ETOS Documentation

Release 1.0.0

Tobias Persson

Mar 24, 2021

Contents:

ETOS (Eiffel Test Orchestration System) is a new test orchestration system which takes away control of how to run and what to run from the system itself and places it into the hands of the relevant engineers.

The idea of having a system dictate what and how to run is finished. Let's bring back control to the testers.

With ETOS we define how to run tests using recipes and we define what to run with collections of recipes.

Table 1: ETOS Roles

Test Engineer	*Test Automation Engineer*	*System Engineer*
Create collections	Create recipes	Deploy ETOS
Analyze results	Create tests	Infrastructure
What to run?	How to run?	Where to run?

This means that only people who knows how and what to run decide these factors. ETOS will only receive the collection of recipes and execute it accordingly. You can also mix test suites. For instance, let's say you want to run a "go" unittest and a "python" function test in the same suite, that's easy to do; just add them to your collection.

This is the strength of ETOS. Relying on the humans to decide what to run, how to run and where to run.

ETOS is a collection of multiple services working together. This repository is a facilitator of versioning, Helm charts and documentation. The services are located in these repositories.

- *ETOS Client*
- *ETOS API*
- *ETOS Suite Starter*
- *ETOS Suite Runner*
- *ETOS Test Runner*
- *ETOS Environment Provider*
- *ETOS Library*
- *ETOS Test Runner Containers*

CHAPTER 1

Features

- Generic test suite execution based solely on JSON.
- Mix and match test suites, regardless of programming language.
- Separation of concerns between testers, test automation engineers and system engineers.
- Eiffel protocol implementation.

CHAPTER 2

Installation

2.1 Requirements

In order to install ETOS, you need to meet the following requirements.

- An up and running kubernetes cluster (https://kubernetes.io/)
- Helm version 3.x installed (https://helm.sh/)

2.2 Installation Steps

1. First we need to add the Helm repository where the ETOS Helm charts are stored

```
helm repo add Eiffel registry.nordix.org/eiffel
```

2. Then simply install ETOS using Helm

```
helm install <name of the ETOS deployments> eiffel/etos --namespace <your kubernetes␣
↪namespace>
```

2.3 Deployment Configuration

Following the installation step will give you a default configured ETOS deployment. Chances are that the default deployment configuration of ETOS will not work for your Infrastructure. To tailor the deployment to your specific infrastructure you need to create a configuration file and tell Helm to use that file when installing ETOS.

Here is an example of a standard ETOS configuration file that should get most configurations up and running.

```yaml
global:
  # This is the URL to the Eiffel Graphql API
  graphqlServerUrl: http://eiffel-graphql-api.my.cluster-url.com
  # This is the URL where the deployed ETOS Environment Provider will be available
  environmentProviderUrl: http://environment-provider.my.cluster-url.com
  # This is the URL where the deployed ETOS API will be available
  etosApiUrl: http://etos-api.my.cluster-url.com

suite-starter:
  rabbitMQ:
    # this is the message queue where suite starter listens for Eiffel
    queue_name: suite_starter.queue

# This is the configuration that should match your rabbitMQ deployment
# ETOS needs a rabbitMQ service to be able to subscribe and publish Eiffel events
rabbitmqHost: dev-rabbitmq.myhost.com
rabbitmqExchange: my.eiffel.exchange
rabbitmqPort: "5671"
rabbitmqVhost: myvhost
rabbitMQ:
  username: rabbit_user
  password: rabbit_password

# This is the configuration that should match your redis deployment
# ETOS uses redis for internal communication and data storage
databaseHost: redis.redis.svc.cluster.local
databasePort: "26379"
redis:
  password: my_redis_password
```

CHAPTER 3

Contribute

Please write issues in the relevant repositories for where you found the issue.
If you do not know which repository to write the issue for, feel free to write it here and it will be moved.
Documentation issues are reported here.

- Issue Tracker: https://github.com/eiffel-community/etos/issues
- Source Code: https://github.com/eiffel-community/etos

CHAPTER 4

Support

If you are having issues, please let us know. There is a mailing list at: etos-maintainers@googlegroups.com or just write an Issue.

4.1 Getting started

4.1.1 Step by step

Checklist

Setting up ETOS is a daunting task when you are doing it the first time around. But it is also easy to miss a few steps that are required for it to work.

For this reason we have created a checklist in order to keep track of what needs to be done so that no steps are forgotten.

- Set up a Kubernetes cluster (ETOS runs in Kubernetes)
- Set up an *Execution Space* (Execute test runner containers)
- Set up a *Log Area* (Storing logs after execution)
- Deploy MongoDB (Event storage DB)
- Deploy Redis Sentinel (ETOS internal communications)
- Deploy RabbitMQ (must be accessible outside of kubernetes as well)
- Deploy Eiffel GraphQL API
  - Deploy an Eiffel event storage tool. Example
- Deploy ETOS helm chart: *Installation*
- Configure *ETOS Environment Provider*
  - *IUT Provider*
  - *Execution Space Provider*

    - *Log Area Provider*
- Create an *ETOS Test Collection*
    - Upload the *ETOS Test Collection* to an area with no required authorization.
- Generate an ArtC
- Generate an ArtP
- Verify that the ArtC and ArtP events exist in the Eiffel GraphQL API.
- Install *Installing ETOS Client on a workstation*
- Set *Environment variables* for ETOS Client
- Test ETOS deployment using *ETOS Client*

Installing ETOS Client on a workstation

This section will guide you through the process of setting up *ETOS Client*.

