.

See *Vault Integration* for further details.

CHAPTER

THREE

USING AS3 NINJA

3.1 Run AS3 Ninja with Docker

Starting an ephemeral/temporary container for use of the API:

```
docker run -it --rm -p 8000:8000 simonkowallik/as3ninja
```

Creating a persistent container for CLI and API usage:

```
docker container create --name as3ninja -p 8000:8000 simonkowallik/as3ninja

# start the as3ninja container in background
docker start -a as3ninja
```

Important: The AS3 Schema files need to be downloaded from github.com to validate AS3 Declarations. AS3 Ninja `as3ninja.AS3Schema.updateschemas()` is doing that for you automatically, but the Docker Container will need access to https://github.com.

3.1.1 Using the CLI with Docker

Docker can be used to run the command line.

```
$ tree ./examples/simple/
./examples/simple/
├── http_path_header.iRule
├── ninja.yaml
├── sorry_page.iRule
└── template.j2
```

This example assumes the relevant Template Configurations and Declaration Templates are stored in `./examples/simple`.

```
as3ninja:
  declaration_template: "examples/simple/template.j2"
```

The `declaration_template` statement within the `ninja.yaml` provides the template location as `examples/simple/template.j2`. as3ninja expects to find the template at this location.

```
1 $ docker run --rm --tty --interactive \
2     --volume \$(pwd)/examples:/examples \
3     simonkowallik/as3ninja:latest \
4     as3ninja transform -c /examples/simple/ninja.yaml \
5     | jq ."
```

Instructs docker to bind mount the `$(pwd)/examples` folder to `/examples` (line 2) for the container image `simonkowallik/as3ninja:latest` (line 3).

Docker then executes `as3ninja transform -c /examples/simple/ninja.yaml` (line 4) within the container and pipes the result to `jq .`.

Todo: more cli examples

3.2 Command Line Usage

```
# for manual system wide installation (not recommended)
$ git clone https://github.com/simonkowallik/as3ninja
$ cd as3ninja
$ poetry build
$ pip3 install $(ls build/as3ninja*.whl)
```

```
# via PyPI using pip
$ pip=$(type -p pip3 || type -p pip)
$ $pip install as3ninja
```

3.3 API Usage

Use `curl` or `httpie` to query all available AS3 Schema versions:

```
$ http localhost:8000/api/schema/versions

$ curl -s localhost:8000/api/schema/versions | jq .
```

Navigate to http://localhost:8000/api/docs and http://localhost:8000/api/redoc to explore the API.

Todo: Postman collection for API calls

3.4 Validating a declaration

Using an ephemeral container with `docker run`:

```
$ docker run -it --rm -v $PWD/declaration.json:/declaration.json \
    simonkowallik/as3ninja:latest \
    as3ninja validate -d /declaration.json
INFO: Validation passed for AS3 Schema version: 3.22.1

$ docker run -it --rm -v $PWD/declaration.json:/declaration.json \
    simonkowallik/as3ninja:latest \
    as3ninja validate -d /declaration.json --version 3.17.0
INFO: Validation passed for AS3 Schema version: 3.17.0
```

Using the API via `curl`:

```
# start the docker container on port 8000
docker run -d --rm -p 8000:8000 simonkowallik/as3ninja:latest
6dd7a4a9cc65f84974a122e0605dd74fe087a7e61e67298e529bcd96fa133c7

# POST declaration to /api/schema/validate endpoint (curl)
curl -s http://localhost:8000/api/schema/validate -d @declaration.json | jq .
{
  "valid": true,
  "error": null
}

# POST declaration to /api/schema/validate endpoint (httpie)
cat $PWD/examples/dynamic-irule/declaration.json | \
    http POST 'localhost:8000/api/schema/validate?version=3.20.0'
HTTP/1.1 200 OK
content-length: 27
content-type: application/json
date: Sun, 13 Sep 2020 12:14:03 GMT
server: uvicorn

{
    "error": null,
    "valid": true
}
```

3.5 Postman collection

An AS3 Ninja Postman collection is available on Github.

3.6 Python Package

Todo: Update usage as a module

To use AS3 Ninja in your python project:

```
from as3ninja.schema import AS3Schema, AS3ValidationError
from as3ninja.declaration import AS3Declaration

# Declaration Template (str)
declaration_template = """
{
    "class": "AS3",
    "declaration": {
        "class": "ADC",
        "schemaVersion": "3.11.0",
        "id": "urn:uuid:{{ uuid() }}",
        "{{ ninja.Tenantname }}": {
            "class": "Tenant"
        }
    }
}
"""

# Template Configuration (dict)
template_configuration = {
    "Tenantname": "MyTenant"
}

# generate the AS3 Declaration
as3declaration = AS3Declaration(
    template_configuration=template_configuration,
    declaration_template=declaration_template
    )

from pprint import pprint
# the transformed AS3 Declaration is available via the declaration attribute
pprint(as3declaration.declaration)
{'class': 'AS3',
 'declaration': {'MyTenant': {'class': 'Tenant'},
                 'class': 'ADC',
                 'id': 'urn:uuid:f3850951-4a63-43ec-b2a3-28ab2c315479',
                 'schemaVersion': '3.11.0'}}

# create an AS3 schema instance
as3schema = AS3Schema()
```

(continues on next page)

(continued from previous page)

```

# Validate the AS3 Declaration against the AS3 Schema (latest version)
try:
    as3schema.validate(declaration=as3declaration.declaration)
except AS3ValidationError:
    # an AS3ValidationError exception is raised when the validation fails
    raise
```

CHAPTER

FOUR

TEMPLATING 101

As Jinja2 is used as the templating engine it is highly recommended to familiarize yourself with Jinja2.

Here are several helpful articles:

- Jinja2 Background
- About Template Engines

And finally the Jinja2 Template Designer Documentation, a must read for Jinja2 template authors.

4.1 Template Configuration

The Template Configuration provides data, influences logic and control structures or points to further resources.

All data of the Template Configuration is available to the Declaration Template as python data structures and can be accessed through the `ninja` namespace.

4.1.1 Template Configuration Files

The Template Configuration is generated from Template Configuration Files.

Two data formats are supported as Template Configuration Files:

- YAML
- JSON

Combining Multiple Template Configuration Files is supported by AS3 Ninja, we discuss the details later.

Note: There are many Pros and Cons about JSON vs. YAML. While it is out of scope to discuss this in detail, YAML is often easier to start with for simple use-cases. Two good articles about the challenges YAML and JSON introduce for use as configuration files:

- The downsides of JSON for config files
- YAML: probably not so great after all

An example:

```
services:
  My Web Service:
    type: http
```

(continues on next page)

(continued from previous page)

```
4     address: 198.18.0.1
5     irules:
6       - ./files/irules/myws_redirects.iRule
7     backends:
8       - 192.0.2.1
9       - 192.0.2.2
```

The highlighted lines provide data which will be used in the Declaration Template to fill out specific fields, like the desired name for the service (line 2), its IP address and backend servers.

```
1  services:
2    My Web Service:
3      type: http
4      address: 198.18.0.1
5      irules:
6        - ./files/irules/myws_redirects.iRule
7      backends:
8        - 192.0.2.1
9        - 192.0.2.2
```

On line 3 `type: http` is used to indicate the service type. This information is used in the Declaration Template logic to distinguish between types of services and apply type specific settings.

```
1  services:
2    My Web Service:
3      type: http
4      address: 198.18.0.1
5      irules:
6        - ./files/irules/myws_redirects.iRule
7      backends:
8        - 192.0.2.1
9        - 192.0.2.2
```

`irules` on line 5 references to a list of iRule files. The logic within the Declaration Template can use this list to load the iRule files dynamically and add them to the service.

4.1.2 as3ninja namespace

The namespace `as3ninja` within the Template Configuration is reserved for AS3 Ninja specific directives and configuration values.

Here is an overview of the current `as3ninja` namespace configuration values.

```
1  as3ninja:
2    declaration_template: /path/to/declaration_template_file.j2
```

The `declaration_template` points to the Declaration Template File on the filesystem. It is optional and ignored when a Declaration Template is referenced explicitly, for example through a CLI parameter.

The `as3ninja` namespace is accessible under the `ninja` namespace, as with any other data from Template Configurations.

Caution: The `as3ninja` namespace is reserved and might be used by additional integrations, therefore it should not be used for custom configurations.

Back to our service example:

```
as3ninja:
  declaration_template: ./files/templates/main.j2
services:
  My Web Service:
    type: http
    address: 198.18.0.1
    irules:
      - ./files/irules/myws_redirects.iRule
    backends:
      - 192.0.2.1
      - 192.0.2.2
```

We extended our Template Configuration with the `declaration_template` directive to point to the Declaration Template `./files/templates/main.j2`. AS3 Ninja will use this Declaration Template unless instructed otherwise (eg. through a CLI parameter).

4.1.3 Git and the *as3ninja* namespace

In addition `as3ninja.git` is updated during runtime when using AS3 Ninja's *Git* integration. It holds the below information which can be used in the Declaration Template.

```
as3ninja:
  git:
    commit:
      id:       commit id (long)
      id_short: abbreviated commit id
      epoch:    unix epoch of commit
      date:     human readable date of commit
      subject:  subject of commit message
    author:
      name:     author's name of commit message
      email:    author's email
      epoch:    epoch commit was authored
      date:     human readable format of epoch
    branch:     name of the branch
```

To use the short git commit id within the Declaration Template you would reference it as `ninja.as3ninja.git.commit.id_short`.

Note: Git Authentication is not explicitly supported by AS3 Ninja.

However there are several options:

1. AS3 Ninja invokes the *git* command with privileges of the executing user, hence the same authentication facilities apply.
2. Implicitly providing credentials through the URL should work: `https://<username>:<password>@gitsite.domain/repository`

When using Github: Personal Access Tokens can be used instead of the user password.

3. .netrc, which can be placed in the docker container at `/as3ninja/.netrc`, see confluence.atlassian.com : Using the .netrc file for an example.

4.1.4 Merging multiple Template Configuration Files

AS3 Ninja supports multiple Template Configuration Files. This provides great flexibility to override and extend Template Configurations.

Template Configuration Files are loaded, de-serialized and merged in the order specified. Starting from the first configuration every following configuration is merged into the Template Configuration. As the de-serialization takes place before merging, JSON and YAML can be combined.

Let's use our previous example, and add two additional Template Configuration Files. `as3ninja` is removed for conciseness.

```yaml
# main.yaml
services:
  My Web Service:
    type: http
    address: 198.18.0.1
    irules:
      - ./files/irules/myws_redirects.iRule
    backends:
      - 10.0.2.1
      - 10.0.2.2
```

```yaml
# internal_service.yaml
services:
  My Web Service:
    address: 172.16.0.1
    backends:
      - 172.16.2.1
      - 172.16.2.2
```

```yaml
# backends_dev.yaml
services:
  My Web Service:
    backends:
      - 192.168.200.1
      - 192.168.200.2
```

`main.yaml` is our original example. `internal_service.yaml` specifies the same `My Web Service` and contains two keys: `address` and `backends`. `backends_dev.yaml` again contains our `My Web Service` but only lists different `backends`.

When AS3 Ninja is instructed to use the Template Configurations Files in the order:

1. `main.yaml`
2. `internal_service.yaml`

AS3 Ninja loads, de-serializes and then merges the configuration. This results in the below python dict.

```
# merged: main.yaml, internal_service.yaml
{
  'services': {
    'My Web Service': {
      'address': '172.16.0.1',
      'backends': ['172.16.2.1', '172.16.2.2'],
      'irules': ['./files/irules/myws_redirects.iRule'],
      'type': 'http',
    }
  }
}
```

`'address'` and `'backends'` was overridden by the data in `internal_service.yaml`.

