Příjemci podpory:

Poskytovatel:

Nástroje pro simulaci útoků a emulaci průniku do kritické informační infrastruktury (VI20202022133):

Název předkládaného výsledku:

SW pro statickou verifikaci bezpečnostních opatření

Typ výsledku dle UV č. 837/2017	Evidenční číslo (příjemce)	Rok vzniku
R	**BEAST-R2**	2022
ISBN-ISSN	**Webový odkaz na výsledek**	**Kde a kdy publikováno**
	https://is.muni.cz/auth/publication/2250099	

Stručná anotace k výsledku: *(max. 8 řádků)*

Předložený SW poskytuje jazyk a nástroje pro specifikaci scénářů statické verifikace bezpečnostních opatření. Umožňuje realizaci těchto scénářů v součinnosti s výsledkem *SW pro ovládání nástrojů ofenzivní bezpečnosti* a je primárně využitelný pro realizaci předpřipraveného penetračního testování a pro přípravu kyberbezpečnostních cvičení.

Řešitelský tým:

Jiří Rája, Milan Boháček, Lukáš Daubner, Michal Drobňák, Ivo Nutár

Obsah

1 Výsledek 2

1.1 Úvod

Výsledek 2 má za úkol vytvoření SW pro statickou verifikaci bezpečnostních opatření. Aby toho dosáhl, spolupracuje s výsledkem 1. Výsledek je aktuálně rozdělen do několika částí. Plánovač, který umožňuje vytvářet komplexní útočné plány a kontrolovat jejich výsledky. Uživatelské rozhraní, které umožňuje jednoduché ovládání, automatizaci či vyhodnocovat jednotlivé kroky útočných scénářů. Nedílnou součástí je i naše testovací prostředí, které nám umožňuje testovat výsledky 1 i 2 společně a dává nám přehled o jejich aktuálním stavu.

1.2 Plánovač

Hlavní úkol plánovače (Cryton Core) je statická verifikace bezpečnostních opatření. Za pomoci vytvořeného jazyku je možné definovat jednotlivé scénáře za účelem testování reálné infrastruktury či při kyberbezpečnostním cvičení jako náhrada za červený tým. Již zmíněný univerzální jazyk pro popis útočných plánů umožňuje popisovat závislosti a návaznosti jednotlivých kroků testování takovým způsobem, aby bylo možné definovat komplexní útočné scénáře. Díky vytvořenému jazyku a automatizaci je možné redukovat požadavky na množství a expertízu členů bezpečnostních či červených týmů. Zároveň umožňuje zjednodušit proces kontinuálního testování cílové infrastruktury s využitím výsledku 1.

Jednou z předností je možnost definovat šablonu, pomocí které je možné tvořit samotné útočné plány s pozměněným zadáním, aby odpovídalo cílové infrastruktuře a testování. Důležitou funkcí je i možnost připravit identický plán pro všechny týmy kyberbezpečnostního cvičení se stejnou časovou osou. Díky tomu je možné dosáhnout stejných podmínek pro všechny a získat konzistentní výsledky. Následně je možné porovnat jednotlivé výsledky z plánů a to i v reálném čase.

Z důvodu většího počtu součástí výsledku bylo třeba vytvořit co nejjednotnější postup instalace. Aktuálně je možné nasadit plánovač pomocí Poetry, Pip, PipX, Docker Compose. Všechny způsoby mají své pro i proti a společně vyhovují všem testovaným podmínkám. Ostatní komponenty je možné nasadit některou ze zmíněných metod.

1.3 Uživatelská rozhraní

Plánovač sám o sobě disponuje REST API, pomocí kterého je možné ho ovládat. Na tomto rozhraní staví jak CLI, tak webové rozhraní. Oba nástroje mají své klady i zápory. V případě, že uživateli nevyhovuje ani jeden, je možné napsat vlastní skript či nástroj, který implementuje již dříve zmíněné REST API. Tímto způsobem je také možné automatizovat ovládání samotného nástroje.

CLI (Cryton CLI) je hlavním nástrojem pro ovládání komplexních útočných scénářů, kontrolu Workerů a zobrazení výsledků verifikace bezpečnostních opatření. Důvodem je převážně jednoduchost nasazení, použití a možnosti automatizace. Automatizace útočných plánů je jeden z hlavních účelů CLI. Z toho důvodu je možné jednoduše přepnout mezi uživatelsky přívětivým a strojově zpracovatelným výstupem.

Webové rozhraní (Cryton Frontend) přináší naopak grafické prostředí, ve kterém uživatel získá přehled o aktuální situaci. Disponuje obdobnou funkcionalitou, jako CLI, ale přidává možnost asistované tvorby plánu, grafické zobrazení výsledků a časové osy plánu, popřípadě možnost ovládat jednotlivé útočné plány.

1.4 E2E testy

Cryton-e2e jsou automatizované "end-to-end"testy, které mají za účel otestovat funkčnost všech komponent Crytona v reálném prostředí. Testy v podstatě prochází každou operací, kterou může Cryton provést, aby se otestovalo, jak vše komunikuje s hardwarem, síťovou konektivitou, externími závislostmi, databázemi a dalšími aplikacemi.

Pro spuštění testů je potřeba mít nainstalováno Poetry a Docker s Compose pluginem. Ve složce s cryton-e2e testy zadáme příkaz `poetry install` a následně vstoupíme do virtuálního prostředí pomocí `poetry shell` . Nyní je potřeba exportovat nastavení pomocí příkazu `export $(cat .env | sed "s/#.*//g"| xargs)` a pro kontrolu správné instalace můžeme použít příkaz `poetry run cryton-e2e` .

Poté lze příkazem `poetry run cryton-e2e build-infrastructure -c all` automaticky postavit veškerou infrastrukturu potřebnou pro testování a příkazem `poetry run cryton-e2e run-tests -t all` automaticky spustit všechny e2e testy. Testy jsou prováděny spouštěním jednotlivých útočných plánů, které využívají veškerých funkcionalit Crytona a zároveň jsou při jejich běhu testovány interaktivní uživatelské akce, které Cryton umožňuje. Poté je prováděna kontrola žádaných výsledků, podle kterých se vyhodnotí úspěšnost testů.

1.5 Instalace

V této sekci se podíváme na samotnou instalaci výsledku 2. K té je potřeba nástroj Cryton Core (plánovač), CLI, webové rozhraní a jejich prerekvizity.

Nejjednodušší způsob, jak dosáhnout instalace již zmíněných nástrojů je použití přiložené Docker Compose konfigurace.

Rozbalíme přiložený balíček SW, otevřeme ho a zadáme příkaz `docker compose up -d` . Nyní se nám postaví vše potřebné a spustí se samotný plánovač, CLI a webové rozhraní. V rámci Compose konfigurace je zahrnuty i prerekvizity:

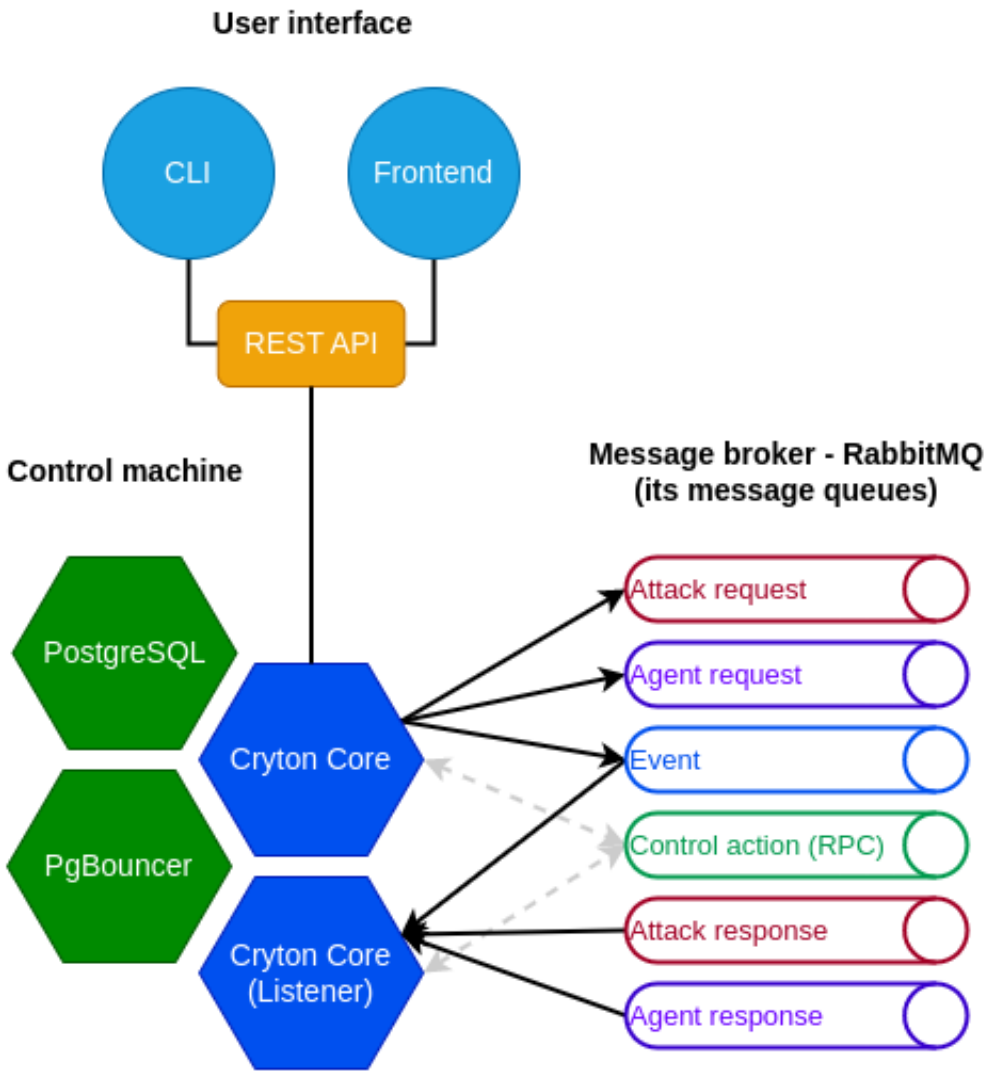
- RabbitMQ server, který zprostředkovává komunikaci mezi plánovačem a orchestrátorem (výsledek 1)
- databáze PostgreSQL, která slouží k uchování získaných výsledků a umožňuje tak jejich následné použití či přezkoumání
- nástroj PgBouncer, který má za úkol zajistit bezproblémovou výměnu dat mezi plánovačem a databází