ETOS Client is the default tool for executing the test suites with. We always recommend using the client.

Requirements

- Python 3.6 (or higher)

Installation

ETOS Client can be found on PyPi and is installable with pip.

```
pip install etos_client
```

CLI Usage

```
etos_client --help
```

More on usage can be found *here*

Setting up a Jenkins delegation job

This page describes how to set up delegation jobs for ETOS. A delegation job's function is described *here*

Note that a delegation job can be created just the way you want to (as long as it follows the instructions from the execution space), this is just a sample of how you could implement it.

Prerequisites

- Jenkins
- Jenkins Pipelines

Example setup

1. Create a pipeline job.
2. Recommended to set cleanup policy for the job.
3. Add multi-line string parameter named 'docker'.
4. Configure *Execution Space* to send the 'docker' parameter to Jenkins.
5. Add script to delegation

```
node() {
    stage('ETOS') {
        def jsonslurper = new groovy.json.JsonSlurper()
        def json = params.docker
        def dockerJSON = jsonslurper.parseText(json)

        def environmentJSON = dockerJSON["environment"]
        def parametersJSON = dockerJSON["parameters"]
        def dockerName
        if (parametersJSON.containsKey("--name")) {
            dockerName = parametersJSON["--name"]
        } else {
            dockerName = UUID.randomUUID().toString()
            parametersJSON["--name"] = dockerName
        }
        env.DOCKERNAME = dockerName
        def environment = ""
        def parameters = ""
        environmentJSON.each{entry -> environment += "-e $entry.key=$entry.value "}
        parametersJSON.each{entry -> parameters += "$entry.key $entry.value "}
        def image = dockerJSON["image"]
        def command = "docker run --rm " + environment + parameters + image + " &"
        /*
          Write a bash file which will trap interrupts so that the docker container
          is properly removed when canceling a build.
        */
        writeFile file: 'run.sh', text: (
            '_terminate() {\n'
            + '    echo "Stopping container"\n'
            + "    docker stop $dockerName\n"
            + '}\n'
            + 'trap _terminate SIGTERM\n'
            + "$command \n"
            + 'child=$!\n'
            + 'wait "$child"\n'
        )
        sh "docker pull $image || true"
        sh """
        bash run.sh
        docker rm $dockerName || true
        """
        sh "rm run.sh"
    }
}
```

Example execution space provider

Checkout any number of static execution spaces. More information about execution space providers *here*

```
{
  "execution_space": {
        "id": "jenkins",
        "list": {
            "possible": {
                "$expand": {
                    "value": {
                        "request": {
                            "url": "https://jenkins/job/DELEGATION/build",
                            "method": "POST",
                            "headers": {
                                "Accept": "application/json"
                            },
                            "data": {
                                "json": {
                                    "$json_dumps": {
                                        "parameter": [
                                            { "name": "docker", "value": {
                                                "$json_dumps": "$execution_space_
↪instructions"
                                            }
                                        }
                                    ]
                                }
                            }
                        }
                    },
                    "to": "$amount"
                }
            },
            "available": "$this.possible"
        }
    }
}
```

4.1.2 ETOS Test Collection

This page describes the ETOS test collection.

A test collection is how you trigger your tests. The test collection is a collection of recipes that are defined by the test automation engineers. These recipes defines how ETOS will execute the tests.

This test collection is JSON file with several elements and can be quite tricky to create manually.

Structure

Top-level properties

```
[
   {
      "name": "Name of you suite",
      "priority": 1,
      "recipes": []
   }
]
```

Table 1: Top level properties

Property	Type	Description
name	String	Test suite name. Will be referenced in the confidence level.
priority	Integer	The execution priority. Unused at present and should be set to 1.
recipes	List	The collection of test execution recipes. Each recipe is a test case.

Recipes

```
[
   {
      "recipes": [
         {
            "constraints": [],
            "id": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
            "testCase": {
               "id": "test_case_name",
               "tracker": "Name of test case tracker",
               "url": "http://testcasetracker.com/test_case_name"
            }
         }
      ]
   }
]
```

Table 2: Recipe properties

Property	Type	Description
constraints	List	List of constraints describing how to execute this test case.
id	String	Unique ID for this test case. Can be used to re-trigger failing test cases.
testCase	Dictionary	Test case metadata.
testCase/id	String	Name of this test case.
testCase/tracker	String	The name of the tracker where this test case can be found.
testCase/url	String	URL to the tracker for this test case.