When AS3 Ninja is instructed to use all three Template Configurations Files in the order:

1. `main.yaml`
2. `internal_service.yaml`
3. `backends_dev.yaml`

The resulting python dict looks as below.

```
# merged: main.yaml, internal_service.yaml, backends_dev.yaml
{
  'services': {
    'My Web Service': {
      'address': '172.16.0.1',
      'backends': ['192.168.200.1', '192.168.200.2'],
      'irules': ['./files/irules/myws_redirects.iRule'],
      'type': 'http',
    }
  }
}
```

The `'address'` and `'backends'` definition was first overridden by the data in `internal_service.yaml` and `'backends'` was then again overridden by `backends_dev.yaml`.

Important: Please note that sequences (lists, arrays) are not merged, they are replaced entirely.

4.1.5 Including further Template Configurations using *as3ninja.include* namespace

Further Template Configuration files can be included using `include` within the `as3ninja` namespace.

Combined with the ability to merge multiple Template Configuration files, this becomes a powerful feature which can raise complexity. So use with care.

Important rules for using `as3ninja.include`:

1. Files included via `as3ninja.include` cannot include further Template Configuration files.
2. All Template Configuration files supplied to *as3ninja* can use `as3ninja.include`.
3. Every file included via `as3ninja.include` will only be included once, even if multiple configuration files reference this file.

4. Files will be included in the order specified.
5. Files are included just after the current configuration file (containing the include statement).
6. When filename and/or path globbing is used, all matching files will be included alphabetically.
7. Finally when all includes have been identified `as3ninja.include` will be updated with the full list of all includes in the order loaded.

The following example illustrates the behavior. Suppose we have the below tree structure and three Template Configuration files.

```
./configs
├── one.yaml
├── second
│   ├── 2a.yaml
│   ├── 2b.yaml
│   └── 2c.yaml
└── third
    ├── 3rd.yaml
    ├── a
    │   ├── 3a.yaml
    │   └── a2
    │       └── 3a2.yaml
    ├── b
    │   ├── 3b1.yaml
    │   └── 3b2.yaml
    └── c
        └── 3c.yaml
```

```yaml
# first.yaml
as3ninja:
  include: ./configs/one.yaml  # a single file include can use key:value
```

```yaml
# second.yaml
as3ninja:
  include:  # multiple file includes require a list
    - ./configs/second/2c.yaml  # explicitly include 2c.yaml first
    - ./configs/second/*.yaml  # include all other files
    # The above order ensures that 2c.yaml is merged first and the
    # remaining files are merged afterwards.
    # 2c.yaml will not be imported twice, hence this allows to
    # control merge order with wildcard includes.
```

```yaml
# third.yaml
as3ninja:
  include:
    - ./configs/third/**/*.yaml  # recursively include all .yaml files
    - ./configs/one.yaml  # try including one.yaml again
```

This will result in the following list of files, which will be merged to one configuration in the order listed:

```
first.yaml
configs/one.yaml
second.yaml
```

(continues on next page)

(continued from previous page)

```
configs/second/2c.yaml  # notice 2c.yaml is included first
configs/second/2a.yaml
configs/second/2b.yaml
third.yaml
configs/third/3rd.yaml
configs/third/a/3a.yaml
configs/third/a/a2/3a2.yaml
configs/third/b/3b1.yaml
configs/third/b/3b2.yaml
configs/third/c/3c.yam
# notice that configs/one.yaml is not included by third.yaml
```

Assume every YAML file has an `data: <filename>` entry and you have a *template.jinja2* with `{{ ninja | jsonify }}`.

```
as3ninja transform --no-validate -t template.jinja2 \
  -c first.yaml \
  -c second.yaml \
  -c third.yaml \
  | jq .
```

would produce:

```
{
  "as3ninja": {
    "include": [
      "configs/one.yaml",
      "configs/second/2c.yaml",
      "configs/second/2a.yaml",
      "configs/second/2b.yaml",
      "configs/third/3rd.yaml",
      "configs/third/a/3a.yaml",
      "configs/third/a/a2/3a2.yaml",
      "configs/third/b/3b1.yaml",
      "configs/third/b/3b2.yaml",
      "configs/third/c/3c.yaml"
    ]
  },
  "data": "configs/third/c/3c.yaml"
}
```

Note: The above example is intended to demonstrate the behavior but could be seen as an example for bad practice due to the include complexity.

4.1.6 Including further YAML files using *!include*

AS3 Ninja uses a custom yaml `!include` tag which provides additional functionality to include further YAML files.

`!include` is followed by a filename (including the path from the current working directory) or a python list of filenames. The filename(s) can include a globbing pattern following the rules of python3's pathlib Path.glob.

Note: Nesting `!include` is possible, e.g. *a.yaml* includes *b.yaml* which includes *c.yaml* but should be avoided in favor of a cleaner and more understandable design.

Suppose we have the below tree structure:

```
.
├── main.yaml
└── services
    ├── A
    │   ├── serviceA1.yaml
    │   ├── serviceA2.yaml
    │   └── serviceA3.yaml
    └── B
        ├── serviceB1.yaml
        └── serviceB2.yaml
```

Each *serviceXY.yaml* file contains definitions for its service, for example:

```yaml
ServiceXY:
  address: 198.18.x.y
```

In *main.yaml* we use `!include` to include the *serviceXY.yaml* files as follows:

```yaml
# Use globbing to traverse all subdirectories in `./services/`
# and include all `.yaml` files:
all_services: !include ./services/**/*.yaml

# simply include a single yaml file:
service_a1: !include ./services/A/serviceA1.yaml

# include a single yaml file but make sure it is included as a list element:
service_b1_list: !include [./services/B/serviceB1.yaml]

# include two yaml files explicitly:
service_a2_b2: !include [./services/A/serviceA2.yaml, ./services/B/serviceB2.yaml]

# include all files matching serviceB*.yaml in the directory ./services/B/
services_b: !include ./services/B/serviceB*.yaml
```

The above yaml describes all syntaxes of `!include` and is equivalent to the below yaml.

Please specifically note the behavior for the following examples:

- *all_services* contains a list of all the yaml files the globbing pattern matched.
- *service_a1* only contains the one yaml file, because only one file was specified, it is included as an object not a list.
- *service_a2_b2* contain a list with the entries of serviceA2.yaml and serviceB2.yaml

- *service_b1_list* includes only serviceB1.yaml but as a list entry due to the explicit use of *[]*

Note: Also note that the above paths are relative to the CWD where as3ninja is executed. That means if *ls ./services/A/serviceA2.yaml* is successful running as3ninja from the current directory will work as well.

```yaml
all_services:
  - ServiceA2:
      address: 198.18.1.2
  - ServiceA3:
      address: 198.18.1.3
  - ServiceA1:
      address: 198.18.1.1
  - ServiceB2:
      address: 198.18.2.2
  - ServiceB1:
      address: 198.18.2.1

service_a1:
  ServiceA1:
    address: 198.18.1.1

service_b1_list:
  - ServiceB1:
      address: 198.18.2.1

service_a2_b2:
  - ServiceA2:
      address: 198.18.1.2
  - ServiceB2:
      address: 198.18.2.2

services_b:
  - ServiceB2:
      address: 198.18.2.2
  - ServiceB1:
      address: 198.18.2.1
```

It is important to note that `!include` does not create a "new yaml file" similar to the above example, instead it de-serializes the *main.yaml* file and treats `!include` as an "instruction", which then de-serializes the files found based on the `!include` statement.

So de-serializing the *main.yaml* actually results in the below python data structure (dict):

```
{
  "all_services": [
    { "ServiceA2": { "address": "198.18.1.2" } },
    { "ServiceA3": { "address": "198.18.1.3" } },
    { "ServiceA1": { "address": "198.18.1.1" } },
    { "ServiceB2": { "address": "198.18.2.2" } },
    { "ServiceB1": { "address": "198.18.2.1" } }
  ],
  "service_a1": { "ServiceA1": { "address": "198.18.1.1" } },
  "service_b1_list": [
```

(continues on next page)

(continued from previous page)

```
    { "ServiceB1": { "address": "198.18.2.1" } }
  ],
  "service_a2_b2": [
    { "ServiceA2": { "address": "198.18.1.2" } },
    { "ServiceB2": { "address": "198.18.2.2" } }
  ],
  "services_b": [
    { "ServiceB2": { "address": "198.18.2.2" } },
    { "ServiceB1": { "address": "198.18.2.1" } }
  ]
}
```

Caution: `!include` does not prevent against circular inclusion loops, which would end in a RecursionError exception.

4.1.7 Default Template Configuration File

If no Template Configuration File is specified, AS3 Ninja will try to use the first of the following files.

1. `./ninja.json`
2. `./ninja.yaml`
3. `./ninja.yml`

This is useful if you do not need multiple Template Configuration Files or only occasionally need them.

4.2 Declaration Template

The Declaration Template defines how the configuration is used to render an AS3 Declaration.

Declaration Templates use the Template Configuration, which is available in the Jinja2 Context.

4.2.1 A question of paradigms: Declarative or Imperative

If you thought you already choose the declarative paradigm with AS3 you are mostly correct. The AS3 Declaration is declarative.

But how do you produce the AS3 Declaration?

This is where AS3 Ninja and specifically Jinja2 comes into play. Jinja2 provides a wide spectrum between declarative and imperative to fit your specific needs.

A quick overview of Imperative vs. Declarative Programming, which can help understand the topic better: Imperative vs Declarative Programming

AS3 Ninja the declarative way

Let's look at a declarative way to render an AS3 Declaration.

```
{# Declaration Template #}
{
  "class": "AS3",
  "declaration": {
    "class": "ADC",
    "schemaVersion": "3.11.0",
    "id": "urn:uuid:{{ ninja.uuid }}",
    "{{ ninja.tenant }}": {
      "class": "Tenant",
      "{{ ninja.app.name }}": {
        "class": "Application",
        "template": "http",
        "backends": {
          "class": "Pool",
          "monitors": ["http"],
          "members": [
            {
              "servicePort": 80,
              "serverAddresses": [ {{ ninja.app.backends }} ]
            }
          ]
        },
        "serviceMain": {
          "class": "Service_HTTP",
          "virtualAddresses": ["{{ ninja.app.address }}"],
          "pool": "backends"
        }
      }
    }
  }
}
```

The above Declaration Template uses Jinja2 to fill specific values using variables. No logic, no control structures nor commands are used.

```
# Template Configuration
tenant: MyTenant
uuid: 2819307c-d8c3-4d1e-911e-40889e1df6c7
app:
  name: MyApp
  address: 198.18.0.1
  backends: "\"192.168.0.1\", \"192.168.0.2\""
```

Above is an example Template Configuration for our Declaration Template. As our backends are expected to be a JSON array, the value of `backends` isn't very pretty.

Adding additional services, tenants or service specific configurations will require changes in the Template Configuration as well as the Declaration Template.

AS3 Ninja the imperative way

Now let's find an imperative way to render a similar AS3 Declaration.

```
{# Declaration Template #}
{
  "class": "AS3",
  "declaration": {
    "class": "ADC",
    "schemaVersion": "3.11.0",
    "id": "urn:uuid:{{ uuid() }}",
    {% for tenant in ninja.tenants %}
    "{{ tenant.name }}": {
      "class": "Tenant",
      {% for app in tenant.apps %}
      "{{ app.name }}": {
        "class": "Application",
        "template": "{{ app.type }}",
        "backends": {
          "class": "Pool",
            "monitors":
            {% if app.monitors is defined %}
                {{ app.monitors | jsonify }},
            {% else %}
                {{ ninja.mappings.monitor[app.type] | jsonify }},
            {% endif %}
            "members": {{ app.backends | jsonify }}
        },
        "serviceMain": {
          "class": "{{ ninja.mappings.service[app.type] }}",
          "virtualAddresses": {{ app.address | jsonify }},
          "pool": "backends"
        }
      }
    {% if not loop.last %},{% endif %}
    {% endfor %}
    }
  {% if not loop.last %},{% endif %}
  {% endfor %}
  }
}
```

This Declaration Template not only uses Jinja2 to fill specific values using variables but also uses control structures, mainly loops and conditions (highlighted), to render the AS3 Declaration.