Pro nasazení výsledku 2 do produkce je možné využít i Ansible role, která umožňuje jednoduché nasazení všech součástí, včetně prerekvizit. Díky tomu je snadné začlenit výsledek 2 do již existující či nové konfigurace pro testování infrastruktury anebo kyberbezpečnostího cvičení.

Pro alternativní způsoby instalace je možné nahlédnout do uživatelské dokumentace. Pro samotné vyzkoušení výsledku doporučujeme demo obsažené ve výsledku 3, případně použít E2E testovací prostředí, jak je popsané v předchozí sekci.

1.6 Architektura

Následující text zdůvodňuje volby aktuálně používaných technologií. Je třeba vzít na vědomí, že tyto technologie nemají být konečné a neměnné. V době vývoje se nejvíce hodily pro daný úkol a v budoucnu se mohou změnit.



Obrázek 1: Architektura systému ukazující vztah jednotlivých komponent

1.6.1 APScheduler

Toto byla první volba pro modul plánovače. Umožňuje nám načasovat úkony tak, aby byly naplánovány na konkrétní čas, den nebo dokonce interval. Jedná se o malou a udržovanou knihovnu, která nevyžaduje mnoho zdrojů ani kapacity. S provozováním této knihovny jako služby jsou sice malé problémy, ale existují způsoby, jak je obejít. Pro tento úkol se zatím nenašla vhodná náhrada.

1.6.2 Django ORM

Na začátku Cryton používal databázi SQLite s přímým přístupem. To se změnilo, protože SQLite není dobré se škálováním do budoucna. Druhou volbou byl PostgreSQL, který zůstal dodnes, ale byl aktualizován pomocí Django ORM. Z této volby také vzešlo použití knihovny Django, která nám umožňuje provozovat REST API.

1.6.3 RabbitMQ

Pro vývoj architektury Master-Worker, kde můžeme zadávat příkazy na dálku, jsme potřebovali nějaký druh synchronního a asynchronního RPC. Z toho důvodu jsme zvolili nástroj RabbitMQ jako systém pro zasílání zpráv.

1.6.4 Metasploit

Metasploit framework je jeden z nejúplnějších a nejpoužitelnějších dostupných nástrojů pro útoky s otevřeným zdrojovým kódem. Cryton jej samozřejmě používá pro některé útočné moduly – většina simulovaných útoků v rámci kyberbezpečnostího cvičení obvykle nějakým způsobem využívá Metasploit. Jeho útočné schopnosti ale nejsou jediným důvodem k jeho použití. Jeho skutečnou výhodou je správa relací. Pokaždé, když otevřete relaci na nějakém počítači, uloží ji pod specifickým ID, které můžete později použít ke komunikaci s cílem. Toto je jedna z hlavních funkcí, kterou můžete použít při provádění útočného scénáře v Crytonu.

1.6.5 Empire

Pro post-exploitační útoky jsme se rozhodli přidat podporu pro open-source projekt s názvem Empire. Empire je post-exploitační framework, který zahrnuje agenty PowerShell Windows, Python 3 Linux/OS X a #. Tento framework nabízí kryptograficky zabezpečenou komunikaci a flexibilní architekturu. To se děje prostřednictvím asynchronní komunikace mezi naší komponentou Worker a serverem Empire c2.

1.6.6 Docker Compose

Při nasazení jednotlivých komponent se ukázalo, že nástroje mají mnoho prerekvizit a bude potřeba určité zjednodušení z důvodu potřeby eliminace chybného nastavení a možnost rychlé reinstalace. Compose je nástroj pro definování a spouštění více-kontejnerových aplikací Docker pomocí jediného příkazu, což spolu s jednoduchým nastavením řeší dříve zmíněné požadavky.

2 Přílohy

2.1 Uživatelská a vývojářská dokumentace

Tato příloha představuje uživatelskou a vývojářskou dokumentaci, jež je dodávána jako součást zdrojového kódu. S přihlédnutím k mezinárodnímu potenciálu předloženého nástroje a probíhající mezinárodní spolupráci, jež tento nástroj mimo jiné využívá, je jazykem textu angličtina. Základní spuštění a ovládání simulátoru je popsáno česky v kapitole 1.

Tato dokumentace je také dostupná v aktuální podobě zde: `https://beast-public.gitlab-pages.ics.muni.cz/cryton/cryton-documentation/`

Table of contents

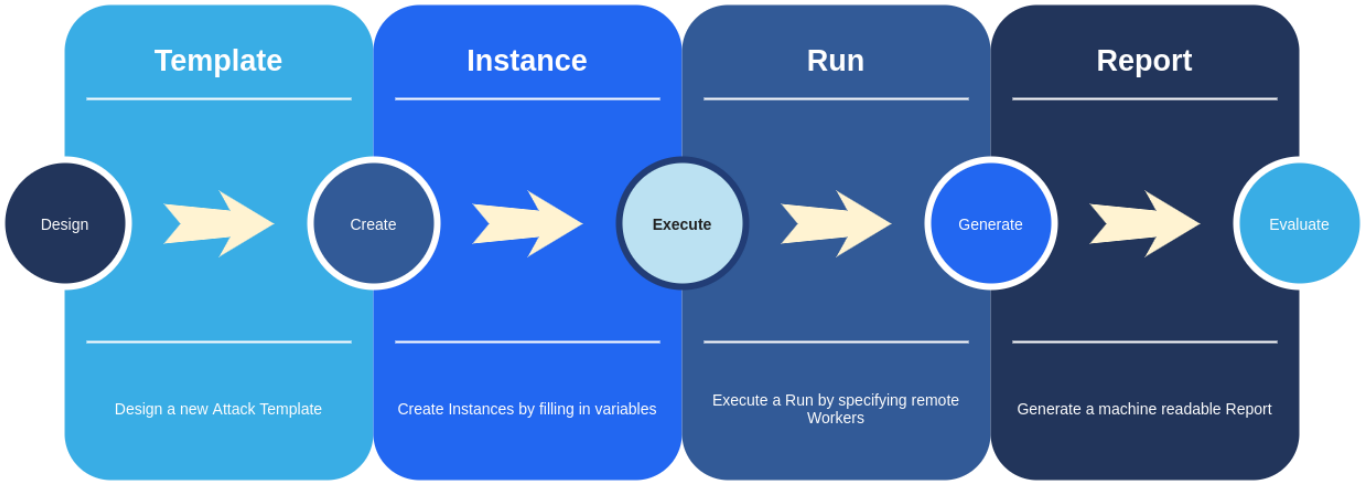
1. Home

1.1 About Cryton

Cryton is a Cron-like red team framework for complex attack scenarios automation and scheduling. Through the usage of Core, Worker, and attack modules it provides ways to plan, execute and evaluate multistep attacks.

All of its open-source components can be found here.