Constraints

```
[
   {
      "recipes": [
         {
            "constraints": [
               {
                  "key": "ENVIRONMENT",
                  "value": {
                     "MY_ENVIRONMENT": "my_environment_value"
                  }
               },
               {
                  "key": "COMMAND",
                  "value": "tox"
               },
               {
                  "key": "PARAMETERS",
                  "value": {
                     "-e": "py3"
                  }
               },
               {
                  "key": "TEST_RUNNER",
                  "value": "eiffel-community/etos-python-test-runner"
               },
               {
                  "key": "EXECUTE",
                  "value": [
                     "echo 'hello world'"
                  ]
               },
               {
                  "key": "CHECKOUT",
                  "value": [
                     "git clone https://github.com/eiffel-community/etos-client"
                  ]
               }
            ]
         }
      ]
   }
]
```

Table 3: Constraint properties

Property	Value	Description
ENVIRONMENT	Dictionary	The environment key defines which environment variables that are needed for this test case execution.
COMMAND	String	The command key defines which command to execute in order to run the specified test case.
PARAMETERS	Dictionary	The parameters key defines which parameters you want to supply to the command that is executing the tests.
TEST_RUNNER	String	Which test runner you need to execute the test cases in: See *ETOS Test Runner Containers* for more information.
EXECUTE	List	The execute key defines a set of shell commands to execute before this test case.
CHECKOUT	List	The checkout key defines how to checkout your test cases. The checkout values are executed in bash. This command is only executed once if it has already been executed.

4.2 Concepts

4.2.1 Execution Space

An execution space is a system where docker containers can run.

In the ETOS world the execution space is responsible for receiving a run order where it will start a docker container using the instructions from the *ETOS Environment Provider*.

This execution space will then execute the docker container until completion and then shut down.

An example of an execution space is Jenkins

4.2.2 IUT

IUT (or Item under test) is the current software, hardware or system that is currently under test.

This can be the website you've just created, a docker container or even a full software/hardware system.

4.2.3 Log Area

A log area is a system where ETOS stores logs after completing test execution.

This log area must have an API where it is possible to upload logs, test execution workspace, test artifacts and test suites.

An example of a log area is JFrog Artifactory

4.3 Versioning rules

ETOS uses semantic versioning in the form of "MAJOR.MINOR.PATCH"

- MAJOR: A large change, sometimes breaking changes.
- MINOR: A smaller change, never a breaking change.
- PATCH: A bugfix required in production.

Whenever a new ETOS MAJOR or MINOR deployment is made, we _ALWAYS_ deploy every tool and all versions will be the same in each service (at the time of release).

PATCH updates in productions are allowed to be deployed on individual services but the PATCH number must also be incremented in the 'ETOS' repository. The ETOS repository holds the main helm chart and the main version number. A PATCH is ALWAYS a small bugfix to a specific component that is REQUIRED for production to work.

Examples

If we release version 1.0.0 of ETOS:

- etos: 1.0.0
- etos-client: 1.0.0
- etos-api: 1.0.0
- etos-suite-starter: 1.0.0
- etos-suite-runner: 1.0.0
- etos-test-runner: 1.0.0
- etos-environment-provider: 1.0.0
- etos-library: 1.0.0
- etos-test-runner-containers: 1.0.0

Then we notice a bug in the "etos-client" which makes it impossible to run ETOS. We fix this bug and increase the PATCH number and release it:

- etos: 1.0.1
- etos-client: 1.0.1
- etos-api: 1.0.0
- etos-suite-starter: 1.0.0
- etos-suite-runner: 1.0.0
- etos-test-runner: 1.0.0
- etos-environment-provider: 1.0.0
- etos-library: 1.0.0
- etos-test-runner-containers: 1.0.0

Now we notice a bug in the "etos-environment-provider". Let's fix it and release.

- etos: 1.0.2
- etos-client: 1.0.1

- etos-api: 1.0.0
- etos-suite-starter: 1.0.0
- etos-suite-runner: 1.0.0
- etos-test-runner: 1.0.0
- etos-environment-provider: 1.0.1
- etos-library: 1.0.0
- etos-test-runner-containers: 1.0.0

Note that the PATCH number of ETOS increases everytime and that we are not increasing the PATCH number of every tool.

Now if we make a MINOR release of ETOS, this will happen with the versions:

- etos: 1.1.0
- etos-client: 1.1.0
- etos-api: 1.1.0
- etos-suite-starter: 1.1.0
- etos-suite-runner: 1.1.0
- etos-test-runner: 1.1.0
- etos-environment-provider: 1.1.0
- etos-library: 1.1.0
- etos-test-runner-containers: 1.1.0

If you want all PATCH releases for the current MINOR release: ">= 1.1.0 < 1.2.0"

If you want all MINOR releases: ">= 1.0.0 < 2.0.0"

We do not recommend you running latest at all times, since we can break backwards compatability between MAJOR releases. Breaking changes WILL BE announced ahead of time.

4.4 Release Process

The ETOS release process is as follows.

ETOS will do a full release every other wednesday (uneven weeks). This release will be done at any time that day and it will include updating all docker containers, updating the helm chart, creating a tag and release on every repository on github.

The ETOS workflow is to work with milestones.

Each milestone is considered a new release with a version number (*either minor or major*) and the issues that are solved in milestones will be added to that release. (Note that we might hold back issues for a release if they are not deemed stable enough, but we will communicate this by removing the issue from the milestone).

These miletones are updated and created every friday and there will always be at least two future milestones in the works so that users have a heads-up what will be coming in the future.

If backwards compatibility is broken then the feature will be deprecated, the new one activated and an issue for removing it will be added to a major milestone in the future so that users know ahead of time whene a breaking change is coming.