You can already see that this Declaration Template iterates over a list of tenants and a list of apps for each tenant. This clearly shows this example is probably easy to extend with additional tenants and apps.

As this Declaration Template contains a lot more details we will take a closer look at each step, but first let's have a look at the Template Configuration:

```
# Template Configuration
tenants:
- name: MyTenant
  apps:
```

(continues on next page)

(continued from previous page)

```
  - name: MyApp
    type: http
    address:
    - 198.18.0.1
    backends:
    - servicePort: 80
      serverAddresses:
      - 192.168.0.1
      - 192.168.0.2
mappings:
  service:
    http: Service_HTTP
  monitor:
    http:
    - http
```

The Template Configuration is longer than the previous *declarative* example, but it is also more flexible. The non-pretty representation of the backends has been replaced with a more flexible `backends` definition (highlighted).

As this Configuration Template works hand in hand with the Declaration Template we will take a closer look at both in the next section.

4.2.2 Building a Declaration Template

A *declarative* Declaration Template and the corresponding Template Configuration is pretty straightforward as you saw earlier.

So instead we will look at the *imperative* example above and walk through each step. For conciseness we will remove parts from the Declaration Template and Template Configuration and focus on the subject.

Looping Tenants and their Apps

```
# Template Configuration
tenants:
- name: MyTenant
  # ... tenant specific configuration
  apps:
  - name: MyApp
    type: http
    # ... app specific configuration
```

The above Template Configuration excerpt contains a list of Tenants (line 2) with the first list entry having `name` key with value `MyTenant` (line 3). Within this Tenant a list of Applications (Apps) is defined (line 5), with the first list entry having a `name` key with value `MyApp` (line 6).

```
{# Declaration Template #}
{
  "class": "AS3",
  {# ... more code ... #}
    {% for tenant in ninja.tenants %}
    "{{ tenant.name }}": {
```

(continues on next page)

(continued from previous page)

```
7         "class": "Tenant",
8         {% for app in tenant.apps %}
9         "{{ app.name }}": {
10        {# ... app specific code ... #}
11        }
12      {% if not loop.last %},{% endif %}
13      {% endfor %}
14      }
15    {% if not loop.last %},{% endif %}
16    {% endfor %}
17    }
18  }
```

The Declaration Template is built to iterate over a list of Tenants (line 5). The Template Configuration list of Tenants is accessible via `ninja.tenants` and each Tenant is assigned to `tenant`, which is now available within the for loop. On line 6 the Tenant name is read from `tenant.name`.

Furthermore on line 8 the Declaration Template will iterate the list of Applications defined for this Tenant. The list of Applications for this particular Tenant is available via `tenant.apps`. `apps` refers to the definition in the Template Configuration (on line 5). The Application specific configuration starts on line 9, where `app.name` is used to declarative the Application class of the AS3 Declaration.

Line 12 is checking for the last iteration of the inner "Application loop" and makes sure the comma (`,`) is included when there are further elements in the Application list. This is important as JSON does not tolerate a trailing comma. Line 13 defines the end of the loop.

The same is done on line 15 and 16 for the outer "Tenants loop".

Note: More details on control structures in Jinja2 can be found at List of Control Structures in the Jinja2 Template Designer Documentation.

Application specific settings

Now let's look at the Application specific settings.

```
1  # Template Configuration
2  tenants:
3  - name: Tenant1
4    apps:
5    - name: MyApp
6      type: http
7      address:
8      - 198.18.0.1
9      backends:
10     - servicePort: 80
11       serverAddresses:
12       - 192.168.0.1
13       - 192.168.0.2
14 mappings:
15   service:
16     http: Service_HTTP
```

(continues on next page)

(continued from previous page)

```
17  monitor:
18    http:
19    - http
```

The YAML is more structured to not only fit the Declaration Template but also the AS3 data structures. A `mappings` data structure was added to assist with default values / mappings to Application types.

```
1  {# Declaration Template #}
2    {# ... more code ... #}
3    "{{ app.name }}": {
4      "class": "Application",
5      "template": "{{ app.type }}",
6      "backends": {
7        "class": "Pool",
8          "monitors":
9          {% if app.monitors is defined %}
10             {{ app.monitors | jsonify }},
11         {% else %}
12             {{ ninja.mappings.monitor[app.type] | jsonify }},
13         {% endif %}
14         "members": {{ app.backends | jsonify }}
15     },
16     "serviceMain": {
17       "class": "{{ ninja.mappings.service[app.type] }}",
18       "virtualAddresses": {{ app.address | jsonify }},
19       "pool": "backends"
20     {# ... more code ... #}
```

The `app.type` is used on line 5 to map to the `http` AS3 template, on line 12 `app.type` is used again as a key for `mappings.service`. This allows us to create multiple *App type* to *Service_<type>* mappings. In this case `http` maps to the AS3 service class `Service_HTTP`.

Line 9-13 deals with monitors, if `app.monitors` is defined it is used, otherwise `app.type` is used again to lookup the default monitor to use, based on the Template Configuration (line 17-19). Note that `"monitors"` is expected to be a JSON array of monitors, this is why the Template Configuration YAML uses a list for `monitor.http`. `jsonify` is an AS3 Ninja Filter (see *as3ninja.jinja2.filterfunctions.jsonify()*) which will convert any "piped" data to a valid JSON format. A python list (which the YAML de-serializes to) is converted to a JSON array.

The `"members"` key for a *AS3 Pool class* is expected to be a list, each list entry is an object with several key:value pairs. `serverAddresses` are again expected to be a list of IP addresses.

Looking at the `backends` part of the Template Configuration again:

```
1    backends:
2    - servicePort: 80
3      serverAddresses:
4      - 192.168.0.1
5      - 192.168.0.2
```

`app.backends` and it's YAML exactly represents this structure, making it easy for the Declaration Template to just convert it to JSON (using the `jsonify` filter). Sometimes it is easier to look at the resulting JSON, as it is used by AS3 as well. Here is how the above YAML for `backends` looks like:

```
{
  "backends": [
    {
      "servicePort": 80,
      "serverAddresses": ["192.168.0.1", "192.168.0.2"]
    }
  ]
}
```

"virtualAddresses", on line 18 Declaration Template, is also expected to be a JSON array, which is what the Template Configuration perfectly represents and `jsonify` converts to.

Adding more Tenants

Based on the above *imperative* example, it is easy to add further Tenants.

Here is an example adding one more Tenant:

```
# Template Configuration
tenants:
- name: Tenant1
  apps:
  - name: MyApp
    type: http
    address:
    - 198.18.0.1
    backends:
    - servicePort: 80
      serverAddresses:
      - 192.168.0.1
      - 192.168.0.2
- name: Tenant2
  apps:
  - name: TheirApp
    type: http
    address:
    - 198.18.100.1
    monitors:
    - http
    - icmp
    backends:
    - servicePort: 80
      serverAddresses:
      - 192.168.100.1
mappings:
  service:
    http: Service_HTTP
  monitor:
    http:
    - http
```

Adding an additional App type

What if we want to add an additional type of Application? Let's assume we want to add a SSH server, using AS3's *Service_TCP*.

As this service class doesn't come with a default value for `virtualPort` we will need to modify our Declaration Template.

```
{# Declaration Template #}
{
  "class": "AS3",
  "declaration": {
    "class": "ADC",
    "schemaVersion": "3.11.0",
    "id": "urn:uuid:{{ uuid() }}",
    {% for tenant in ninja.tenants %}
    "{{ tenant.name }}": {
      "class": "Tenant",
      {% for app in tenant.apps %}
      "{{ app.name }}": {
        "class": "Application",
        "template": "{{ app.type }}",
        "backends": {
          "class": "Pool",
            "monitors":
            {% if app.monitors is defined %}
                {{ app.monitors | jsonify }},
            {% else %}
                {{ ninja.mappings.monitor[app.type] | jsonify }},
            {% endif %}
            "members": {{ app.backends | jsonify }}
        },
        "serviceMain": {
          {% if app.port is defined %}
          "virtualPort": {{ app.port }},
          {% endif %}
          "class": "{{ ninja.mappings.service[app.type] }}",
          "virtualAddresses": {{ app.address | jsonify }},
          "pool": "backends"
        }
      }
    {% if not loop.last %},{% endif %}
    {% endfor %}
    }
  {% if not loop.last %},{% endif %}
  {% endfor %}
  }
}
```

We added a conditional check for `app.port` (line 26-28). If it is set, `"virtualPort"` will be added to the AS3 Declaration with the value of `app.port`. Of course this `app.port` can be used by other service types as well.

```
# Template Configuration
tenants:
```

(continues on next page)

(continued from previous page)

```
3  - name: Tenant1
4    apps:
5    - name: MyApp
6      type: http
7      address:
8      - 198.18.0.1
9      backends:
10     - servicePort: 80
11       serverAddresses:
12       - 192.168.0.1
13       - 192.168.0.2
14 - name: Tenant2
15   apps:
16   - name: TheirApp
17     type: http
18     address:
19     - 198.18.100.1
20     monitors:
21     - http
22     - icmp
23     backends:
24     - servicePort: 80
25       serverAddresses:
26       - 192.168.100.1
27   - name: TcpApp
28     type: tcp
29     port: 22
30     address:
31     - 198.18.100.1
32     backends:
33     - servicePort: 22
34       serverAddresses:
35       - 192.168.100.1
36 mappings:
37   service:
38     http: Service_HTTP
39     tcp: Service_TCP
40   monitor:
41     http:
42     - http
43     tcp:
44     - tcp
```

Line 29 has the new `port` key, which is used in the Declaration Template. Along with the TCP based service we also updated the mappings.

Hint: If you use Visual Studio Code, the jinja-json-syntax Syntax Highlighter is very helpful.

CHAPTER

FIVE

VAULT INTEGRATION

HashiCorp Vault is a reliable and secure secret management engine widely used in the DevOps community.

The integration in AS3 Ninja is based on hvac.

5.1 Background

The term *secrets* describes secret information, like private cryptographic keys for x509 certificates, passwords and other shared secrets. As secrets are often used to ensure confidentiality and integrity, it is crucial to prevent compromise. Configuration management and version control systems, like git(hub), are not well suited nor meant to hold secret information. HashiCorp Vault provides a solution to manage secrets for AS3 Ninja.

Different types of secrets exist, therefore Vault provides a variety of *Secrets Engines* to fullfil the specific needs of these secret types. Two *Secrets Engines* are useful particular for AS3 Ninja:

- KV1
- KV2

Both are Key Value stores, where KV2 provides versioning of secrets. See Vault Docs: Secrets Engines: Key/Value for more details.

5.2 Concept

AS3 Ninja is a client to Vault and retrieves secrets during AS3 Declaration generation. Although secrets management is a complicated topic the AS3 Ninja Vault integration is relatively straightforward. One important assumption is that the authentication to Vault is not performed directly by AS3 Ninja. Also see Vault Docs: Concepts: Authentication.

AS3 Ninja assumes that a *token* is provided, which represents an authenticated session to communicate with Vault. Generating/Fetching the token is out of scope of AS3 Ninja.