The lifecycle of the Attack scenario in the Cryton context can be seen in the following picture:



With Cryton you can:

- Design an attack **Template**
- Create an **Instance**
- Schedule (or directly Execute) a **Run**
- Generate a **Report**
- Evaluate results

1.2 Purpose

The purpose of the Cryton tool is **to execute complex attack scenarios, in which the system under test is known in advance**. It was designed as such to assist red teams in cybersecurity exercises in means of repeatability of certain attack scenarios. These scenarios are often prepared in advance and reflect vulnerabilities hidden in the blue team's infrastructure.

Imagine you are taking part in a cyber defense exercise as a tutor. The task for your trainees is to defend a system or a whole infrastructure (which you prepared) against an attacker. This system is full of vulnerabilities and misconfigurations (which you prepared as well). Your trainees have e.g. one hour to fix as many of these issues as they can find. Imagine then that you have to check each system for all the fixes to see how your trainees managed to succeed. How would you do that effectively?

This is where Cryton comes to play. If you know all the vulnerabilities in the trainees' system - and you do - you can prepare an attack scenario to check if they are still available and working after the fix. Cryton will execute the plan against all targets you tell it to and then generate reports (human and machine process-able). You can then not only see, which attack steps did succeed on which system, but also score your trainees based on these results.

With this in mind, you should not expect Cryton to be some kind of evil artificial intelligence capable of taking over the world. It is simply a scheduler for python modules. Scheduler which executes these modules according to some execution tree with conditions based on each step of the scenario. Each module is a script orchestrating some well-known attack tools, but that is it.

1.3 Support

Cryton toolset is tested and targeted primarily on **Debian** and **Kali Linux**, however it **should** be possible to use it everywhere if the requirements are met. Please keep in mind that **only the latest version is supported** and issues regarding different OS or distributions may **not** be resolved.

More detailed information can be found in each release's documentation or each project's README.

1.4 Technological decisions

The next section tries to explain the choices for currently employed technologies. Please take into account that these technologies are not supposed to be final and unchangeable. They just appeared to be best suited for the task at the time of development, they may change in the future.

1.4.1 APScheduler

This was the first choice made for the scheduler module. It allows you to time your python function to be scheduler on a specific time or day or even interval. It is pretty lightweight and does not need much in terms of resources or capacity. So far I have not found anything better suited for the task. There is though one small problem with running it as a service, but there are ways around it.

1.4.2 Django ORM

In the beginning, Cryton used the SQLite database with direct access. That changed as SQLite is not good with scaling for the future. The second choice was PostgreSQL, which stayed to this day, but it was updated with the use of Django ORM. Using the Django REST framework for the REST interface also emerged from this choice.

1.4.3 Rabbit MQ

For developing Master-Worker architecture, where you can issue commands remotely, we needed some kind of RPC. Although, as experience showed us, we also needed it to be asynchronous. That's why we chose a messaging system Rabbit MQ.

1.4.4 Metasploit

I guess everyone in the IT security field has heard about the Metasploit framework. It is one of the most complete and usable open-source attack tools available. Of course, Cryton uses it for some attack modules - the majority of simulated attacks in CDXs usually do use Metasploit in some way. But its attacking capabilities are not the only reason to use it. Its real advantage is Metasploit's session management. Every time you open a session to some machine it stores it under a specific ID which you can later use to communicate with the target. This is one of the main features you can use while executing your attack scenario in Cryton.

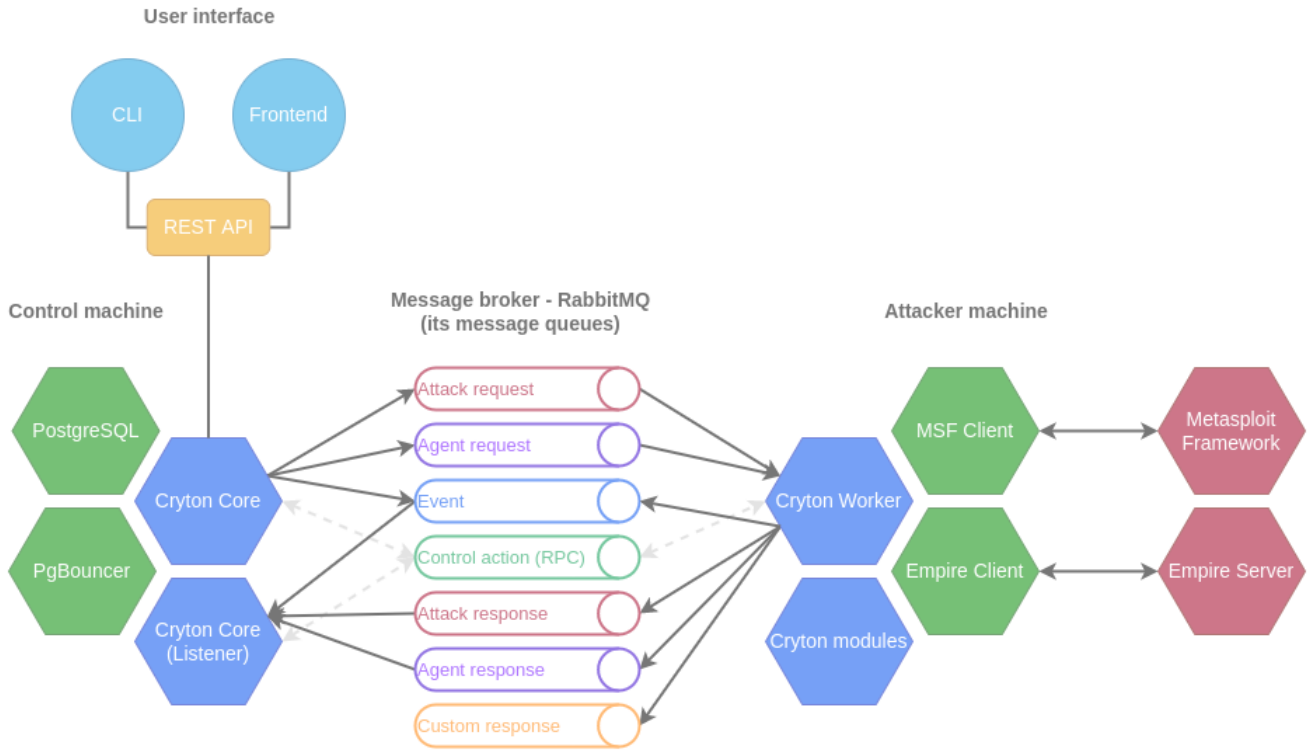
1.4.5 Empire

For post-exploitation attacks, we decided to add support for an open-source project called Empire. Empire is a post-exploitation framework that includes pure-PowerShell Windows agents, Python 3 Linux/OS X agents, and C# agents. The framework offers cryptological-secure communications and flexible architecture. This is done via asynchronous communication between our Worker component and an Empire c2 server.

1.4.6 Docker (compose)

To bundle everything together and make the deployment effortless, we use docker-compose.

2. Architecture



2.1 Core

All main functionality is implemented in Cryton Core. As the name suggests, this is the core component of the Cryton toolset. It provides all the functionality for parsing the attack Plan(s), creation of Executions, and scheduling (executing) of Runs.

For issuing commands to Core, REST API allows HTTP requests and is utilized by CLI or Frontend for user interactions.

2.2 CLI and Frontend

There are two ways to interact with Cryton or, more precisely, to use its API. One of them is Cryton CLI, a python script that we can use to run actions from the terminal simply. A slightly more user-friendly is Cryton Frontend, a graphical web interface providing additional functionality to improve the user experience and make the automation process smoother. It uses Cryton Core's REST API as well.

There is also the option to develop a completely custom application that will send HTTP requests to Cryton's REST API. If you are interested, you can find more about Cryton's REST API here.

2.3 Do I need to have all components installed?

If you are starting with Cryton, you should install all the main components - CLI and Core.

Depending on your use case, the composition of Cryton may vary. For example, installing the Frontend is unnecessary if you wish to control Cryton using only the CLI.

3. Starting point

3.1 Core

3.1.1 Description

Cryton Core is the center point of the Cryton toolset. It is used for: - Creating, planning, and scheduling attack scenarios. - Generating reports from attack scenarios. - Controlling Workers and scenarios execution.

To be able to execute the attack scenarios, you also need to install the **Cryton Worker** and **Cryton CLI** package. Optionally you can install Cryton Frontend for non-command line experience.

Cryton toolset is tested and targeted primarily on **Debian** and **Kali Linux**. Please keep in mind that **only the latest version is supported** and issues regarding different OS or distributions may **not** be resolved.

Link to the repository.