This means that, at worst, users get a 4 week heads up for breaking changes. And by breaking changes, we mean that a developer may have to change something in their code or *ETOS Test Collection* before they can upgrade.

In other words:

- New minor or major release on wednesday, uneven weeks.
- Milestones communicate new releases ahead of time with issues.
  - Example milestone
- Impact of issues decide whether milestone is major or minor.
- Milestones are created and updated every friday.
- There will always be at least two milestones active for two and four weeks in the future, respectively.

4.5 Code Rules

Tox is executed on each pull request to execute all tests, linters and code rules.
This can also be run locally by installing tox and running the command.

- black for general code formatting.
- pydocstyle for checking docstring formats using pep257.
- pylint as the main linter.

4.6 Role specific documentation

Table 4: ETOS Roles

Test Engineer	*Test Automation Engineer*	*System Engineer*
Create collections	Create recipes	Deploy ETOS
Analyze results	Create tests	Infrastructure
What to run?	How to run?	Where to run?

4.6.1 Test Engineer

The test engineers role is to create the recipe collection that defines what to test.

Responsibilities of a test engineer

- Creating collections of recipes.
- *ETOS Client*
- Checking test results

This documentation is unfinished.

4.6.2 Test Automation Engineer

The test automation engineers role is to create the recipes that defines how to execute any test case.

Responsibilities of a test automation engineer

- *ETOS Test Runner* (The recipe part)
- Creating the tests
- Making sure the tests work

This documentation is unfinished.

4.6.3 System Engineer

The system engineers role is to maintain the system and to create provider instructions.

Responsibilities of a system engineer

- *ETOS API*
- *ETOS Suite Starter*
- *ETOS Suite Runner*
- *ETOS Environment Provider*
  - Registering providers
- *ETOS Test Runner*
- Infrastructure
  - Eiffel GraphQL API
  - RabbitMQ
  - Execution spaces
    - Jenkins
    - Kubernetes
    - etc
  - Log areas
    - JFrog Artifactory
    - etc

This documentation is unfinished.

4.7 Services

This page documents the use of all services within ETOS.

If you are just learning about ETOS we recommend the *Getting started* page.

4.7.1 ETOS API

The launcher of ETOS.

Github: ETOS API

The ETOS API is a python microservice running on Falcon that provides the following functionalities:

- Generate and execute and ETOS test suite. (Sending *ETOS Test Collection* to ETOS)

4.7.2 ETOS Client

Main client for ETOS.

Github: ETOS Client

This page describes the ETOS client and its options as well as some examples of how it can be used. ETOS Client is a command-line tool that can be used in Continuous Integration tools such as jenkins as well as locally on workstations.

Why ETOS Client?

The ETOS client is a tool built to make it easier to execute ETOS (Eiffel Test Orchestration System). It is used for starting test suites in ETOS, and reporting the outcome.

Eager to get started

A getting started page can be found here: *Getting started*

CLI Overview and Command Reference

Prerequisites

- Python 3.6 or higher.
- Git (optional)

Installation

There are two methods for installing the package.

Install using pip (Recommended)

Recommended for most users. It installs the latest stable version released. ETOS client can be found on PyPi. If you have the pip package manager installed, then the simplest way of installing ETOS Client is using the following command:

```
pip install etos_client
```

Instructions that virtually take you by the hand and guide you every step of the way is available among our *Step by step* articles.

Install using Git

Recommended for developers who want to install the package along with the full source code. Clone the package repository, and install the package in your site-package directory:

```
git clone "https://github.com/eiffel-community/etos-client.git" client
cd client
pip install -e .
```

General Syntax

The usage example below describes the interface of etos_client, which can be invoked with different combinations of options. The example uses brackets "[ ]" to describe optional elements. Together, these elements form valid usage patterns, each starting with the program's name etos_client.

Below the usage patterns, there is a table of the command-line options with descriptions. They describe whether an option has short/long forms (-h, –help), whether an option has an argument (–identity=<IDENTITY>), and whether that argument has a default value.

```
etos_client [-h] -i IDENTITY -s TEST_SUITE [--no-tty] [-w WORKSPACE] [-a ARTIFACT_
↪DIR] [-r REPORT_DIR] [-d DOWNLOAD_REPORTS] [--iut-provider IUT_PROVIDER] [--
↪execution-space-provider EXECUTION_SPACE_PROVIDER] [--log-area-provider LOG_AREA_
↪PROVIDER] [--dataset DATASET] [--cluster CLUSTER] [--version] [-v]
```

Command-line options

Table 5: ETOS Client

Option	Argument	Meaning
-h, –help	n/a	Show help message and exit.
-i, –identity	IDENTITY	Artifact created identity purl to execute test suite on.
-s, –test-suite	TEST_SUITE	URL to test suite where the test suite collection is stored.
–no-tty	n/a	Disable features requiring a tty.
-w, –workspace	WORKSPACE	Which workspace to do all the work in.
-a,–artifact-dir	ARTIFACT_DIR	Where test artifacts should be stored. Relative to workspace.
-r,–report-dir	REPORT_DIR	Where test reports should be stored. Relative to workspace.
-d,–download-reports	DOWNLOAD_REPORTS	Should we download reports. Can be ‘yes’, ‘y’, ‘no’, ‘n’.
–iut-provider	IUT_PROVIDER	Which IUT provider to use. Defaults to ‘default’.
–execution-space-provider	EXECUTION_SPACE_PROVIDER	Which execution space provider to use. Defaults to ‘default’.
–log-area-provider	LOG_AREA_PROVIDER	Which log area provider to use. Defaults to ‘default’.
–dataset	DATASET	Additional dataset information to the environment provider. Check with your provider which information can be supplied.
–cluster	CLUSTER	Cluster should be in the form of URL to ETOS API.
–version	n/a	Show program’s version number and exit
-v, –verbose	n/a	Set loglevel to DEBUG.