AS3 Ninja can also communicate to multiple Vault instances in case secrets are spread across different Vault deployments.

5.3 Vault Communication

Communication with Vault is established through Vault's REST API.

To initiate communication a three parameters are required:

1. Vault's address
2. communication parameters
3. A valid token

There are multiple ways to specify these parameters depending on how you use the Vault integration.

Vault can be accessed with *as3ninja.vault.vault*, which is available as Jinja2 filter as well as a Jinja2 function.

A specific *client* can be created using the Jinja2 function *as3ninja.vault.VaultClient*. This *client* is tied to the Vault instance defined by the parameters passed to *VaultClient*. The *vault* filter/function optionally accepts this *client* as a parameter.

This is helpful in case a specific Vault instance must be contacted or multiple Vault instances are needed.

The usage of *vault* and *VaultClient* is explained later.

When using *vault* and not passing a specific *client*, AS3 Ninja will use the *as3ninja.vault.VaultClient.defaultClient()* to initiate communication with Vault. The *defaultClient* connection logic is as follows:

1. It will first check if an authenticated vault connection exists already.

   This is helpful in case AS3 Ninja is executed from the command line and a Vault connection has been established using Vaults own cli.

2. If 1. isn't successful It will then check the Jinja2 Context for the namespace `ninja.as3ninja.vault` and use `addr`, `token` and `ssl_verify` for connection establishment

   This provides great flexibility as the Vault connection can be parametrized by the Template Configuration.

3. For any namespace variable in 2., it will check the environment variables `VAULT_ADDR`, `VAULT_TOKEN` and `VAULT_SKIP_VERIFY`.

   This allows to fallback to environment variables. This is helpful when AS3 Ninja is used through the cli. It is also very helpful when AS3 Ninja runs as a docker container as the default Vault connection can be specified on the container level.

4. If `VAULT_SKIP_VERIFY` doesn't exist, it will use `VAULT_SSL_VERIFY` from the AS3 Ninja configuration file (`as3ninja.settings.json`).

The variables in `ninja.as3ninja.vault` can be specified using the Template Configuration, for example:

```
as3ninja:
  vault:
    addr: https://192.0.2.100:8201
    token: s.jbm5eO3rmh1kxrraNA9Q0N5r
    ssl_verify: false
```

Note: Remember that anything defined in the Template Configuration will be stored in the `ninja` namespace within the Jinja2 context. That's why `ninja.as3ninja.vault` is used but the YAML example starts by defining `as3ninja:`.

5.4 Referencing Secrets in Template Configurations

To retrieve a secret from Vault a couple of parameters are required:

1. The *mount_point* of the Secrets Engine
2. The *path* of the Secret
3. The Secrets *engine*
4. The *version* (in case of Secrets Engine kv2)
5. The *filter* selects the exact piece of information required from the response

5.4.1 mount_point

If the *mount_point* is part of the *path* and is configured during setup of the Secrets *engine* in Vault. If the *mount_point* is just one level, for example */mySecretEngineKV2*, it can be omitted if it is part of *path*.

5.4.2 path

The *path* defines which secret to retrieve. If *mount_point* is omitted is must include the *mount_point*, see paragraph above.

5.4.3 engine

engine defines the Secrets Engine the secret is stored in. Default is KV2.

Supported Secrets Engines:

- KV1
- KV2

5.4.4 version

In case KV2 is used, secrets can be versioned. When *version* is provided, a specific version of the secret is fetched. Default is *version=0*, which is the most recent version. *version* is optional.

5.4.5 filter

filter is an optional setting and can be used to select a specific element from the Vault response. The filter is a string of keys separated by dots (e.g. *key1.key2.key3*). If a key contains a dot in the name, it can be escaped (e.g. *k\\.e\\.y.anotherKey* would be split to *k.e.y* and *anotherKey*).

5.4.6 Examples

```
secrets:
  myWebApp:
    path: /secretkv2/myWebApp/privateKey
```

The simplest definition of a secret just contains the path. *vault* will use the KV2 secrets *engine* and return the most recent *version* of the secret.

```
secrets:
  myAPI:
    path: /secretOne/myAPI/sharedKey
    engine: kv1
```

When using KV1, the *engine* must be explicitly specified.

```
secrets:
  v1Service:
    path: /otherService/privateKey
    mount_point: /SecEnginePath/myKV2
    version: 1
  latestService:
    path: /otherService/privateKey
    mount_point: /SecEnginePath/myKV2
```

Say a secrets engine was created with: `vault secrets enable -path=/SecEnginePath/myKV2 kv-v2`

As the path has multiple levels, the *mount_point* must be explicitly specified.

The secret *v1Service* references to a specific version of the secret (*version: 1*), where *latestService* refers to the most recent version. *latestService* could have used *version: 0* to explicitly state that the most recent version should be used but this is optional.

5.5 Using Vault with AS3 Ninja

Let's look at using *vault* as a jinja2 filter and function as well as using *VaultClient*.

Note: To keep the examples concise, none of the below produce a valid AS3 declaration. Therefore the *–no-validate* flag is required.

5.5.1 A simple example (Secrets Engine: KV1)

```
# Template Configuration
secrets:
  myAPI:
    path: /secretOne/myAPI/sharedKey
    engine: kv1
```

Our secret will be accessible during transformation of the Declaration Template as `ninja.secrets.myAPI`. `ninja.secrets.myAPI.path` will refer to the value `/secretOne/myAPI/sharedKey` and `ninja.secrets.myAPI.engine` will refer to `kv1`.

```
{# Declaration Template #}
{
  "myAPI": {{ ninja.secrets.myAPI | vault | jsonify }}
}
```

We use *vault* as a filter and the value of `ninja.secrets.myAPI` is passed as the first parameter automatically by jinja2. `vault` will read all keys in the passed parameter and try to retrieve the relevant secret from Vault.

Run as3ninja:

```
as3ninja transform -c ninja.yml -t template.j2 --no-validate | jq .
```

Resulting JSON:

```
{
  "myAPI": {
    "request_id": "308c8b5c-fadc-ff32-8543-ad611fc53d72",
    "lease_id": "",
    "renewable": false,
    "lease_duration": 2764800,
    "data": {
      "secretKey": "AES 128 4d3642df883756b0d5746f32463f6005"
    },
    "wrap_info": null,
    "warnings": null,
    "auth": null
  }
}
```

The value of `"myAPI"` contains details about the fetched Vault secret, probably more than needed. Likely we are only interested in a specific value, for example *data -> secretKey*. Modifying the Declaration Template like below would just extract this specific value:

```
{
  "myAPI": {{ (ninja.secrets.myAPI | vault)['data']['secretKey'] | jsonify }}
}
```

Using a *filter* in the secret's definition within the Template Configuration is a better alternative as this separates the configuration further from the implementation (the Declaration Template). Here is the updated Template Configuration:

```
# Template Configuration
secrets:
  myAPI:
    path: /secretOne/myAPI/sharedKey
    engine: kv1
    filter: data.secretKey
```

The resulting JSON now only contains the information we are looking for:

```
{
  "myAPI": "AES 128 4d3642df883756b0d5746f32463f6005"
}
```

5.5.2 Example using Secrets Engine KV2

```
  # Template Configuration
  latestService:
    path: /otherService/privateKey
    mount_point: /SecEnginePath/myKV2
```

```
{# Declaration Template #}
{
  "latestService": {{ ninja.secrets.latestService | vault | jsonify }}
}
```

Run as3ninja:

```
as3ninja transform -c ninja.yml -t template.j2 --no-validate | jq .
```

Resulting JSON:

```
{
  "latestService": {
    "request_id": "25b2debe-7514-de9a-8beb-dd798f898ddf",
    "lease_id": "",
    "renewable": false,
    "lease_duration": 0,
    "data": {
      "data": {
        "privateKey": "-----BEGIN RSA PRIVATE KEY-----\
↪nMIHzAgEAAjEAvAI1w37cQcrflizN6Qa6GYVO26Sup5J0WWirYDS1aoxXCjQDcN4Q\
↪nf7cCQ82kSzcjAgMBAAECMFS5sjzdiKjlogjtPAYNkAQ8PSNifYrqxlpT4D5+TpWj\
↪nM1ODUjTVZBPQXuUIJYo6gQIZAOBcs33j5C6k7sisCVAvJTCTmdMx037zYQIZANaF\
↪nLSMLGaEhYz1da3OR6IHyM9Anx/h9AwIZAL4vlq+GeKzZfth4jMR90malF+Yg/IlG\
↪nwQIZAJKgRqDMRoFfK9DW2MoOsgiX/xhJCKLs9wIYPHBqLjhfB5Ycuk+WyxHj2uNQ\nNpf7zbsE\n-----END␣
↪RSA PRIVATE KEY-----"
      },
      "metadata": {
        "created_time": "2019-11-30T13:05:16.5110593Z",
        "deletion_time": "",
        "destroyed": false,
        "version": 2
      }
    },
    "wrap_info": null,
    "warnings": null,
    "auth": null
  }
}
```

As we already know the result carries likely more information than we need. In contrast to KV1 the KV2 Secrets Engine uses one more level of nesting as it does provide explicit metadata (line 11) about the secret. The information we are looking for is found at *data* -> *data* -> *privateKey* (line 7-9). Within the secret's metadata the version of the retrieved secret is displayed (`"version": 2` at line 15).

As we already learnt we can filter the response data by either updating the Declaration Template or using the filter. Updated Template Configuration:

```
  # Template Configuration
  latestService:
    path: /otherService/privateKey
    mount_point: /SecEnginePath/myKV2
    filter: data.privateKey
```

Note: Although KV2 stores the *privateKey* in *data* -> *data* we can omit the first instance of *data* as this is automatically prepended by the vault jinja2 filter/function. If you would like to access the *version* in the *metadata* the filter would be *metadata.version*.

Result:

```
{
  "latestService": "-----BEGIN RSA PRIVATE KEY-----\
↪nMIHzAgEAAjEAvAI1w37cQcrflizN6Qa6GYVO26Sup5J0WWirYDS1aoxXCjQDcN4Q\
↪nf7cCQ82kSzcjAgMBAAECMFS5sjzdiKjlogjtPAYNkAQ8PSNifYrqxlpT4D5+TpWj\
↪nM1ODUjTVZBPQXuUIJYo6gQIZAOBcs33j5C6k7sisCVAvJTCTmdMx037zYQIZANaF\
↪nLSMLGaEhYz1da3OR6IHyM9Anx/h9AwIZAL4vlq+GeKzZfth4jMR90malF+Yg/IlG\
↪nwQIZAJKgRqDMRoFfK9DW2MoOsgiX/xhJCKLs9wIYPHBqLjhfB5Ycuk+WyxHj2uNQ\nNpf7zbsE\n-----END␣
↪RSA PRIVATE KEY-----"
}
```

5.5.3 Using *vault* as a jinja2 function

Note: The below example is based on the KV2 example above

We can use *vault* as a jinja2 function as well. This allows us to implement more generic queries and re-use the secret information without asking Vault all the time.

```
{
{% set s = namespace() %}
{% set s.latestService = vault(secret=ninja.secrets.latestService, filter="") %}
{% set s.latestService_privKey = s.latestService['data']['data']['privateKey'] %}
{% set s.latestService_ver = s.latestService['data']['metadata']['version'] %}
    "latestService_privateKey": {{ s.latestService_privKey | jsonify }},
    "latestService_version": {{ s.latestService_ver | jsonify }}
}
```

The above Declaration Template creates a jinja2 variable namespace for better reusability. *vault* is invoked passing `ninja.secrets.latestService` to the *secret* parameter manually. When using *vault* as a jinja2 filter, this isn't necessary as the "piped" variable name is passed to the *secret* parameter automatically. In addition the *filter* parameter is set to an empty string to override any *filter* set within the Template Configuration. The empty string is not treated as a filter, therefore the whole secret is returned.