3.1.2 Settings

Cryton Core uses environment variables for its settings. Please update them to your needs.

name	value	example	description
CRYTON_CORE_RABBIT_HOST	string	127.0.0.1	RabbitMQ server host.
CRYTON_CORE_RABBIT_PORT	int	5672	RabbitMQ server port.
CRYTON_CORE_RABBIT_USERNAME	string	admin	Username for RabbitMQ server login.
CRYTON_CORE_RABBIT_PASSWORD	string	mypass	Password for RabbitMQ server login.
CRYTON_CORE_DB_HOST	string	127.0.0.1	Postgres server host.
CRYTON_CORE_DB_PORT	int	5432	Postgres server port.
CRYTON_CORE_DB_NAME	string	cryton	Used Postgres database name. **(do not change, if you don't know what you're doing)**
CRYTON_CORE_DB_USERNAME	string	cryton	Username for Postgres server login.
CRYTON_CORE_DB_PASSWORD	string	cryton	Password for Postgres server login.
CRYTON_CORE_Q_ATTACK_RESPONSE	string	cryton_core.attack.response	Queue name for processing attack responses. **(do not change, if you don't know what you're doing)**
CRYTON_CORE_Q_AGENT_RESPONSE	string	cryton_core.agent.response	Queue name for processing agent responses. **(do not change, if you don't know what you're doing)**
CRYTON_CORE_Q_EVENT_RESPONSE	string	cryton_core.event.response	Queue name for processing event responses. **(do not change, if you don't know what you're doing)**
CRYTON_CORE_Q_CONTROL_REQUEST	string	cryton_core.control.request	Queue name for processing control requests.

name	value	example	description **(do not change, if you don't know what you're doing)**
CRYTON_CORE_DEBUG	boolean	false	Make Core run with debug output.
CRYTON_CORE_TZ	string	UTC	Internally used timezone. **(do not change, if you don't know what you're doing)**
CRYTON_CORE_DEFAULT_RPC_TIMEOUT	int	120	Timeout (in seconds) for RabbitMQ RPC requests.
CRYTON_CORE_API_SECRET_KEY	string	XF37..56 chars..6HB3	Key (64 chars) used by REST API for cryptographic signing. More information can be found here.
CRYTON_CORE_API_PUBLIC_PORT	int	8000	Port on which the Apache reverse proxy will be served (this only affects the *cryton_apache* Compose configuration).
CRYTON_CORE_API_ALLOWED_HOSTS	list of strings separated by space	*	Domain names that the site can serve. **(do not change, if you don't know what you're doing)** More information can be found here.
CRYTON_CORE_API_STATIC_ROOT	string	/var/www/example.com/static/	Directory for storing static files. **(do not change, if you don't know what you're doing)** More information can be found here.
CRYTON_CORE_API_USE_STATIC_FILES	boolean	true	

name	value	example	description
			Whether to serve static files or not. **(do not change, if you don't know what you're doing)**
CRYTON_CORE_CPU_CORES	int	3	The maximum number of CPU cores (processes) Cryton Core can utilize. **(do not change/set/export, if you don't know what you're doing)** This affects the speed of starting/consuming Steps/Rabbit requests. Set value to `auto` for the best CPU utilization.
CRYTON_CORE_EXECUTION_THREADS_PER_PROCESS	int	7	How some payloads or Rabbit's channel consumers should be distributed. **(do not change/set/export, if you don't know what you're doing)** This affects the speed of starting/consuming Steps/Rabbit requests.
CRYTON_CORE_APP_DIRECTORY	string	~/.local/cryton-core/	Path to the Cryton Core directory. **(do not change/set/export, if you don't know what you're doing)** If changed, update the commands in this guide accordingly.

To save the settings **create an app directory**:

```
mkdir ~/.local/cryton-core/
```

The directory will be also used to store logs and other data created by Cryton Core.
This doesn't apply to the Docker installation. It will be available in the same directory as the Dockerfile (`/path/to/cryton-core/cryton-core`).

To make the installation easier, we need to set our target version first. Versions can be found here. Export the $C_VERSION variable to match the desired version:

```
export C_VERSION=version
```

Next, we download example settings:

```
curl -o ~/.local/cryton-core/.env https://gitlab.ics.muni.cz/beast-public/cryton/cryton-core/-/raw/$C_VERSION/.env
```

Update these settings to your needs.

Overriding the settings

NOTICE: This doesn't apply to the Docker Compose installation.

To override the persistent settings, you can set/export the variables yourself using the **export** command (use **unset** to remove the variable). For example:

```
export CRYTON_CORE_DEBUG=false
```

3.1.3 Prerequisites

Install these prerequisites before running Cryton Core. Feel free to use our Compose configuration.

- PostgreSQL database (optionally, use a Docker image)
- RabbitMQ server (optionally, use a Docker image)
- PgBouncer (optionally, use a Docker image)

Using Compose configuration

The easiest way to satisfy the prerequisites is to use our predefined Compose configuration. To do so, you need to install Docker Compose.

Now, continue to the installation, where you'll find a guide on how to install the prerequisites using Compose:

- using pipx
- using Docker Compose
- development

3.1.4 Installation (using pip/pipx)

Cryton Core is available in the PyPI and can be installed using *pip* (`pip install --user cryton-core`). However, we **highly recommend** installing the app in an isolated environment using pipx.

Requirements

Install the following requirements: - Python >=3.8 - pipx

Install prerequisites for pipx installation using Compose config

Only perform this step if you want to install the prerequisites mentioned here using Docker Compose.

First, make sure you have:

- installed Docker Compose
- correctly set the settings, you can't change the settings on a running container

To install the prerequisites simply use:

```
cd ~/.local/cryton-core/
curl -o ~/.local/cryton-core/docker-compose.prerequisites.yml https://gitlab.ics.muni.cz/beast-public/cryton/cryton-core/-/raw/$C_VERSION/docker-compose.prerequisites.yml
curl -o ~/.local/cryton-core/docker-compose.prerequisites.override.yml https://gitlab.ics.muni.cz/beast-public/cryton/cryton-core/-/raw/$C_VERSION/docker-compose.prerequisites.override.yml
docker compose -f docker-compose.prerequisites.yml -f docker-compose.prerequisites.override.yml up -d --build
```

Update the settings accordingly:

```
CRYTON_CORE_RABBIT_HOST=localhost
CRYTON_CORE_DB_HOST=localhost
CRYTON_CORE_DB_PORT=16432
```

Installing with pipx

Once you have *pipx* ready on your system, you can start the installation:

```
pipx install cryton-core
```

Make sure you've correctly set the settings.

If you're not using a reverse proxy, set `CRYTON_CORE_API_USE_STATIC_FILES=false`.

Everything should be set. Check out the usage section.

3.1.5 Installation (using Docker Compose)

Cryton Core can be installed using Docker Compose.

First, we have to clone the repo and switch to the correct version.

```
git clone https://gitlab.ics.muni.cz/beast-public/cryton/cryton-core.git
cd cryton-core
git checkout $C_VERSION
```

Requirements

- Docker Compose

Add yourself to the group *docker*, so you can work with Docker CLI without *sudo*:

```
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
docker run hello-world
```

Install prerequisites for Compose deployment using Compose config

Only perform this step if you want to install the prerequisites mentioned here using Docker Compose.

First, make sure you have:

- installed Docker Compose
- correctly set the settings, you can't change the settings on a running container

To install the prerequisites simply use:

```
docker compose -f docker-compose.prerequisites.yml up -d --build
```

Update the settings accordingly:

```
CRYTON_CORE_RABBIT_HOST=cryton_rabbit
CRYTON_CORE_DB_HOST=cryton_pgbouncer
```

Installing and running with Docker Compose

Make sure you've correctly set the settings. You can't change the settings on a running container.

Finally, copy your settings:

```
cp ~/.local/cryton-core/.env .env
```

We are now ready to build and start the Core:

```
docker compose up -d --build
```

After a while you should see a similar output:

```
[+] Running
6/6
 ⠿ Container cryton_rabbit     Started
 ⠿ Container cryton_apache     Started
 ⠿ Container cryton_db         Healthy
 ⠿ Container cryton_pgbouncer  Started
 ⠿ Container cryton_app        Started
 ⠿ Container cryton_listener   Started
```

Everything should be set. Check if the installation was successful and the Core is running by either installing Cryton CLI or testing REST API with curl:

```
curl localhost:8000/api/
```

Expected result:

```
{"runs":"http://localhost:8000/cryton/api/v1/runs/","plans":"http://localhost:8000/cryton/api/v1/plans/",
"plan_executions":"http://localhost:8000/cryton/api/v1/plan_executions/","stages":"http://localhost:8000/cryton/api/v1/stages/",
"stage_executions":"http://localhost:8000/cryton/api/v1/stage_executions/","steps":"http://localhost:8000/cryton/api/v1/steps/",
"step_executions":"http://localhost:8000/cryton/api/v1/step_executions/","workers":"http://localhost:8000/cryton/api/v1/workers/"}
```

Docker can sometimes create dangling (`<none>:<none>`) images which can result in high disk space usage. You can remove them using:

```
docker image prune
```

3.1.6 Development

To install Cryton Core for development, you must install Poetry.