Environment variables

It is possible to specify an option by using one of the environment variables described below.

Precedence of options

If you specify an option by using a parameter on the CLI command line, it overrides any value from the corresponding environment variable.

Table 6: Optional Environment Variables

Name	Required	Meaning
ETOS_GRAPHQL_API	Yes	Specifies URL to Eiffel GraphQL API instance to use.
ETOS_API	Yes	Specifies the URL to the ETOS API for starting tests.
WORKSPACE	no	Which workspace to do all the work in.
IDENTITY	Environment or required input	Artifact created identity purl to execute test suite on.
TEST_SUITE	Environment or required input	URL to test suite where the test suite collection is stored.

Examples

TBD

Known issues

All issues can be found here: https://github.com/eiffel-community/etos-client/issues

Stuck and in need of help

There is a mailing list at: etos-maintainers@google.groups.com or just write an Issue.

4.7.3 ETOS Environment Provider

Github: ETOS Environment Provider

The ETOS environment provider is used to fetch an environment in which to execute test automation on. An environment is the combination of *IUTs*, *Log areas* and *Execution spaces*.

Each provider is in the form of *JSONTas* and must be registered with a name in the environment provider before starting any tests. The ETOS environment provider can have multiple providers registered for each role and a provider can be chosen as an input parameter to *ETOS Client*.

It is recommended to add one provider of each type with the name 'default' in order to make it easier for the user (as they don't have to supply a name to *ETOS Client*.

JSONTas

The ETOS environment provider uses JSONTas in order to structure the providers. Static JSON is still supported (JSONTas just won't do anything).

A JSONTas structure can be quite complex but is useful for generating dynamic JSON structures based on several factors. More examples on JSONTas can be found in their examples

There are a few built-in datastructures in the environment provider as well in order to make life easier.

- json_dumps: Dump an entire segment to string.
- uuid_generate: Generate a UUID4 string

- join: Join a list of strings together.

For the *Execution Space Provider* there is another data structure.

- execution_space_instructions: Used to override the instructions or add more instructions for the *Execution Space*.

General Structure

The general structure of a provider is comrised of at least 3 different main parts.

- List
- Checkout
- Checkin

List

List is where we list the possible and available items. Possible is the number of possible items. If it's 0 then the environment provider will exit with "Could not checkout".

If possible>0 but available is 0, that means there are items possible to checkout but they are not available yet and the environment provider will wait until they are (with a maximum limit).

Note that the listing can be a request to a management system or just a static list of items, but both the 'available' and 'possible' keys must be set.

Checkout

An optional parameter in the provider structure. It's a command which will 'checkout' the item from a management system, if there is one available.

The checkout can also return more values to add to the resulting JSON.

Checkin

An optional parameter in the provider structure. It's a command which will 'check in' the item to a management system, making the item available for others.

IUT Provider

An IUT provider returns a list of JSON data describing an *IUT* (Item Under Test).

The IUT provider follows the *general structure* of 'list', 'checkout' and 'checkin' but also adds another part which is the 'prepare' part.

Prepare

The prepare part of the IUT Provider is defined with stages and steps. A stage is 'where shall this preparation run' and the step is 'what should we run to prepare the IUT'.

There is currently only a single 'stage' and that stage is 'environment_provider' which is run just after the 'checkout' step in the provider.

Each step is a key, value pair where the key is the name of the step and the value is a *JSONTas* structure.

A sample preparation step which will execute three steps. One where the return value is a dictionary, one where the return value is a part of the previous step and the third requests a webpage.

Note that this example does not do anything with the IUT. It is virtually impossible for us to describe the steps required for your technology domain as it all depends on how your systems are set up. This preparation step can request APIs that you've set up internally for various scenarios.

```
{
   "prepare": {
      "stages": {
         "environment_provider": {
            "steps": {
               "step1": {
                  "something": "text",
                  "another": "text2"
               },
               "step2": {
                  "previous_something": "$steps.step1.something"
               },
               "step3": {
                  "$request": {
                     "url": "https://jsonplaceholder.typicode.com/users/1",
                     "method": "GET"
                  }
               }
            }
         }
      }
   }
}
```

Example

A single static *IUT*

```
{
    "iut": {
          "id": "default",
          "list": {
               "possible": [
                    {
                         "type": "$identity.type",
                         "namespace": "$identity.namespace",
                         "name": "$identity.name",
                         "version": "$identity.version",
                         "qualifiers": "$identity.qualifiers",
                         "subpath": "$identity.subpath"
                    }
               ],
               "available": "$this.possible"
          }
    }
}
```