`secrets.latestService` now contains all the data we saw in the previous example and we create two more variables to store and later use the specific secret information we are interested in.

The resulting JSON looks like this:

```
{
  "latestService_privateKey": "-----BEGIN RSA PRIVATE KEY-----\
↪nMIHzAgEAAjEAvAI1w37cQcrflizN6Qa6GYVO26Sup5J0WWirYDS1aoxXCjQDcN4Q\
↪nf7cCQ82kSzcjAgMBAAECMFS5sjzdiKjlogjtPAYNkAQ8PSNifYrqxlpT4D5+TpWj\
↪nM1ODUjTVZBPQXuUIJYo6gQIZAOBcs33j5C6k7sisCVAvJTCTmdMx037zYQIZANaF\
↪nLSMLGaEhYz1da3OR6IHyM9Anx/h9AwIZAL4vlq+GeKzZfth4jMR90malF+Yg/IlG\
↪nwQIZAJKgRqDMRoFfK9DW2MoOsgiX/xhJCKLs9wIYPHBqLjhfB5Ycuk+WyxHj2uNQ\nNpf7zbsE\n-----END␣
↪RSA PRIVATE KEY-----",
  "latestService_version": 2
}
```

5.5.4 Specifying a secret version

A secret version can be specified either in the secrets configuration statement or explicitly via *vault*'s `version` parameter.

If we modify the *vault* call from the previous example like below, version 1 of the secret will be retrieved. The `version` parameter is optional and overrules any version configuration. It is valid regardless if *vault* is used as a filter or function.

```
{% set secrets.latestService = vault(secret=ninja.secrets.latestService,version=1) %}
```

```
{
  "latestService_privateKey": "-----BEGIN RSA PRIVATE KEY-----\
↪nMIGrAgEAAiEAyKNcibrMfVxuEwtifphGvEH1eP5Gjb3jbq8o0NfjjAMCAwEAAQIg\
↪nRp5RJN0NupX83FEmgr5gLqSYKeiIFCF4/vEcLrvVhOkCEQD5WC8HQPmQLFU//171\
↪n92OVAhEAzf5bxQk73WWXG6Wzcy7LNwIRANUDlQmpZIralOnbjJCtDBECECmOR6sf\
↪nKsGGLg64xdPVu88CEQDrfrKtfD5cSVENuhJ1LLie\n-----END RSA PRIVATE KEY-----",
  "latestService_version": 1
}
```

5.5.5 Using VaultClient

as3ninja.vault.VaultClient provides a way to connect to a specific Vault instance explicitly. *VaultClient* will return a *client* which can be passed to the *vault* filter/function.

Re-using the *myAPI* example with the following Declaration Template:

```
{
{% set vc = namespace() %}
{% set vc.client = VaultClient(addr="https://localhost:8201", verify=False) %}
"myAPI": {{
        ninja.secrets.myAPI | vault(client=vc.client) | jsonify
    }}
}
```

In this example the *client* is created on line 3 and stored in `vc.client`, which is then used in the *vault* filter as an argument to the *client* parameter. No explicit *token* was specified in this example. If no *token* is specified *VaultClient* will try to use the environment variable `VAULT_TOKEN` or an existing authenticated session based on Vault's cli (in that order).

An explicit *token* can be specified via the *VaultClient* `token` parameter.

Here is a fully parametrized example.

```yaml
# Template Configuration
dev:
  vault:
    token: s.iorspPP7f7EFpyudye6DB6Jn
    server_url: "https://dev-vault.example.net:8200"
    verify: false
secrets:
  myAPI:
    path: /secretOne/myAPI/sharedKey
    engine: kv1
    filter: data.secretKey
```

```
{# Declaration Template #}
{
{% set vc = namespace() %}
{% set vc.client = VaultClient(
                        addr=ninja.dev.vault.server_url,
                        token=ninja.dev.vault.token,
                        verify=ninja.dev.vault.verify
                        )
%}
"myAPI": {{
        ninja.secrets.myAPI | vault(client=vc.client) | jsonify
    }}
}
```

5.6 Using the AS3 Ninja vault integration directly with python

Although it is out of scope AS3 Ninja's vault integration can be used from python directly.

```
from as3ninja.vault import VaultClient, vault

my_vault_token = "s.tCU2wabNVCcySNncK2Mf6dwT"

s_myAPI = {
      'path':'/secretOne/myAPI/sharedKey',
      'engine':'kv1',
      'filter':'data.secretKey',
      }

s_latestService = {
      'path':'/otherService/privateKey',
      'mount_point':'SecEnginePath/myKV2',
      'filter':'data.privateKey',
    }

# using vault with an explicit Vault client

vc = VaultClient(addr="http://localhost:8200/",token=my_vault_token)

```

(continues on next page)

(continued from previous page)

```
vault(ctx={}, client=vc, secret=s_myAPI)
'AES 128 4d3642df883756b0d5746f32463f6005'

vault(ctx={} ,client=vc, secret=s_latestService)['data']['data']['privateKey']
'-----BEGIN RSA PRIVATE KEY-----\nMIHzAgEAAjEAvAI1w37cQcrflizN6Q...'


# using vault with a mocked jinja2 context

vault_settings = {'addr':'http://localhost:8200', 'token':my_vault_token}

jinja2_context = {'ninja':{'as3ninja':{ 'vault': vault_settings }}}

vault(ctx=jinja2_context, secret=s_myAPI)
'AES 128 4d3642df883756b0d5746f32463f6005'

vault(ctx=jinja2_context, secret=s_latestService)
'-----BEGIN RSA PRIVATE KEY-----\nMIHzAgEAAjEAvAI1w37cQcrflizN6Q...'

vault(ctx=jinja2_context, secret=s_latestService, version=1)
'-----BEGIN RSA PRIVATE KEY-----\nMIGrAgEAAiEAyKNcibrMfVxuEwtifp...'
```

CHAPTER

SIX

AS3NINJA

6.1 as3ninja package

6.1.1 Subpackages

as3ninja.jinja2 package

Submodules

as3ninja.jinja2.filterfunctions module

This module holds Jinja2 functions which also work as filters.

as3ninja.jinja2.filterfunctions.**b64decode**(*data*, *urlsafe=False*)

Accepts a string and returns the Base64 decoded representation of this string. *urlsafe=True* decodes urlsafe base64

Return type
Union[str, bytes]

as3ninja.jinja2.filterfunctions.**b64encode**(*data*, *urlsafe=False*)

Accepts a string and returns the Base64 encoded representation of this string. *urlsafe=True* encodes string as urlsafe base64

Return type
str

as3ninja.jinja2.filterfunctions.**env**(*env_var*, *default=None*)

Reads an environment variable and returns its value.

Use `default` to specify a default value in case the environment variable does not exist, Empty environment variables will return an empty string.

Examples:

```
{# using env as a filter #}
"HOME_DIR": "{{ 'HOME' | env }}"
```

```
{# using env as a function #}
{% set home_dir = env("HOME") %}
{% set temp_dir = env("TEMPDIR", default="/tmp") %}
```

Return type
str

as3ninja.jinja2.filterfunctions.**hashfunction**(*data*, *hash_algo*, *digest_format='hex'*)

Returns the digest of `data` for hash algorithm `hash_algo`. The digest is returned as hex by default, but can be returned as binary as well (`digest_format`)

Check the hashlib documentation of your python version for supported hash functions.

Parameters

- **hash_algo** (str) – hash algorithm
- **digest_format** (str) – Digest format to return. Either *hex* (default) or *binary*.

Return type
Union[str, bytes]

For the variable length shake digest, 256 bits are returned.

```
{% set whirlpool_hexdigest = hashfunction("fun with hashes", "whirlpool") %}
{# value of whirlpool_hexdigest is
↪"ce38f0a536e71b5b0758932c1d5f32d2ab6cc5bff9f02fb7c97a70291d45efa4516d4e3d99000956587c7c9f691f64b
↪"
 # note that whirlpool is not a guaranteed hash function, hence might not be␣
↪available on all platforms
 #}
```

as3ninja.jinja2.filterfunctions.**jsonify**(*data*, *quote=True*)

serializes data to JSON format.

`quote=False` avoids surrounding quotes, For example:

```
"key": "{{ ninja.somevariable | jsonify(quote=False) }}"
```

Instead of:

```
"key": {{ ninja.somevariable | jsonify }}
```

Return type
str

as3ninja.jinja2.filterfunctions.**md5sum**(*data*)

Returns the hash as a hexdigest of `data`. `data` is automatically converted to bytes, using backslashreplace for utf8 characters.

Return type
str

as3ninja.jinja2.filterfunctions.**readfile**(*ctx*, *filepath*, *missing_ok=False*)

Reads a file and returns its content as ASCII. Expects the file to be a ASCII (not utf8!) encoded file.

missing_ok=True prevents raising an exception when the file is missing and will return an empty string (default: missing_ok=False).

Return type
str

as3ninja.jinja2.filterfunctions.**sha1sum**(*data*)

Returns the hash as a hexdigest of `data`. `data` is automatically converted to bytes, using backslashreplace for utf8 characters.

Return type
`str`

as3ninja.jinja2.filterfunctions.**sha256sum**(*data*)

Returns the hash as a hexdigest of `data`. `data` is automatically converted to bytes, using backslashreplace for utf8 characters.

Return type
`str`

as3ninja.jinja2.filterfunctions.**sha512sum**(*data*)

Returns the hash as a hexdigest of `data`. `data` is automatically converted to bytes, using backslashreplace for utf8 characters.

Return type
`str`

as3ninja.jinja2.filterfunctions.**to_list**(*data*)

Converts `data` to a list. Unlike `list` it will not convert `str` to a list of each character but wrap the whole `str` in a list. Does not convert existing lists.

For example:

```
"virtualAddresses": {{ ninja.virtual_addresses | to_list | jsonify }},
```

If `ninja.virtual_addresses` is a list already it will not be nested, if it is a string, the string will be placed in a list.

Another example using the python REPL:

```
( to_list("foo bar") == ['foo bar'] ) == True  # strings

( to_list(["foo", "bar"]) == ['foo', 'bar'] ) == True  # existing lists

( to_list(245) == [245] ) == True  # integers
```

Return type
`list`

as3ninja.jinja2.filterfunctions.**uuid**(*_=None*)

Returns a UUID4

Return type
`str`

as3ninja.jinja2.filters module

This module holds Jinja2 filters for AS3 Ninja.

as3ninja.jinja2.filters.**ninjutsu**(*ctx*, *value*, ***kwargs*)

ninjutsu passes its input to jinja2 rendereing using the existing jinja2 environment and context. You can specify arbitary keyword arguments to pass additional variables to the renderer. This is important if you define variables within control structures, for example loops. These variables are not exported in the context and can therefore not be accessed within the jinja2 template passed to ninjutsu.