Clone the repository and then go to the correct directory:

```
git clone https://gitlab.ics.muni.cz/beast-public/cryton/cryton-core.git
cd cryton-core
```

Install prerequisites for development using Compose config

Only perform this step if you want to install the prerequisites mentioned here using Docker Compose.

First, make sure you have:

- installed Docker Compose
- correctly set the settings, you can't change the settings on a running container

To install the prerequisites simply use:

```
docker compose -f docker-compose.prerequisites.yml -f docker-compose.prerequisites.override.yml up -d --build
```

Update the settings accordingly:

```
CRYTON_CORE_RABBIT_HOST=localhost
CRYTON_CORE_DB_HOST=localhost
CRYTON_CORE_DB_PORT=16432
```

Installation and setup with Poetry

Now we can install the project:

```
poetry install
```

To spawn a shell use:

```
poetry shell
```

Make sure you've correctly set the settings.
To override the settings quickly, you can use this handy one-liner:

```
export $(grep -v '^#' .env | xargs)
```

If you're not using a reverse proxy, set `CRYTON_CORE_API_USE_STATIC_FILES=false`.

Everything should be set, check out the usage section.

3.1.7 Usage

NOTICE: If you're using Docker Compose to install the app, you don't need to migrate the database or start the services mentioned in this section.

Move to the app directory, since some files and directories can be spawned in a relative path

```
cd ~/.local/cryton-core/
```

Use the following to invoke the app:

```
cryton-core
```

You should see a help page:

```
Type 'cryton-core help <subcommand>' for help on a specific subcommand.

Available subcommands:
...
```

To learn about each command's options use:

```
cryton-core help <your command>
```

Before we do anything, **we need to migrate the database**:

```
cryton-core migrate
```

To be able to use Cryton Core, we need to start the application and its RabbitMQ listener (start each in a separate shell or use the `nohup` command).

First, **start the application**:

```
cryton-core runserver 0.0.0.0:8000
```

Use the Gunicorn server for the production deployment:

```
cryton-core startgunicorn
```

Start the RabbitMQ listener:

```
cryton-core startlistener
```

REST API and control

REST API is the only way to communicate with Cryton Core. It is by default running at http://0.0.0.0:8000. Interactive documentation can be found at http://0.0.0.0:8000/doc.

To be able to control Cryton Core, you have to send requests to its REST API. This can be done manually, or via Cryton CLI or Cryton Frontend.

Execution example

Every Run can be described by a simple formula:

```
Plan template + inventory = Plan instance
Plan instance + Worker = Plan execution
Plan instance + Workers = Run
```

1. Choose or design plan template
Choose one of the YAML plan templates (in the `examples` directory) or design your own.

2. Create Plan instance
Plan templates can utilize a number of variables that need to be provided during the instantiation process. Do this by specifying an inventory file.

3. Create Worker
Define Worker(s) that will be used to execute the Plan instance.

4. Create Run
Create a Run by choosing the Plan instance and providing a list of Workers for execution.

5. Schedule or execute Run
You can either schedule Run for a specific date/time, or execute it directly. Run will then be executed on every Worker simultaneously.

6. Read Run Report
A report can be generated anytime during the execution (also compliant with YAML format). It contains a list of Stages/Steps and their results.

3.2 CLI

3.2.1 Description

Cryton CLI is a command line interface used to interact with Cryton Core (its API).

To be able to execute attack scenarios, you also need to install **Cryton Core** and **Cryton Worker** tools.

Cryton toolset is tested and targeted primarily on **Debian** and **Kali Linux**. Please keep in mind that **only the latest version is supported** and issues regarding different OS or distributions may **not** be resolved.

Link to the repository.

3.2.2 Settings

Cryton CLI uses environment variables for its settings. Please update them to your needs.

name	value	example	description
CRYTON_CLI_TIME_ZONE	string	AUTO	What timezone to use for scheduling (for example when scheduling a Run). Use the `AUTO` value to use your system timezone.
CRYTON_CLI_API_HOST	string	127.0.0.1	Cryton Core's API address.
CRYTON_CLI_API_PORT	int	8000	Cryton Core's API port.
CRYTON_CLI_API_SSL	boolean	false	Use SSL to connect to REST API.
CRYTON_CLI_API_ROOT	string	api/	REST API URL. **(do not change, if you don't know what you're doing)**
CRYTON_CLI_APP_DIRECTORY	string	~/.local/cryton-cli/	Path to the Cryton CLI directory. **(do not change/set/export, if you don't know what you're doing)** If changed, update the commands in this guide accordingly.

To save the settings **create an app directory**:

```
mkdir ~/.local/cryton-cli/
```

Next, we download example settings (**change the version to match the app version**):

```
curl -o ~/.local/cryton-cli/.env https://gitlab.ics.muni.cz/beast-public/cryton/cryton-cli/-/raw/<version>/.env
```

Update these settings to your needs.

Overriding the settings

To override the persistent settings, you can set/export the variables yourself using the **export** command (use **unset** to remove the variable). For example:

```
export CRYTON_CLI_API_HOST=127.0.0.1
```

Some environment variables can be overridden in CLI. Try using `cryton-cli --help`.

3.2.3 Installation

Cryton CLI is available in the PyPI and can be installed using *pip* (`pip install --user cryton-cli`). However, we **highly recommend** installing the app in an isolated environment using pipx.

Requirements

Install the following requirements: - Python >=3.8 - pipx

Installing with pipx

Once you have *pipx* ready on your system, you can start the installation:

```
pipx install cryton-cli
```

Make sure you've correctly set the settings.

Optionally, you can set up shell completion.

Everything should be set, check out the usage section.

3.2.4 Development

To install Cryton CLI for development, you must install Poetry.

Clone the repository:

```
git clone https://gitlab.ics.muni.cz/beast-public/cryton/cryton-cli.git
```

Then go to the correct directory and install the project:

```
cd cryton-cli
poetry install
```

To spawn a shell use:

```
poetry shell
```

Make sure you've correctly set the settings.
To override the settings quickly, you can use this handy one-liner:

```
export $(grep -v '^#' .env | xargs)
```

Optionally, you can set up shell completion.

Everything should be set. Check out the usage section.

3.2.5 Usage

Use the following to invoke the app:

```
cryton-cli
```

You should see a help page:

```
Usage: cryton-cli [OPTIONS] COMMAND [ARGS]...

  A CLI wrapper for Cryton API.

Options:
  ...
```

Please keep in mind that the Cryton Core must be running and its API must be reachable.

To change the default API host/port use *-H* and *-p* options (to change them permanently, see the settings section).

```
cryton-cli -H 127.0.0.1 -p 8000 <your command>
```

To learn about each command's options use:

```
cryton-cli <your command> --help
```

For a better understanding of the results, we highlight the successful ones with **green** and the others with **red** color.

Example

1. CREATE PLAN TEMPLATE

Create a plan template using a file containing the desired plan YAML.

```
cryton-cli plan-templates create my-plan.yml
```

Desired output:

```
Template successfully created! (<response detail>).
```

2. CREATE PLAN INSTANCE

Create a Plan instance with the saved plan template.

```
cryton-cli plans create 1
```

Create a Plan instance using the template and an inventory file.

```
cryton-cli plans create 1 -i inventory_file
```

Desired output:

```
Plan successfully created! (<response detail>).
```

3. CREATE WORKER

To execute Plans (Runs) we have to define a Worker(s).

```
cryton-cli workers create customName -d "This is my first Worker!"
```

Desired output:

```
Worker successfully created! (<response detail>).
```

4. CREATE RUN

Create a Run by choosing a Plan instance and providing a list of Workers for execution.

```
cryton-cli runs create 1 1
```

Desired output:

```
Run successfully created! (<response detail>).
```

5. SCHEDULE OR EXECUTE RUN

You can either schedule the Run for a specific date/time or execute it directly. Run will then be executed on every Worker simultaneously.

Execute Run

```
cryton-cli runs execute 1
```

Desired output:

```
Run successfully executed! (Run 1 was executed.).
```

Schedule Run
You can schedule a Run using the local timezone.

```
cryton-cli runs schedule 1 2020-06-08 10:00:00
```

Desired output:

```
Run successfully scheduled! (Run 1 is scheduled for 2020-06-08 10:00:00.).
```

Or you can schedule it using UTC timezone with the flag `--utc-timezone`. Otherwise, your preset timezone is used.

6. READ RUN REPORT

A report can be generated anytime during the execution (also compliant with YAML format). It contains a list of Stages/Steps and their results.

```
cryton-cli runs report 1
```

Desired output:

```
Successfully created Run's report! (file saved at: /tmp/report_run_1_2020-06-08-10-15-00-257994_xdQeV)
```

Timestamps are displayed in UTC timezone by default. Use the `--localize` flag to display them using your preset timezone.