Using a management system

```
{
    "iut": {
        "id": "mymanagementsystem",
        "checkout": {
            "$condition": {
                "then": {
                    "$request": {
                        "url": "http://managementsystem/checkout",
                        "method": "GET",
                        "params": {
                            "mac": "$identity.name"
                        }
                    }
                },
                "if": {
                    "key": "$response.status_code",
                    "operator": "$eq",
                    "value": 200
                },
                "else": "$response.json.message"
            }
        },
        "checkin": {
            "$operator": {
                "key": {
                    "$from": {
                        "item": {
                            "$request": {
                                "params": {
                                    "id": "$iut.id"
                                },
                                "url": "http://managemenetsystem/checkin",
                                "method": "GET"
                            }
                        },
                        "get": "id"
                    }
                },
                "operator": "$eq",
                "value": "$iut.id"
            }
        },
        "list": {
            "possible": {
                "$request": {
                    "url": "http://managementsystem/list",
                    "method": "GET",
                    "params": {
                        "name": "$identity.name"
                    }
                }
            },
            "available": {
                "$filter": {
                    "items": "$this.possible",
                    "filters": [
                        {
```

(continues on next page)

(continued from previous page)

```
                            "key": "checked_out",
                            "operator": "$eq",
                            "value": "false"
                        }
                    ]
                }
            }
        }
    }
}
```

With a preparation step

```
{
    "iut": {
        "id": "default",
        "list": {
            "possible": [
                {
                    "type": "$identity.type",
                    "namespace": "$identity.namespace",
                    "name": "$identity.name",
                    "version": "$identity.version",
                    "qualifiers": "$identity.qualifiers",
                    "subpath": "$identity.subpath"
                }
            ],
            "available": "$this.possible"
        },
        "prepare": {
            "stages": {
                "environment_provider": {
                    "steps": {
                        "step1": {
                            "something": "text",
                            "another": "text2"
                        },
                        "step2": {
                            "previous_something": "$steps.step1.something"
                        },
                        "step3": {
                            "$request": {
                                "url": "https://jsonplaceholder.typicode.com/users/1",
                                "method": "GET"
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Log Area Provider

A log area provider makes sure that the ETOS system knows where and how to store logs and test artifacts during and after execution.

A *log area* has several parts that must exist within the resulting *log area* definition (after listing and checking out)

- livelogs (required): A path to where live logs from the system can be viewed. Used in the test suite events.
- upload (required): How to upload logs to the *log area* system. Follows the same syntax as JSONTas requests (without the '$' signs)
- **logs (optional): Extra formatting on logs.**
  - prepend: an extra value to prepend to log files.
  - join_character: With which character to join the prepended data. Default: '_'

Example using JFrog Artifactory. Checkout any number of artifactory instances, storing logs in a folder based on the Eiffel context. Also prepend IP Address if the *IUT* has an 'ip_address' property.

```
{
    "log": {
        "id": "artifactory",
        "list": {
            "possible": {
                "$expand": {
                    "value": {
                        "livelogs": {
                                "$join": {
                                        "strings": [
                                                "https://artifactory/logs/",
                                                "$context"
                                        ]
                                }
                        },
                        "upload": {
                                "url": {
                                        "$join": {
                                                "strings": [
                                                        "https://artifactory/logs/",
                                                        "$context",
                                                        "/{folder}/{name}"
                                                ]
                                        }
                                },
                                "method": "PUT",
                                "auth": {
                                        "username": "user",
                                        "password": "password",
                                        "type": "basic"
                                }
                        },
                        "logs": {
                                "$condition": {
                                        "if": {
                                                "key": "$iut.ip_address",
                                                "operator": "$regex",
                                                "value": "^(?:[0-9]{1,3}\\.){3}[0-9]
↪{1,3}$"
```

(continues on next page)

(continued from previous page)

```
                                        },
                                        "then": {
                                                "prepend": "$iut.ip_address"
                                        },
                                        "else": {}
                                }
                        }
                    },
                    "to": "$amount"
                }
            },
            "available": "$this.possible"
        }
    }
}
```

Execution Space Provider

An execution space provider makes sure that the ETOS system knows where it can start the *ETOS Test Runner*. The *execution space* must have one required key, which is the 'request' key. This key is the description of how the *ETOS Suite Runner* can start the *ETOS Test Runner* instance.

There is also a field called 'execution_space_instructions' that is dynamically created every time and can be overriden if more information needs to be added. These instructions are the instructions for how to execute the *ETOS Test Runner* docker container.

Example of a Jenkins execution space provider

```
{
  "execution_space": {
        "id": "jenkins",
        "list": {
            "possible": {
                "$expand": {
                    "value": {
                        "request": {
                            "url": "https://jenkins/job/DELEGATION/build",
                            "method": "POST",
                            "headers": {
                                "Accept": "application/json"
                            },
                            "data": {
                                "json": {
                                    "$json_dumps": {
                                        "parameter": [
                                            {"name": "docker", "value": {
                                                "$json_dumps": "$execution_space_
↪instructions"
                                            }
                                            }
                                        ]
                                    }
                                }
                            }
                        }
```

(continues on next page)

(continued from previous page)

```
                    },
                    "to": "$amount"
                }
            },
            "available": "$this.possible"
        }
    }
}
```

Overriding the execution space instructions (note that the '$json_dumps' value has changed).