Example:

```
...
{% for thisone in allofthem %}
...
{# making thisone available to ninjutsu by passing
 # it as a keyword argument with the same name
 #}
{{ somesource.content.with.jinja2 | ninjutsu(thisone=thisone) }}
...
{% endfor %}
```

If `somesource.content.with.jinja2` uses `myvar` it would fail if we don't specify the keyword argument `myvar=myvar`, as it is not automatically exposed to the existing *jinja2 context*. An alternative are namespaces, just make sure the namespace is defined outside any of the control structures. Also note that the variable name will be an attribute of the namespace.

```
...
{% clipboard = namespace() %}
{% for thisone in allofthem %}
...
{% set clipboard.thisone = thisone %}
...
{# thisone is now available to ninjutsu as clipboard.thisone #}
{{ somesource.content.with.jinja2 | ninjutsu }}
...
{% endfor %}
```

Return type
str

as3ninja.jinja2.functions module

This module holds Jinja2 functions for AS3 Ninja.

class as3ninja.jinja2.functions.**iterfiles**(*pattern*, *missing_ok=False*)

Bases: object

iterates files, returns a tuple of all globbing matches and the file content as dict. Assumes the file content is either JSON or YAML.

iterfiles will ignore missing files if *missing_ok=True* is specified (default: False), otherwise will raise a *FileNotFoundError* exception.

as3ninja.jinja2.j2ninja module

J2Ninja collects jinja2 filters, functions an tests in a single class.

class as3ninja.jinja2.j2ninja.**J2Ninja**

Bases: object

J2Ninja provides decorator methods to register jinja2 filters, functions and tests, which are available as class attributes (dict).

filters: dict = {'b64decode': <function b64decode>, 'b64encode': <function b64encode>, 'env': <function env>, 'hashfunction': <function hashfunction>, 'jsonify': <function jsonify>, 'md5sum': <function md5sum>, 'ninjutsu': <function ninjutsu>, 'readfile': <function readfile>, 'sha1sum': <function sha1sum>, 'sha256sum': <function sha256sum>, 'sha512sum': <function sha512sum>, 'to_list': <function to_list>, 'uuid': <function uuid>, 'vault': <function vault>}

functions: dict = {'VaultClient': <class 'as3ninja.vault.VaultClient'>, 'b64decode': <function b64decode>, 'b64encode': <function b64encode>, 'env': <function env>, 'hashfunction': <function hashfunction>, 'iterfiles': <class 'as3ninja.jinja2.functions.iterfiles'>, 'jsonify': <function jsonify>, 'md5sum': <function md5sum>, 'readfile': <function readfile>, 'sha1sum': <function sha1sum>, 'sha256sum': <function sha256sum>, 'sha512sum': <function sha512sum>, 'to_list': <function to_list>, 'uuid': <function uuid>, 'vault': <function vault>}

classmethod registerfilter(*function*)

Decorator to register a jinja2 filter

classmethod registerfunction(*function*)

Decorator to register a jinja2 function

classmethod registertest(*function*)

Decorator to register a jinja2 test

tests: dict = {}

as3ninja.jinja2.tests module

This module holds Jinja2 tests for AS3 Ninja.

Module contents

Jinja2 filters, functions and tests module for AS3 Ninja.

as3ninja.schema package

Submodules

as3ninja.schema.as3schema module

AS3 Schema Class module. Represents the AS3 JSON Schema as a python class.

class as3ninja.schema.as3schema.**AS3Schema**(*version='latest'*)

Bases: object

Creates a AS3Schema instance of specified version. The *validate()* method provides AS3 Declaration validation based on the AS3 JSON Schema.

Parameters
version (str) – AS3 Schema version (Default value = "latest")

_SCHEMA_FILENAME_GLOB = '/as3-schema-*.json'**

_SCHEMA_LOCAL_FSPATH = PosixPath('/home/docs/.as3ninja/f5-appsvcs-extension/schema')

_SCHEMA_REF_URL_TEMPLATE = '/home/docs/.as3ninja/f5-appsvcs-extension/schema/{{version}}/as3-schema-{{version}}-*.json'

__schemalist_sort_helper(*value*)

Private Method: A sort helper.

Sorts based on the schema version (converted to int).

Return type
int

__version_sort_helper(*value*)

Private Method: A sort helper. converts value: str to int and removes "."

Parameters
value (str) – str: A version str (example: "3.8.1")

Return type
int

_build_ref_url(*version*)

Private Method: _build_ref_url builds the absolute filesystem url to the AS3 Schema file for specified version.

Parameters
version (str) – The AS3 Schema version

Return type
str

_check_version(*version*)

Private Method: _check_version checks if the specified version exists in available schemas. In case the specified schema version is not loaded, it will load the version. It converts "latest" to the actual version.

The checked version is returned as str.

Return type
str

_latest_version: str = ''

_load_schema(*version*, *force=False*)

Private Method: load schema file from disk for specified version. `force` parameter can be used to force load the schema file, even if it has been read already.

Return type
None

_ref_update(*schema*, *_ref_url*)

Private Method: _ref_update performs an in-place update of relative $ref (starting with #) into absolute references by prepending _ref_url.

See: https://github.com/Julian/jsonschema/issues/570

Return type
None

_schema_ref_update(*version*)

Private Method: _schema_ref_update returns the AS3 Schema for specified version with updated references.

Parameters
version (str) – The AS3 Schema version

Return type
dict

_schemas: dict = {}

_schemas_ref_updated: dict = {}

_sort_schemas()

Private Method: Sorts the schemas class attribute according to version

Return type
None

_update_versions(*versions*)

Private Method: Updates and sorts the versions class attribute

Return type
None

static _validate_schema_version_format(*version*)

Private Method: validates the format and minimum version.

Parameters
version (str) – str: AS3 Schema version

Return type
None

_validator(*version*)

Creates jsonschema.Draft7Validator for specified AS3 schema version. Will check schema is valid and raise a jsonschema SchemaError otherwise. Memoizes the Draft7Validator instance for faster re-use.

Return type
None

_validators: dict = {}

_versions: tuple = ()

property is_latest: bool

Property: returns bool(True) if this instance has the latest Schema version available. Returns False otherwise.

property latest_version: str

Property: returns the latest AS3 schema version as str.

property schema: dict

Property: returns the Schema of this AS3 Schema instance as dict.

property schema_asjson: str

Property: returns the Schema as JSON of this AS3 Schema instance as a python str.

property schemas: dict

Property: returns all known AS3 Schemas as dict.

updateschemas(*githubrepo='https://github.com/F5Networks/f5-appsvcs-extension'*, *repodir='/home/docs/.as3ninja/f5-appsvcs-extension'*)

Method: Fetches/Updates AS3 Schemas from the GitHub Repository.

Parameters

- **githubrepo** (str) – str: Git/Github repository to fetch AS3 Schemas from (Default value = constant NINJASETTINGS.SCHEMA_GITHUB_REPO)
- **repodir** (str) – str: Target directory to clone to (Default value = constant NINJASETTINGS.SCHEMA_BASE_PATH)

Return type
None

validate(*declaration*, *version=None*)

Method: Validates a declaration against the AS3 Schema. Raises a AS3ValidationError on failure.

Parameters

- **declaration** (Union[dict, str]) – Declaration to be validated against the AS3 Schema.
- **version** (Optional[str]) – Allows to validate the declaration against the specified version instead of this AS3 Schema instance version. If set to "auto", the version of the declaration is used.

Return type
None

property version: str

Property: returns the Schema version of this AS3 Schema instance as str.

property versions: tuple

Property: returns all versions available as a sorted tuple.

as3ninja.schema.formatcheckers module

AS3 Schema Format Checker for F5 specific formats.

class as3ninja.schema.formatcheckers.**AS3FormatChecker**(**args*, ***kwargs*)

Bases: FormatChecker

AS3FormatChecker subclasses jsonschema.FormatChecker to provide AS3 specific format checks.

static _is_type(*is_type*, *value*)

Helper function _is_type returns *True* when *is_type(value)* does not raise an exception, *False* otherwise

Parameters

- **is_type** (Any) – The type to check against
- **value** (Any) – Value to check

Return type
bool

static _regex_match(*regex*, *value*)

Helper function _regex_match matches a regular expression against the given value. Returns *True* when regex matches, *False* otherwise.

Parameters

- **regex** (str) – The regular expression, for example: `r'^[ -~]+$'`
- **value** (str) – Value to apply the regular expression to

Return type
bool

property as3_schema_format_checkers: dict

Returns dict of AS3 formats: f5ip, f5ipv4, f5ipv6, f5label, f5long-id, f5remark, f5pointer, f5base64 Currently missing formats used in AS3:

- date-time
- uri
- url

Module contents

AS3 Schema package.

6.1.2 Submodules

6.1.3 as3ninja.api module

AS3Ninja's REST API

class as3ninja.api.**AS3Declare**(***data*)

Bases: BaseModel

Model for an inline AS3 Declaration

_abc_impl = <_abc._abc_data object>

declaration_template: str

template_configuration: Union[List[dict], dict]

class as3ninja.api.**AS3DeclareGit**(***data*)

Bases: BaseModel

Model for an AS3 Declaration from a Git repository

_abc_impl = <_abc._abc_data object>

branch: Optional[str]

commit: Optional[str]

declaration_template: Optional[str]

depth: int

repository: str

template_configuration: Union[List[Union[dict, str]], dict, str, None]

class as3ninja.api.**AS3ValidationResult**(***data*)

Bases: BaseModel

AS3 declaration Schema validation result

_abc_impl = <_abc._abc_data object>

error: Optional[str]

valid: bool

class as3ninja.api.**Error**(***data*)

Bases: BaseModel

Generic Error Model

_abc_impl = <_abc._abc_data object>

code: int

message: str

class as3ninja.api.**LatestVersion**(***data*)

Bases: BaseModel

AS3 /schema/latest_version response

_abc_impl = <_abc._abc_data object>

latest_version: str

async as3ninja.api.**_schema_validate**(*declaration*, *version=Query(latest)*)

Validate declaration in POST payload against AS3 Schema of `version` (Default: latest)

async as3ninja.api.**default_redirect**()

redirect / to /api/docs

async as3ninja.api.**docs_redirect**()

redirect /docs to /api/docs

async as3ninja.api.**get_schema_latest_version**()

Returns latest known AS3 Schema version

async as3ninja.api.**get_schema_schema_version**(*version=Query(latest)*)

Returns AS3 Schema of `version`

async as3ninja.api.**get_schema_schemas**()

Returns all known AS3 Schemas

async as3ninja.api.**get_schema_versions**()

Returns array of version numbers for all known AS3 Schemas

async as3ninja.api.**openapi_redirect**()

redirect /openapi.json to /api/openapi.json

async as3ninja.api.**post_declaration_git_transform**(*as3d*)

Transforms an AS3 declaration template, see `AS3DeclareGit` for details on the expected input. Returns the AS3 Declaration.

async as3ninja.api.**post_declaration_transform**(*as3d*)

Transforms an AS3 declaration template, see `AS3Declare` for details on the expected input. Returns the AS3 Declaration.

async as3ninja.api.**redoc_redirect**()

redirect /redoc to /api/redoc

as3ninja.api.**startup**()

preload AS3Schema Class - assume Schemas are available

6.1.4 as3ninja.cli module

AS3 Ninja CLI module

as3ninja.cli.**_output_declaration**(*as3declaration*, *output_file*, *pretty*)

Function to output the transformed declaration

6.1.5 as3ninja.declaration module

The AS3Declaration module. Represents an AS3 Declaration as a python class.

class as3ninja.declaration.**AS3Declaration**(*template_configuration*, *declaration_template=None*, *jinja2_searchpath='.'*)

Bases: `object`

Creates an AS3Declaration instance representing the AS3 declaration.

The AS3 declaration is created using the given template configuration, which can be either a dict or list of dicts. If a list is provided, the member dicts will be merged using `_dict_deep_update()`.

Optionally a jinja2 declaration_template can be provided, otherwise it is read from the configuration. The template file reference is expected to be at *as3ninja.declaration_template* within the configuration. An explicitly specified declaration_template takes precedence over any included template.

Parameters

- **template_configuration** (Dict) – AS3 Template Configuration as dict or list
- **declaration_template** (Optional[str]) – Optional Declaration Template as str (Default value = ````)
- **jinja2_searchpath** (str) – The jinja2 search path for the FileSystemLoader. Important for jinja2 includes. (Default value = ".")