3.2.6 Shell completion

Shell completion is available for the *Bash*, *Zsh*, and *Fish* shell and has to be manually enabled (**the tool must be installed first**).

Bash

First, **create an app directory** (if you haven't already):

```
mkdir ~/.local/cryton-cli/
```

Generate and save the completion script:

```
_CRYTON_CLI_COMPLETE=bash_source cryton-cli > ~/.local/cryton-cli/cryton-cli-complete.bash
```

Source the file in the `~/.bashrc` file:

```
echo ". ~/.local/cryton-cli/cryton-cli-complete.bash" >> ~/.bashrc
```

You may need to restart your shell for the changes to take effect.

Zsh

First, **create an app directory** (if you haven't already):

```
mkdir ~/.local/cryton-cli/
```

Generate and save the completion script:

```
_CRYTON_CLI_COMPLETE=zsh_source cryton-cli > ~/.local/cryton-cli/cryton-cli-complete.zsh
```

Source the file in the `~/.zshrc` file:

```
echo ". ~/.local/cryton-cli/cryton-cli-complete.zsh" >> ~/.zshrc
```

You may need to restart your shell for the changes to take effect.

Fish

Generate and save the completion script:

```
_CRYTON_CLI_COMPLETE=fish_source cryton-cli > ~/.config/fish/completions/cryton-cli-complete.fish
```

You may need to restart your shell for the changes to take effect.

3.3 Frontend

3.3.1 Description

Cryton Frontend is a graphical interface used to interact with Cryton Core (its API).

To be able to execute attack scenarios, you also need to install **Cryton Core** and **Cryton Worker** tools.

Cryton toolset is tested and targeted primarily on **Debian** and **Kali Linux**. Please keep in mind that **only the latest version is supported** and issues regarding different OS or distributions may **not** be resolved.

Link to the repository.

3.3.2 Settings

Before installation, you need to set the environment variables to match **Cryton Core** settings. Environment variables can be found in `src/environments/`. For production build, modify the *environment.prod.ts* file, else modify the *environment.ts* file.

crytonRESTApiHost: REST API host (localhost by default).

crytonRESTApiPort: REST API port (8000 by default).

refreshDelay: Sets artificial delay in milliseconds for refresh API requests, users usually react better if the requests don't happen instantly, but they can see a tiny bit of loading. Initial API request doesn't use delay, this is only for refreshing data (300 milliseconds by default).

3.3.3 Installation (using Docker Compose)

Cryton Frontend can be installed using Docker Compose.

Requirements

- Docker Compose

Add yourself to the group *docker*, so you can work with Docker CLI without *sudo*:

```
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
docker run hello-world
```

Installing and running with Docker Compose

First, we have to clone the repo and switch to the correct version (if you haven't already).

```
git clone https://gitlab.ics.muni.cz/beast-public/cryton/cryton-frontend.git
cd cryton-frontend
```

Make sure you've correctly set the settings. You can't change the settings on a running container.

We are now ready to build and start the frontend:

```
docker compose up -d --build
```

Docker Compose will automatically build the app for production with minimal dependencies and deploy it on a Nginx server inside the container. The default port is set to **8080**, you can change this setting in the `docker-compose.yml` file.

3.3.4 Installation (manual)

Requirements

- npm

Installing manually using npm

1. Clone this repository and cd into it.
2. Run `npm install`, npm will take care of installing all the dependencies.
3. Based on the needs you can:

- Serve the app by running `ng serve` or `ng serve --prod` for production settings.
- Only use `ng serve` for development/testing, in a real production environment use either docker installation or production build deployed on a production-ready web server (for example Nginx).
- App will now be available on **localhost:4200**
- To change the port use argument `--port [port]`
- Build the app by running `npm run build` or `npm run build-prod` for production.
- You can find the build in the **/dist** folder.

3.3.5 Development

Requirements

- npm

Installing and running with npm

- App uses husky to run pre-commit hooks. These include:
- Code formatting with Prettier.
- Linting with ESLint.
- Running unit tests with Karma.
- To start development:

a. Install dependencies with `npm install`.

b. Run `npm start` to run the development server. The app will now listen for changes and refresh itself on every change in the project's filesystem.

3.3.6 Usage

Please keep in mind that the Cryton Core must be running and its API must be reachable.

Use in-app help pages to learn about usage.

3.4 Deployment with Ansible

3.4.1 Description

This project is used for deploying the Cryton toolset using Ansible.

Cryton toolset is tested and targeted primarily on **Debian** and **Kali Linux**. Please keep in mind that **only the latest version is supported** and issues regarding different OS or distributions may **not** be resolved.

- Make sure you run Ansible using Python3 (`ansible_python_interpreter: /usr/bin/python3`).
- Supposedly there is no need for `gather_facts`.
- Run roles using sudo (become).
- For the best experience specify `cryton_COMPONENT_version` where *COMPONENT* is depending on the role (core, cli, worker, modules) and select the latest version (**the master branch is not stable**).
- To update the default variables stored in .env files use `cryton_COMPONENT_environment` where *COMPONENT* is depending on the role (core, cli, worker).
- Values for each role can be found in `cryton-deploy/roles/ROLE/defaults/main.yml`.
- Once you update cryton_COMPONENT_environment, **make sure you've updated all the variables**.

3.4.2 Roles

deploy-core

Install prerequisites, dependencies (RabbitMQ, Postgres, and PgBouncer), and Core using Docker Compose.
Core's REST API is by default served on port 8000.

Override environment variables as specified in the settings. In the Ansible playbook use the following:

```
- role: deploy-core
  cryton_core_environment:
    VARIABLE_TO_OVERRIDE: new_value
    ...
```

For all available role variables, check `cryton-deploy/roles/deploy-core/defaults/main.yml`.

To create, update, and load the Docker configuration saved in /etc/docker/daemon.json, set `update_docker_daemon_configuration: yes`, and use `docker_daemon_configuration` dictionary to create the configuration.
Example and default:

```
docker_daemon_configuration:
  mtu: 1442
```

deploy-worker (with modules)

Install prerequisites and Worker with modules using pipx.
Start the Worker afterward in the background (you have to check for errors manually in the `{{ cryton_worker_output_file }}`).

To start msfrpcd in the background use `start_msfrpcd: yes`.
Set `cryton_cli_runas_user` to the correct user for whom will the Worker be installed.

Optionally, Worker can be installed in a mode fitting for development purposes. To enable this mode, set `development: True` variable for Ansible. This will install and run the Worker using poetry.

Override environment variables as specified in the settings. In the Ansible playbook use the following:

```
- role: deploy-worker
  cryton_worker_environment:
    VARIABLE_TO_OVERRIDE: new_value
    ...
```

For all available role variables, check `cryton-deploy/roles/deploy-worker/defaults/main.yml`.

For running the Ansible playbook, community.general module is needed. Install it by `ansible-galaxy collection install community.general`.

deploy-cli

Install prerequisites, dependencies, and CLI in `~/.local/bin/cryton-cli` using pipx, register it to PATH, and export .env vars into `~/.local/cryton-cli/.env`.

Set `cryton_cli_runas_user` to the correct user for whom will the Worker be installed.

Override environment variables as specified in the settings. In the Ansible playbook use the following:

```
- role: deploy-cli
  cryton_cli_environment:
    VARIABLE_TO_OVERRIDE: new_value
    ...
```

For all available role variables, check `cryton-deploy/roles/deploy-cli/defaults/main.yml`.

register-worker

Register Worker in Core using CLI.

Specify `cryton_worker_name`, `cryton_worker_description`, and `cryton_cli_runas_user` to the correct user with access to the CLI.

Override environment variables as specified in the settings. In the Ansible playbook use the following:

```
- role: register-worker
  cryton_cli_environment:
    VARIABLE_TO_OVERRIDE: new_value
    ...
```

For all available role variables, check `cryton-deploy/roles/register-worker/defaults/main.yml`.

deploy-frontend

Install prerequisites and frontend for Cryton Core API using Docker Compose. The frontend is by default served on port 8080.

!This role requires the host to have at least 2048 MB RAM and 2 CPU cores (tested with AMD Ryzen 5 5600x) otherwise the Frontend installation might fail.!

Override environment variables as specified in the settings. In the Ansible playbook use the following:

```
- role: deploy-frontend
  cryton_frontend_environment:
    VARIABLE_TO_OVERRIDE: new_value
    ...
```

For all available role variables, check `cryton-deploy/roles/deploy-frontend/defaults/main.yml`.