```
{
  "execution_space": {
        "id": "jenkins",
        "list": {
            "possible": {
                "$expand": {
                    "value": {
                        "instructions": {
                           "$execution_space_instructions": {
                              "environment": {
                                 "MYENV": "environment variable"
                              },
                              "parameters": {
                                 "--privileged": ""
                              }
                           }
                        },
                        "request": {
                            "url": "https://jenkins/job/DELEGATION/build",
                            "method": "POST",
                            "headers": {
                                "Accept": "application/json"
                            },
                            "data": {
                                "json": {
                                    "$json_dumps": {
                                        "parameter": [
                                            {"name": "docker", "value": {
                                                "$json_dumps": "$expand_value.
↪instructions"
                                            }
                                            }
                                        ]
                                    }
                                }
                            }
                        }
                    },
                    "to": "$amount"
                }
            },
            "available": "$this.possible"
        }
    }
}
```

The default instructions are as follows (all can be overriden):

```
instructions = {
    "image": self.dataset.get("test_runner"),
    "environment": {
        "RABBITMQ_HOST": rabbitmq.get("host"),
        "RABBITMQ_USERNAME": rabbitmq.get("username"),
        "RABBITMQ_PASSWORD": rabbitmq.get("password"),
        "RABBITMQ_EXCHANGE": rabbitmq.get("exchange"),
        "RABBITMQ_PORT": rabbitmq.get("port"),
        "RABBITMQ_VHOST": rabbitmq.get("vhost"),
        "SOURCE_HOST": self.etos.config.get("source").get("host"),
        "ETOS_GRAPHQL_SERVER": self.etos.debug.graphql_server,
        "ETOS_API": self.etos.debug.etos_api,
        "ETOS_ENVIRONMENT_PROVIDER": self.etos.debug.environment_provider,
    },
    "parameters": {}
}
instructions["identifier"] = str(uuid4())
instructions["environment"]["SUB_SUITE_URL"] = "{}/sub_suite?id={}".format(
   instructions["environment"]["ETOS_ENVIRONMENT_PROVIDER"],
   instructions["identifier"],
)
```

This is a great place to get value from the optional *dataset* that can be passed to *ETOS Client*. The dataset is always added as a dataset to JSONTas and any value can be referenced with the '$' notation in the JSONTas provider files.

Note that the dataset can be added to any part of the JSON files. See *here* for more examples

```
{
   "instructions": {
      "$execution_space_instructions": {
         "environment": {
            "MYENV": "$my_dataset_variable"
         },
         "parameters": {
            "--privileged": "",
            "--name": "$docker_name"
         }
      }
   }
}
```

Splitter

The test suite spliter algorithms describe the strategy in which to split up the test suites when there are more than 1 *IUT* & *Execution Space*.

The feature of configuring this is not yet implemented. The idea is to either describe it with JSONTas or as extensions to ETOS.

The default splitter algorithm is round robin.

Dataset

A dataset in JSONTas is a data structure with values in it that can be accessed using a '$' notation. For instance if the dataset contains a dictionary:

```
{
   "myname": "Tobias"
}
```

Then that value can be accessed using this JSONTas:

```
{
   "aname": "$myname"
}
```

The dataset structure also has what is called a 'DataStructure' which is a command that can be executed to generate JSON from another source or based on conditions.

Dataset:

```
{
   "myname": "Tobias"
}
```

JSONTas

```
{
   "atitle": {
      "$condition": {
         "if": {
            "key": "$myname",
            "operator": "$eq",
            "value": "Tobias"
         },
         "then": "The best",
         "else": "The worst"
      }
   }
}
```

More examples for JSONTas can be found here.

There are also several DataStructures implemented into the ETOS environment provider explained below.

json_dumps

Dump a subvalue to string.

JSON

```
{
   "a_string": {
      "$json_dumps": {
         "a_key": "a_value"
      }
   }
}
```

Result

```
{
   "a_string": "{\"a_key\": \"a_value\"}"
}
```

uuid_generate

Generate a UUID4 value

JSON

```
{
   "uuid": "$uuid_generate"
}
```

Result

```
{
   "uuid": "a72220c2-eca0-491e-8638-b8e4bdd56f56"
}
```

join

Join a list of strings together. These strings can be JSON dataset values.

JSON

```
{
   "joined": {
       "$join": {
           "strings": [
               "I generated this for you: ",
               "$uuid_generate"
           ]
       }
   }
}
```

Result

```
{
   "joined": "I generated this for you: 96566362-98e0-47f6-abdb-9e3e45fc7c1a"
}
```

API

To register a provider into the environment provider you just have to do a POST request to the 'register' API with the JSONTas description.

Example using curl

```
curl -X POST -H "Content-Type: application/json" -d "{\"execution_space_provider\":
↪$(cat myexecutionspaceprovider.json)}" http://environment-provider/register
```

You can also register multiple providers at once