_jinja2_render()

Renders the declaration using jinja2. Raises relevant exceptions which need to be handled by the caller.

Return type
str

_transform()

Transforms the declaration_template using the template_configuration to an AS3 declaration. On error raises: :rtype: None

- AS3TemplateSyntaxError on jinja2 template syntax errors
- AS3UndefinedError for undefined variables in the declaration template
- AS3JSONDecodeError in case the rendered declaration is not valid JSON

property declaration_template: str

Property contains the declaration template loaded or provided during instantiation

dict()

Returns the AS3 Declaration.

Return type
dict

json()

Returns the AS3 Declaration as JSON.

Return type
str

6.1.6 as3ninja.exceptions module

All AS3 Ninja exceptions.

exception as3ninja.exceptions.**AS3JSONDecodeError**(*message=''*, *original_exception=None*)

Bases: ValueError

Raised when the produced JSON cannot be decoded

static _highlight_error(*doc*, *err_lineno*, *err_colno*)

Adds line numbers and highlights the error in the JSON document.

Parameters

- **doc** (str) – (invalid) JSON document
- **err_lineno** (int) – Erroneous line number
- **err_colno** (int) – exact error position on erroneous line

Return type
str

exception as3ninja.exceptions.**AS3SchemaError**(*message=''*, *original_exception=None*)

Bases: SchemaError

Raised when AS3 Schema is erroneous, eg. does not adhere to jsonschema standards.

exception as3ninja.exceptions.**AS3SchemaVersionError**

Bases: ValueError

AS3 Schema Version Error, version is likely invalid or unknown.

exception as3ninja.exceptions.**AS3TemplateConfigurationError**

Bases: ValueError

Raised when a problem occurs during building the Template Configuration.

exception as3ninja.exceptions.**AS3TemplateSyntaxError**(*message*, *declaration_template*, *original_exception=None*)

Bases: Exception

Raised to tell the user that there is a problem with the AS3 declaration template.

static **_highlight_error**(*doc*, *err_lineno*)

Adds line numbers and highlights the error in the Jinja2 template.

Parameters

- **doc** (str) – (invalid) Jinja2 template
- **err_lineno** (int) – Erroneous line number

Return type
str

exception as3ninja.exceptions.**AS3UndefinedError**(*message*, *original_exception=None*)

Bases: UndefinedError

Raised if a AS3 declaration template tries to operate on Undefined.

exception as3ninja.exceptions.**AS3ValidationError**(*message=''*, *original_exception=None*)

Bases: ValidationError

Validation of AS3 declaration against AS3 Schema produced an error.

exception as3ninja.exceptions.**GitgetException**

Bases: SubprocessError

Gitget Exception, subclassed SubprocessError Exception

6.1.7 as3ninja.gitget module

Gitget provides a minimal interface to 'git' to clone a repository with a specific branch, tag or commit id.

class as3ninja.gitget.**Gitget**(*repository*, *depth=None*, *branch=None*, *commit=None*, *repodir=None*, *force=False*)

Bases: object

Gitget context manager clones a git repository. Raises GitgetException on failure. Exports: *info* dict property with information about the cloned repository *repodir* str property with the filesystem path to the temporary directory Gitget creates a shall clone of the specified repository using the specified and optional depth. A branch can be selected, if not specified the git server default branch is used (usually master).

A specific commit id in long format can be selected, depth can be used to reach back into the past in case the commit id isn't available through a shallow clone.

_checkout_commit()

Private Method: checks out specific commit id

Note: short ID (example: 2b54d17) is not allowed, must be the long commit id Note: The referenced commit id must be in the cloned repository or within a depth of 20

_clone()

Private Method: clones git repository

static _datetime_format(*epoch*)

Private Method: returns a human readable UTC format (%Y-%m-%dT%H:%M:%SZ) of the unix epoch

Parameters
epoch (Union[int, str]) – Unix epoch

Return type
str

_get_gitlog()

Private Method: parses the git log to a dict

Return type
None

_gitcmd = ('git', '-c', 'http.sslVerify=True', '-c', 'http.proxy=')

_run_command(*cmd*)

Private Method: runs a shell command and handles/raises exceptions based on the command return code

Parameters
cmd (tuple) – list of command + arguments

Return type
str

static _sh_quote(*arg*)

Private Method: returns a shell escaped version of arg, where arg can by any type convertible to str. uses shlex.quote

Parameters
arg – Argument to pass to *shlex.quote*

Return type
str

property info: dict

Property: returns dict with git log information

property repodir: str

Property: returns the (temporary) directory of the repository

rmrepodir()

Method: Removes the repodir.

This method is useful if repodir has been specified in __init__().

Return type
None

6.1.8 as3ninja.settings module

AS3 Ninja global configuration parameters.

class as3ninja.settings.**NinjaSettings**(*_env_file='<object object>'*, *_env_file_encoding=None*, *_env_nested_delimiter=None*, *_secrets_dir=None*, ***values*)

Bases: `BaseSettings`

AS3 Ninja Settings class.

Holds the default configuration for AS3 Ninja.

Reads from $CWD/as3ninja.settings.json if it exists, otherwise from *$HOME/.as3ninja/as3ninja.settings.json*. If none of the configuration files exist, it creates *$HOME/.as3ninja/as3ninja.settings.json* and writes the current configuration (default + settings overwritten by ENV vars).

Any setting can be overwritten using environment variables. The ENV variable has a prefix of *AS3N_* + name of the setting. The environment variables take precedence over any setting in the configuration file.

class Config

Bases: `object`

Configuration for NinjaSettings BaseSettings class

case_sensitive = True

env_prefix = 'AS3N_'

extra = 'forbid'

GITGET_PROXY: str

GITGET_SSL_VERIFY: bool

GITGET_TIMEOUT: int

SCHEMA_BASE_PATH: str

SCHEMA_GITHUB_REPO: str

VAULT_SSL_VERIFY: bool

_abc_impl = <_abc._abc_data object>

class as3ninja.settings.**NinjaSettingsLoader**

Bases: `object`

The NinjaSettingsLoader class is an utility class which will return a callable instance which in fact returns an instance of NinjaSettings. NinjaSettingsLoader contains utility functions to detect the config file and the SCHEMA_BASE_PATH, it will also create the config file if it does not yet exist.

AS3NINJA_CONFIGFILE_NAME = 'as3ninja.settings.json'

AS3_SCHEMA_DIRECTORY = '/f5-appsvcs-extension'

RUNTIME_CONFIG = ['SCHEMA_BASE_PATH']

classmethod _detect_config_file()

Detect if/where the AS3 Ninja config file *(as3ninja.settings.json)* is located.

First checks for existence of *as3ninja.settings.json* and uses this file if found. Alternatively *Path.home()/.as3ninja/as3ninja.settings.json* is used and created if it doesn't exist.

Return type
Optional[str]

classmethod _detect_schema_base_path()

Detect where AS3 JSON Schema files are stored.

First checks for existence of *Path.cwd()/f5-appsvcs-extension* and uses this path if found. Alternatively *Path.home()/.as3ninja/f5-appsvcs-extension* is used and created if it doesn't exist.

Return type
str

_save_config()

Saves the current settings as JSON to the configuration file in ~/.as3ninja/. It removes any RUNTIME_CONFIG keys before saving.

Return type
None

_settings: *NinjaSettings* = NinjaSettings(GITGET_TIMEOUT=120, GITGET_SSL_VERIFY=True, GITGET_PROXY='', SCHEMA_BASE_PATH='', SCHEMA_GITHUB_REPO='https://github.com/F5Networks/f5-appsvcs-extension', VAULT_SSL_VERIFY=True)

6.1.9 as3ninja.templateconfiguration module

The AS3TemplateConfiguration module allows to compose AS3 Template Configurations from YAML, JSON or dict(s).

class as3ninja.templateconfiguration.**AS3TemplateConfiguration**(*template_configuration=None*, *base_path=''*, *overlay=None*)

Bases: *DictLike*

The AS3TemplateConfiguration module. Allows to build an AS3 Template Configuration from YAML, JSON or dict. Creates a AS3TemplateConfiguration instance for use with AS3Declaration.

The Template Configuration can be created from one or more files or *dicts*. Globbing based on pathlib Path glob is supported to load multiple files. De-serialization for files is automatically performed, YAML and JSON is supported. If a file is included multiple times, it is only loaded once on first occurrence. AS3TemplateConfigurationError exception is raised when a file is not found or not readable.

Files can be included using the as3ninja.include Union[str, List[str]] namespace in every specified configuration file. Files included through this namespace will not be checked for as3ninja.include and therefore cannot include further files.

The as3ninja.include namespace is updated with entries of all as3ninja.include entries, globbing will be expanded. This helps during troubleshooting.

If a list of inputs is provided, the input will be merged using *_dict_deep_update()*.

If template_configuration is None, AS3TemplateConfiguration will look for the first default configuration file it finds in the current working directory (files are in order: *ninja.json*, *ninja.yaml*, *ninja.yml*).

Parameters

- **template_configuration** (Union[List[Union[*dict*, str]], *dict*, str, None]) – Template Configuration (Optional)
- **base_path** (Optional[str]) – Base path for any configuration file includes. (Optional)

Example usage:

```
from as3ninja.templateconfiguration import AS3TemplateConfiguration

as3tc = AS3TemplateConfiguration([
            {"inlineConfig": True},
            "./config.yaml",
            "./config.json",
            "./includes/*.yaml"
    ])

as3tc.dict()
as3tc.json()
as3tc_dict = dict(as3tc)
```

class TemplateConfigurationValidator(***data*)

Bases: BaseModel

Data Model validation and de-serialization for as3ninja.include namespace.

_abc_impl = <_abc._abc_data object>

template_configuration: Union[List[Union[dict, str]], dict, str]

_deserialize_files()

De-serialize configuration files in self._template_configurations

_deserialize_includes(*includes*, *register=True*)

Iterates and expands over the list of includes and yields the deseriealized data.

Parameters

- **includes** (List[str]) – List of include files
- **register** (bool) – Register include file to avoid double inclusion (Default: True)

Return type

Generator

_dict_deep_update(*dict_to_update*, *update*)

Similar to dict.update() but with full depth.

Parameters

- **dict_to_update** (Dict) – dict to update (will be mutated)
- **update** (Dict) – dict: dict to use for updating dict_to_update

Return type

Dict

Example:

```
dict.update:
{ 'a': {'b':1, 'c':2} }.update({'a': {'d':3} })
-> { 'a': {'d':3} }

_dict_deep_update:
{ 'a': {'b':1, 'c':2} } with _dict_deep_update({'a': {'d':3} })
-> { 'a': {'b':1, 'c':2, 'd':3} }
```

_import_includes(*defferred=False*)

Iterates the list of Template Configurations and imports all includes in order.

Parameters
defferred (bool) – Include defferred includes instead of user specified as3ninja.include

_merge_configuration()

Merges _template_configurations list of dicts to a single dict

_ninja_default_configfile()

Identify first config file which exists:ninja.json, ninja.yaml or ninja.yml. Raise AS3TemplateConfigurationError on error.

Return type
str

_path_glob(*pattern*)

Path(self._base_path).glob(pattern) with support for an absolute pattern.