To create, update, and load the Docker configuration saved in /etc/docker/daemon.json, set `update_docker_daemon_configuration: yes`, and use `docker_daemon_configuration` dictionary to create the configuration.
Example and default:

```
docker_daemon_configuration:
  mtu: 1442
```

3.4.3 Examples

Deploy Core

```
- name: Deploy Core
  hosts: c2-server
  become: yes
```

```
  roles:
    - role: deploy-core
```

Deploy Worker (with modules)

```
- name: Deploy Worker with modules
  hosts: attacker
  become: yes
  roles:
    - role: deploy-worker
      cryton_worker_runas_user: my-user
      cryton_worker_environment:
        CRYTON_WORKER_MODULES_DIR: "{{ cryton_modules_directory }}/modules"
        CRYTON_WORKER_RABBIT_HOST: 127.0.0.1
        CRYTON_WORKER_NAME: unique-name
```

Deploy CLI

```
- name: Deploy CLI
  hosts: client
  become: yes
  roles:
    - role: deploy-cli
      cryton_cli_runas_user: my-user
      cryton_cli_environment:
        CRYTON_CLI_API_HOST: 127.0.0.1
```

Register Worker

```
- name: Register Worker
  hosts: client
  roles:
    - role: register-worker
      cryton_cli_runas_user: my-user
      cryton_worker_name: unique-name
      cryton_worker_description: custom Worker description
      cryton_cli_environment:
        CRYTON_CLI_API_HOST: 127.0.0.1
```

Deploy CLI and register new Worker

```
- name: Deploy CLI and register Worker
  hosts: client
  become: yes
  vars:
    cryton_cli_runas_user: my-user
    cryton_cli_environment:
        CRYTON_CLI_API_HOST: 127.0.0.1
  roles:
    - role: deploy-cli
    - role: register-worker
      cryton_worker_name: unique-name
      cryton_worker_description: custom Worker description
```

Deploy frontend

```
- name: Deploy frontend
  hosts: client
  become: yes
  roles:
    - role: deploy-frontend
      cryton_frontend_environment:
        crytonRESTApiHost: 127.0.0.1
```

4. Getting started

4.1 Installation example (local deployment)

This example will walk you through the installation of all Cryton components (locally). However, we will only change the necessary settings, since this is primarily a showcase installation. For Cryton to work correctly, **be strict about the order of components installation to preserve dependencies**.

We will use **pipx** to install CLI and **Docker Compose** to install Core with Frontend.

Please make sure you are using a system that has at least 2048 MB of RAM and 2 CPU cores, otherwise you might experience stability issues.

4.1.1 Install prerequisites

The following packages might be missing on your system and are **necessary**:

- *python3-pip*
- *python3-venv*

Make sure you have installed and set up correctly the following tools:

- pipx (requires restarting terminal session)
- curl
- Docker Compose (requires system reboot)
- git

Also make sure the directory `~/.local/` exists, since we will be using it for the installation.

```
mkdir -p ~/.local/
```

4.1.2 Install CLI

More information can be found in the starting point.

First, we create an application directory and save settings into it. We will install the CLI afterward.

```
cd ~/.local/
mkdir -p ~/.local/cryton-cli/
curl -o ~/.local/cryton-cli/.env https://gitlab.ics.muni.cz/beast-public/cryton/cryton-cli/-/raw/2022.2.0/.env
pipx install cryton-cli
```

4.1.3 Install and run Frontend (optional)

More information can be found in the starting point.

First, we clone the repository and checkout the correct version. Finally, we install and start Cryton Core and all necessary services (RabbitMQ, Postgres, PgBouncer).

```
cd ~/.local/
git clone https://gitlab.ics.muni.cz/beast-public/cryton/cryton-frontend.git
cd cryton-frontend
git checkout 2022.2.0a0
sudo docker compose up -d --build
```

4.1.4 Install and run Core

More information can be found in the starting point.

First, we clone the repository and checkout the correct version. Finally, we install and start Cryton Core and all necessary services (RabbitMQ, Postgres, PgBouncer).

```
cd ~/.local/
git clone https://gitlab.ics.muni.cz/beast-public/cryton/cryton-core.git
cd cryton-core
git checkout 2022.2.1
sudo docker compose -f docker-compose.yml -f docker-compose.prerequisites.yml up -d --build
```

4.1.5 Test the installation

Now we want to test if the CLI and Core are communicating.

Open a new terminal Window, type in the following command and check if the Core sends an empty response:

```
cryton-cli workers list
```

4.2 Simple Workflow

4.2.1 Prerequisites

Before you start with this example, it is assumed that:

- Core is installed and running.
- CLI is already installed.
- Worker is already installed and running.
- Modules are mounted to Worker.
- Nmap and Medusa tools are installed and accessible from Worker.
- Core and CLI are installed on the same machine. If not, see the Cryton CLI settings or usage section to connect cryton-cli to cryton-core.

If you haven't installed the Cryton toolset already, feel free to follow the installation example.

4.2.2 Cryton attack workflow

The following is the ideal sequence of steps to use when planning an attack and using Cryton to automate it. First, you need to prepare an infrastructure for your cyber defense exercise. Once you have it, you can start planning your attack:

1. Create an attack plan template.
2. Install and set up Core, CLI/Frontend.
3. Set up your Workers (one for each team):
   - Download and mount the required modules for the plan.
   - Install module prerequisites (attack tools, system requirements).
   - Update the Worker settings and start it (Python dependencies can be installed on Worker startup).
4. Register Workers and create Run from the attack plan.
5. Execute or schedule the Run.

4.2.3 Create plan template

To execute our attack plan, we must create its template (YAML) first – a description of the actions required to run during attack execution based on tools used during the attack.

We will be using a basic template example, which can be found here as well as other examples.

```
---
# This is a simple example of Step chaining.
# Required modules: mod_cmd, mod_medusa.

plan:
  name: Basic example
  owner: Cryton
  stages:
    - name: get-localhost-credentials
      trigger_type: delta
      trigger_args:
        seconds: 0
      steps:
        - name: check-ssh
          is_init: true
          step_type: worker/execute
          arguments:
            module: mod_nmap
            module_arguments:
              target: {{ target }}
              ports:
                - 22
          next:
            - type: result
              value: OK
              step: bruteforce
```

```
        - name: bruteforce
          step_type: worker/execute
          arguments:
            module: mod_medusa
            module_arguments:
              target: {{ target }}
              credentials:
                username: {{ bruteforce.username }}
                password: {{ bruteforce.password }}
```

Once we are satisfied with our template, we can upload it using CLI:

```
cryton-cli plan-templates create path/to/template.yml
```

Example:

```
cryton-cli plan-templates create cryton-core/examples/basic-example/template.yml
Template successfully created! ({'id': 1})
```

Validating plan template

Before we upload the template, we should validate it. However, for our template to be validated correctly, we have to provide an inventory file, which is described here. Once we have it, we can simply run:

```
cryton-cli plans validate path/to/template.yml -i path/to/my/inventory-file.yml
```

Example:

```
cryton-cli plans validate cryton-core/examples/basic-example/template.yml -i cryton-core/examples/basic-example/inventory.yml
Plan successfully validated! (<response>)
```

4.2.4 Set up Worker(s)

Notice: If you've set up Cryton using the installation example, and you're not using it for production, you can skip this section.

Check if the Workers have the correct settings, mainly `CRYTON_WORKER_RABBIT_*`, `CRYTON_WORKER_NAME`, and `CRYTON_WORKER_MODULES_DIR`.

Use the `--install-requirements` (`-I`) flag to install the requirements (python dependencies) at startup.

If everything is set, you can start each worker:

```
cryton-worker start
```

If you want to run the Worker in the background use:

```
nohup cryton-worker start > ~/.local/cryton-worker/std_out 2>&1 &
```

Notice: If you're using Docker Compose to install and run Worker, and you've changed the settings, you have to restart the container.

At this point, our worker is active and awaiting messages from Core. However, to be able to use it, we have to register it in Core's database. We can do that using CLI (repeat for each Worker). Keep in mind that **WORKER_NAME** must be the same as Worker's `CRYTON_WORKER_NAME` variable:

```
cryton-cli workers create <WORKER_NAME> -d <WORKER_DESCRIPTION>
```

Example:

```
cryton-cli workers create unique-name -d "kali-worker1 on 192.168.56.101"
Worker successfully created! ({'id': 1})
```

To check if the Workers are running use a health check (repeat for each Worker):

```
cryton-cli workers health-check <WORKER_ID>
```

Example:

```
cryton-cli workers health-check 1
The Worker successfully checked! (<response>)
```

4.2.5 Create Plan instance

Now we need to create a Plan instance we will use for the execution. We can create it using the combination of the previously uploaded template and an inventory file.

Create the following inventory file:

```
---
target: 127.0.0.1
bruteforce:
  username: my_user
  password: my_pass
```

If you haven't uploaded the template, because you want to **validate it first**, see this section.