```
curl -X POST -H "Content-Type: application/json" -d "{\"execution_space_provider\":
↪$(cat myexecutionspaceprovider.json), \"log_are_provider\": $(cat mylogareaprovider.
↪json), \"iut_provider\": $(cat myiutprovider.json)}" http://environment-provider/
↪register
```

Note that it may take a short while for a provider to be updated.

4.7.4 ETOS Library

Github: ETOS Library

This page describes the ETOS library which is a library of common components used by ETOS.

4.7.5 ETOS Suite Runner

Github: ETOS Suite Runner

This page describes the ETOS Suite Runner (ESR). A system for executing test suites in ETOS.

4.7.6 ETOS Suite Starter

The gateway to ETOS.

Github: ETOS Suite Starter

This page describes the ETOS Suite starter.

4.7.7 ETOS Test Runner

The ETOS executor.

Github: ETOS Test Runner

This page describes the ETOS test runner (ETR)

ETOS test runner, or short ETR, is the base test runner that executes tests defined in an ETOS test recipe collection. The test collection contains information from the Test Execution Recipe Collection Created Event (TERCC) and also some additional meta data about the test environment.

Test execution

When ETR executes tests it communicates the following events to the Eiffel message bus (RabbitMQ)

- Test Suite Started
- Test Case Triggered
- Test Case Started
- Test Case Finished
- Test Suite Finished
- Confidence Level Modified

ETR always executes one test suite and several test cases within that suite, thus sending only one test suite started/finished pair but several collections of test case triggered/started/finished

When the test suite has finished executing ETR send a Confidence Level Modified event signaling the Confidence level for the test suite it just executed. The confidence level event is typically in larger aggregations of eiffel events to give basis for CI pipeline driving activities.

Test artifacts

When a test is executed generally there are test artifacts created. These artifacts are typically test results, test logs and specific logs and debugging information from the IUT (item under test). All of the test artifacts are uploaded to a log area , such as Jfrog Artifactory.

4.7.8 ETOS Test Runner Containers

Github: ETOS Test Runner Containers

This page describes the available test runner containers.

A test runner container is a docker image with the required tools and commands installed for executing your particular test recipe.

A typical test container is: "etos_python_test_runner" which comes with python pre-installed.

Available test runner containers

Base Test Runner

The base test runner maintained by the ETOS maintainers. This will include a system and python version that works with the *ETOS Test Runner* and will not take into account any other test runner container other than making sure that they work with *ETOS Test Runner*.

This means that whoever is responsible for a test runner container makes sure that all dependencies that are required exists within that test runner.

Versioning based on Debian distribution name.

- Latest Debian
- Latest python
- Pyenv for installation and selection of python versions
- *ETOS Test Runner*

- eiffel-community/etos-base-test-runner:buster
- eiffel-community/etos-base-test-runner:stretch

Python Test Runner

Maintainer: *Maintainers*

Latest Debian and installs python with pyenv

Versioning based on python (examples)

- eiffel-community/etos-python-test-runner:2.7.16
- eiffel-community/etos-python-test-runner:3.5.3
- eiffel-community/etos-python-test-runner:3.7.3
- eiffel-community/etos-python-test-runner:3.8.3

Go Test Runner

Maintainer: *Maintainers*

Latest Debian and installs Go.

Versioning based on Go (examples)

- eiffel-community/etos-go-test-runner:1.12.6

Rust Test Runner

Maintainer: *Maintainers*

Latest Debian and installs Rust.

Versioning based on Rust (examples)

- eiffel-community/etos-rust-test-runner:1.44.0

4.8 License

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.
```

(continues on next page)

(continued from previous page)

```
   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.
```

(continues on next page)

(continued from previous page)

```
3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.
```

(continues on next page)

(continued from previous page)

```
5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
```

(continues on next page)

(continued from previous page)

```
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

4.9 Maintainers

- Anders Josephson <anders.josephson@axis.com>
- Fredrik Fristedt <fredrik.fristedt@axis.com>
- Tobias Persson <tobias.persson@axis.com>

4.10 Changelog

4.10.1 Version 1.12.0

4.10.2 Version 1.11.0

4.10.3 Version 1.10.0

- a24536f Minor grammar fix in README (#75)

4.10.4 Version 1.9.0

- 1fb2347 Add filebeat sidecars that can be enabled (#72)

4.10.5 Version 1.8.0

4.10.6 Version 1.7.0

4.10.7 Version 1.6.0

4.10.8 Version 1.5.0

4.10.9 Version 1.4.0

4.10.10 Version 1.3.1

4.10.11 Version 1.3.0

- 36cfea0 Allow overrides on tags even on dev (#63)
- 3db7acd Fixed a typo (#61)

4.10.12 Version 1.2.0

- c8b97fc Add documentation for the IUT provider preparation step (#58)

4.10.13 Version 1.1.1

- 7cdb9d1 Remove commands from environment provider and use another docker (#55)
- 7a2f1c1 Fixed docker repositories and tags (#53)

4.10.14 Version 1.1.0

- c5b4031 Added a release process for ETOS. (#45)
- 4b11a6a Update README.rst (#47)
- e019955 Adding installation/deployment support via Helm (#46)
- f4ef08e Adding a checklist to follow when setting up ETOS (#44)
- 95f1596 Add CODEOWNERS file with etos maintainers (#35)
- 06e0a0a Clean up links in the coderules document (#9)

4.10.15 Version 1.0.0

- Initial open source version of ETOS.
- Eiffel Test Orchestration System.
- System for executing generic test suites where separation of concerns is key.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search