Return type
Generator[Path, None, None]

_tidy_as3ninja_namespace()

Tidy as3ninja. namespace in the configuration. Removes:

- __deserialize_file
- removes entire as3ninja namespace if empty

_update_configuration_includes()

Updates as3ninja.include with the full list of included files and removes __deserialize_file

dict()

Returns the merged Template Configuration

Return type
dict

json()

Returns the merged Template Configuration as JSON

Return type
str

6.1.10 as3ninja.types module

AS3 Ninja types.

class as3ninja.types.**BaseF5IP**

Bases: str

F5IP base class. Accepts IPv4 and IPv6 IP addresses in F5 notation.

static _get_addr(*ipaddr*)

Return type
str

static _get_mask(*ipaddr*)

Return type
str

static _get_rdid(*ipaddr*)

Return type
str

classmethod _validate_ip(*value*)

Return type
None

classmethod _validate_ipany(*value*)

Return type
None

static _validate_ipv4(*value*)

Return type
None

static _validate_ipv6(*value*)

Return type
None

static _validate_rdid(*rdid*)

Return type
None

classmethod validate(*value*)

Validate method is automatically called pydantic.

class as3ninja.types.**BaseF5IPv4**

Bases: *BaseF5IP*

F5IPv4 base class. Accepts IPv4 addresses in F5 notation.

classmethod _validate_ip(*value*)

Return type
None

class as3ninja.types.**BaseF5IPv6**

Bases: *BaseF5IP*

F5IPv6 base class. Accepts IPv6 addresses in F5 notation.

classmethod _validate_ip(*value*)

Return type
None

class as3ninja.types.**F5IP**(*f5ip*)

Bases: BaseModel

Accepts and validates IPv6 and IPv4 addresses in F5 notation.

_abc_impl = <_abc._abc_data object>

addr: str

f5ip: *BaseF5IP*

mask: Optional[Any]

rdid: Optional[Any]

class as3ninja.types.**F5IPv4**(*f5ip*)

Bases: *F5IP*

Accepts and validates IPv4 addresses in F5 notation.

_abc_impl = <_abc._abc_data object>

f5ip: *BaseF5IPv4*

class as3ninja.types.**F5IPv6**(*f5ip*)

Bases: *F5IP*

Accepts and validates IPv6 addresses in F5 notation.

_abc_impl = <_abc._abc_data object>

f5ip: *BaseF5IPv6*

6.1.11 as3ninja.utils module

Various utils and helpes used by AS3 Ninja

class as3ninja.utils.**DictLike**

Bases: object

Makes objects *feel* like a dict.

Implements required dunder and common methods used to access dict data.

_dict: dict = {}

get(*key*, *default=None*)

Return type
Any

items()

Return type
ItemsView[Any, Any]

keys()

Return type
KeysView[Any]

values()

Return type
ValuesView[Any]

exception as3ninja.utils.**PathAccessError**(*exc*, *seg*, *path*)

Bases: KeyError, IndexError, TypeError

An amalgamation of KeyError, IndexError, and TypeError, representing what can occur when looking up a path in a nested object.

class as3ninja.utils.**YamlConstructor**

Bases: object

Organizes functions to implement a custom PyYAML constructor

INCLUDE_TAG = '!include'

classmethod **_include_constructor**(_, *node*)

The PyYAML constructor for the INCLUDE_TAG (!include). This method should not be called directly, it is passed to PyYAML as a constructor function.

Parameters
node – The yaml node to be inspected

Return type
Union[List, Dict]

static **_path_glob**(*value*)

A Path().glob() helper function, checks if *value* actually contains a globbing pattern and either returns *value* or the result of the globbing.

Parameters
value (str) – String to check for globbing pattern and, if pattern found, to feed to Path().glob()

Return type
List[str]

classmethod **add_constructors**(*yaml_module*)

Adds constructors to PyYAML module.

Parameters
yaml_module – Name of loaded PyYAML module

as3ninja.utils.**deserialize**(*datasource*)

deserialize de-serializes JSON or YAML from a file to a python dict.

A ValueError exception is raised if JSON and YAML de-serialization fails. A FileNotFoundError is raised when an included file is not found.

Parameters
datasource (str) – The filename (including path) to deserialize

Return type
Dict

as3ninja.utils.**dict_filter**(*dict_to_filter*, *filter=None*)

Filters a dict based on the provided filter.

dict_filter will walk the dict keys based on the filter and will return the value of the last key. If filter is empty, dict_to_filter will be returned.

Example: assert dict_filter({ 'a': { 'b': [1,2,3] } }, filter="a.b") == [1,2,3]

Parameters

- **dict_to_filter** (dict) – Python dict to filter

- **filter** (Union[tuple, str, None]) – Filter to apply to the dict. Filter can be a str (will be split on .) or a tuple.

Return type
Any

as3ninja.utils.**escape_split**(*string_to_split*, *seperator='.'*)

Splits a string based on the provided seperator.

escape_split supports escaping the seperator by prepending a backslash.

Parameters

- **string_to_split** (str) – String to split
- **seperator** (str) – Seperator to use for splitting (Default: ".")

Return type
tuple

as3ninja.utils.**failOnException**(*wrapped_function*)

sys.exit(1) on any exception

6.1.12 as3ninja.vault module

HashiCorp Vault integration

class as3ninja.vault.**VaultClient**(*addr*, *token=None*, *verify=True*)

Bases: object

Vault Client object, returns a hvac.v1.Client object.

Parameters

- **addr** (str) – Vault Address (url, eg. *https://myvault:8200/*)
- **token** (Optional[str]) – Vault Token to use for authentication
- **verify** (Union[str, bool]) – If *True* Verify TLS Certificate of Vault (Default: *True*)

Client()

Returns hvac.client callable based on VaultClient() initialization parameters.

Return type
Client

_defaultClient = None

classmethod defaultClient(*ctx*)

Returns a hvac.v1.Client based on system/environment settings.

This is method is not intended to be used directly.

First checks for existing authentication based on *vault* cli. If authenticated no further action is performed.

Then check the Jinja2 Context for the namespace ninja.as3ninja.vault and use addr, token and ssl_verify to establish a Vault connection. For any of the above variables that doesn't exist the respective environment variable will be used as a fallback: addr = VAULT_ADDR token = *VAULT_TOKEN* ssl_verify = VAULT_SKIP_VERIFY

If VAULT_SKIP_VERIFY does not exist VAULT_SSL_VERIFY from the AS3 Ninja configuration file (*as3ninja.settings.json*) is used.

Parameters
ctx (Context) – Context: Jinja2 Context

Return type
Client

class as3ninja.vault.**VaultSecret**(**args*, *path: str*, *mount_point: str*, *engine: Union[str, VaultSecretsEngines] = VaultSecretsEngines.kv2*, *filter: Optional[str] = None*, *version: int = 0*)

Bases: BaseModel

Vault Secret configuration BaseModel.

Parameters

- **path** – The secret path. If *mount_point* is not specified the first path element is assumed to be the *mount_point*.
- **mount_point** – The secrets engine path. Optional.
- **engine** – The secrets engine. Optional.
- **filter** – Optional Filter to select specific data from the secret, e.g. "data.privateKey". Filter automatically prepends "data." for kv2 to replicate the same behaviour for kv1 and kv2.
- **version** – The version of the secret. Only relevant for KV2 Secrets Engine. Optional. Default: 0 (latest secret version)

_abc_impl = <_abc._abc_data object>

static _split_mount_point_path(*path*)

Splits mount_point from path. The first path element is treated as the mount_point.

Parameters
path (str) – path parameter

Return type
tuple

engine: Union[str, *VaultSecretsEngines*]

filter: Optional[str]

mount_point: str

path: str

classmethod validate_engine(*value*)

Validate engine against VaultSecretsEngines

classmethod validate_pathlike(*value*)

Basic secrets path validation using pathlib.Path. This should work for most vault secrets paths.

classmethod validate_version(*value*)

Validate version

version: int

class as3ninja.vault.**VaultSecretsEngines**(*value*)

Bases: Enum

Supported Vault Secret Engines

default = 'kv2'

kv = 'kv1'

kv1 = 'kv1'

kv2 = 'kv2'

Aliases

as3ninja.vault.**vault**(*ctx*, *secret*, *client=None*, *filter=None*, *version=None*)

Vault filter to retrieve a secret. The secret is returned as a dict.

Parameters

- **ctx** (Context) – Context: Jinja2 Context (automatically provided by jinja2)
- **secret** (Dict) – secret configuration statement, automatically passed by "piping" to the vault filter
- **client** (Optional[*VaultClient*]) – Optional Vault client
- **filter** (Optional[str]) – Optional Filter to select specific data from the secret, e.g. "data.privateKey". Filter automatically prepends "data." for kv2 to replicate the same behaviour for kv1 and kv2.
- **version** (Optional[int]) – Optional secret version (overrides version provided by secret configuration statement)

Return type

Dict

6.1.13 Module contents

Top-level package for AS3 Ninja.

CHAPTER

SEVEN

AS3 NINJA SETTINGS

`as3ninja.settings.json` holds the default configuration settings for AS3 Ninja.

AS3 Ninja looks for `as3ninja.settings.json` in the following files and uses the first it finds:

1. `$CWD/as3ninja.settings.json`
2. `$HOME/.as3ninja/as3ninja.settings.json`

If none of the configuration files exist, it creates `$HOME/.as3ninja/as3ninja.settings.json` and writes the current configuration (default + settings overwritten by ENV vars).

Any setting can be overwritten using environment variables. The ENV variable has to be prefixed by `AS3N_`. The environment variables take precedence over any setting in the configuration file.

For specific settings and its meaning check the source of *as3ninja.settings.NinjaSettings*.

CHAPTER

EIGHT

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

8.1 Types of Contributions

8.1.1 Report Bugs

Report bugs at https://github.com/simonkowallik/as3ninja/issues.

If you are reporting a bug, please include:

- Your operating system name and version, python version and as3ninja version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

8.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

8.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

8.1.4 Write Documentation

AS3 Ninja could always use more documentation, whether as part of the official AS3 Ninja docs, in docstrings, or even on the web in blog posts, articles, and such.

8.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/simonkowallik/as3ninja/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

8.2 Get Started!

Ready to contribute? Here's how to set up *as3ninja* for local development.

1. Fork the *as3ninja* repo on GitHub.
2. Clone your fork locally:

```
$ git clone --branch edge git@github.com:your_name_here/as3ninja.git
```

3. Install your local copy into a virtualenv. Assuming you use poetry:

```
$ cd as3ninja/
$ poetry shell
$ poetry install
```

4. Create a branch for local development:

```
$ git checkout -b (bugfix|feature|enhancement)/name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes comply to code formatting and pass the tests:

```
$ make lint
$ make code-format
$ make test
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit
$ git push origin (bugfix|feature|enhancement)/name-of-your-bugfix-or-feature
```

7. Submit a pull request through GitHub.

8.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring and update the relevant documentation.
3. The pull request should work for Python 3.8 and up. Check https://github.com/simonkowallik/as3ninja/actions and make sure that the tests pass for all supported Python versions.

CHAPTER

NINE

SUPPORT

AS3 Ninja is an open source project started by a single individual and released under the ISC license.

By using AS3 Ninja you confirm to understand, agree and adhere to the license.

Warning: There is no commercial/technical support nor any other form of support for AS3 Ninja. It comes without SLAs or any form of warranty.

You are welcome to *contribute*, for example by opening a GitHub issue to report a problem.

CHAPTER

TEN

CREDITS

10.1 Author

- Simon Kowallik <github@simonkowallik.com>

10.2 Contributors

None yet. Why not be the first?

CHAPTER

ELEVEN

HISTORY

For a detailed history please head to GitHub releases.

CHAPTER

TWELVE

OPEN DOC TASKS

This is the list of outstanding documentation tasks.

Todo: more cli examples

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/as3ninja/checkouts/edge/docs/usage.rst, line 79.)

Todo: Postman collection for API calls

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/as3ninja/checkouts/edge/docs/usage.rst, line 121.)

Todo: Update usage as a module

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/as3ninja/checkouts/edge/docs/usage.rst, line 182.)

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

Y