To create a new Plan instance use:

```
cryton-cli plans create <TEMPLATE_ID> -i path/to/my/inventory-file.yml
```

Example:

```
cryton-cli plans create 1 -i cryton-core/examples/basic-example/inventory.yml
Plan Instance successfully created! ({'id': 1})
```

4.2.6 Create Run

The last step we have to make is to create a new Run from the previously created Plan instance and Worker(s). To do so, use:

```
cryton-cli runs create <PLAN_INSTANCE_ID> <WORKER_ID1> <WORKER_ID2> <WORKER_ID3> ...
```

Example:

```
cryton-cli runs create 1 1 2 3
Run successfully created! ({'id': 1})
```

4.2.7 Execute Run

Now that everything is prepared, we can execute our Run immediately or schedule it for later. To execute the Run immediately, use:

```
cryton-cli runs execute <RUN_ID>
```

Example:

```
cryton-cli runs execute 1
Run successfully executed! (Run 1 was executed.)
```

4.2.8 Schedule Run

Run executions can be scheduled to a specific date and time. By default, the system timezone will be used. To use UTC timezone, use the `--utc-timezone` flag.

```
cryton-cli runs schedule <RUN_ID> <DATE> <TIME>
```

Example:

```
cryton-cli runs schedule 1 2020-06-08 10:00:00
Run successfully scheduled! (Run 1 is scheduled for 2020-06-08 10:00:00.)
```

4.2.9 Pause/Resume Run

In case you need to pause a Run in progress use:

```
cryton-cli runs pause <RUN_ID>
```

Example:

```
cryton-cli runs pause 1
Run successfully paused! (Run 1 was paused.)
```

Run will be paused after all the running Steps finish.

To resume the execution use:

```
cryton-cli runs resume <RUN_ID>
```

Example:

```
cryton-cli runs resume 1
Run successfully resumed! (Run 1 was resumed.)
```

4.2.10 Report results

It is crucial to know the current state of your Run and its results. That is why a report can be generated anytime during the execution.

Show Run information

To see if the executed Run has finished, you can check its state (and other useful information):

```
cryton-cli runs show <RUN_ID>
```

Example:

```
cryton-cli runs show 1
id: 1, schedule_time: None, start_time: 2021-05-24T00:08:45.200025, pause_time: None, finish_time: 2021-05-24T00:09:18.397199, state: RUNNING
```

View report

Reports can be viewed directly in cryton-cli (**to quit, press Q**):

```
cryton-cli runs report <RUN_ID> --less
```

Example:

```
cryton-cli runs report 1 --less
```

Generate report

You can also save the report into a file:

```
cryton-cli runs report <RUN_ID>
```

Example:

```
cryton-cli runs report 1
Successfully created Run's report! (file saved at: /tmp/report_run_1_2020-06-08-10-15-00-257994_xdQeV)
```

5. Designing phase

5.1 What is an attack scenario

Let's start with the description of the attack scenario. I use attack scenario and Plan interchangeably - Plan is just the name of the element in the formal description of the attack scenario.

An attack scenario is a sequence of steps with some common objective. This objective may be data exfiltration, access to target systems, denial of service, or any other harmful action. For some exercises, every attack should be divisible into different stages. Imagine you have to attack infrastructure with multiple machines - each machine can be a separate stage. Or you want to attack according to some kill-chain, e.g. the first stage would be scanning of the infrastructure, the second is brute force attack on credentials to found systems, etc.

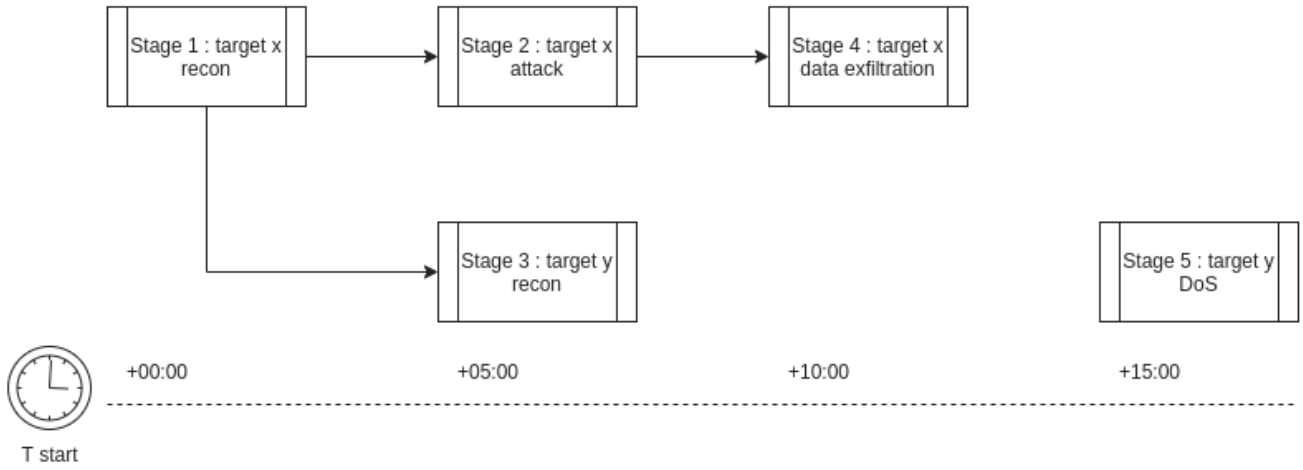
The last and most important element of the Plan description is the attack step. This is the execution of an attack script or tool against the target. A step can be running a Metasploit exploit, or running a Nmap scan. Steps are dependent on each other, and so they create an execution tree, where each of them has set the list of successors based on some condition. The condition may be a success or a string value returned by its predecessor.

Putting this all together you get the whole attack scenario (called **Plan**), which is divided into different stages (called **Stage**). Every stage is set to run at a specific time, as this is often required by the exercise. And finally, each stage consists of attack steps (called **Step**), which are organized in a non-binary tree described above.

5.2 Template

5.2.1 What is a template

The most important part is creating a plan template. This is basically a Plan object written in YAML format. You can find a detailed description of this format in section Plan.



Template itself is not a fully described attack scenario. The structure of the Attack (execution tree) is correct, but there are still unfilled places (e.g. IP addresses of targets or some other inventory variables). This way an Attack Template can be designed before knowing these details and used in multiple different environments.

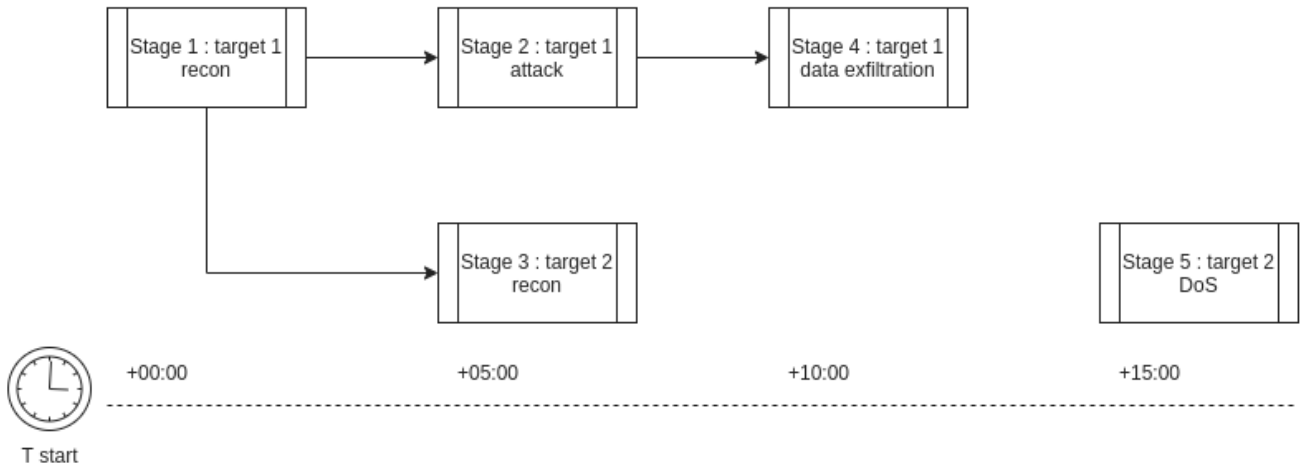
The first step in designing a Plan (attack scenario) is creating a template. In this step the user is supposed to write the whole scenario in the form of Plan, Stages and Steps.

An abstract Plan can look like this:

```
Plan
  Stage 1
    Step 1
    Step 2
  Stage 2
    Step 3
```

5.2.2 Plan

Plan is the basic unit of an attack scenario. It contains the name and owner of the Plan and a list of Stages.



Example of defining a Plan using YAML:

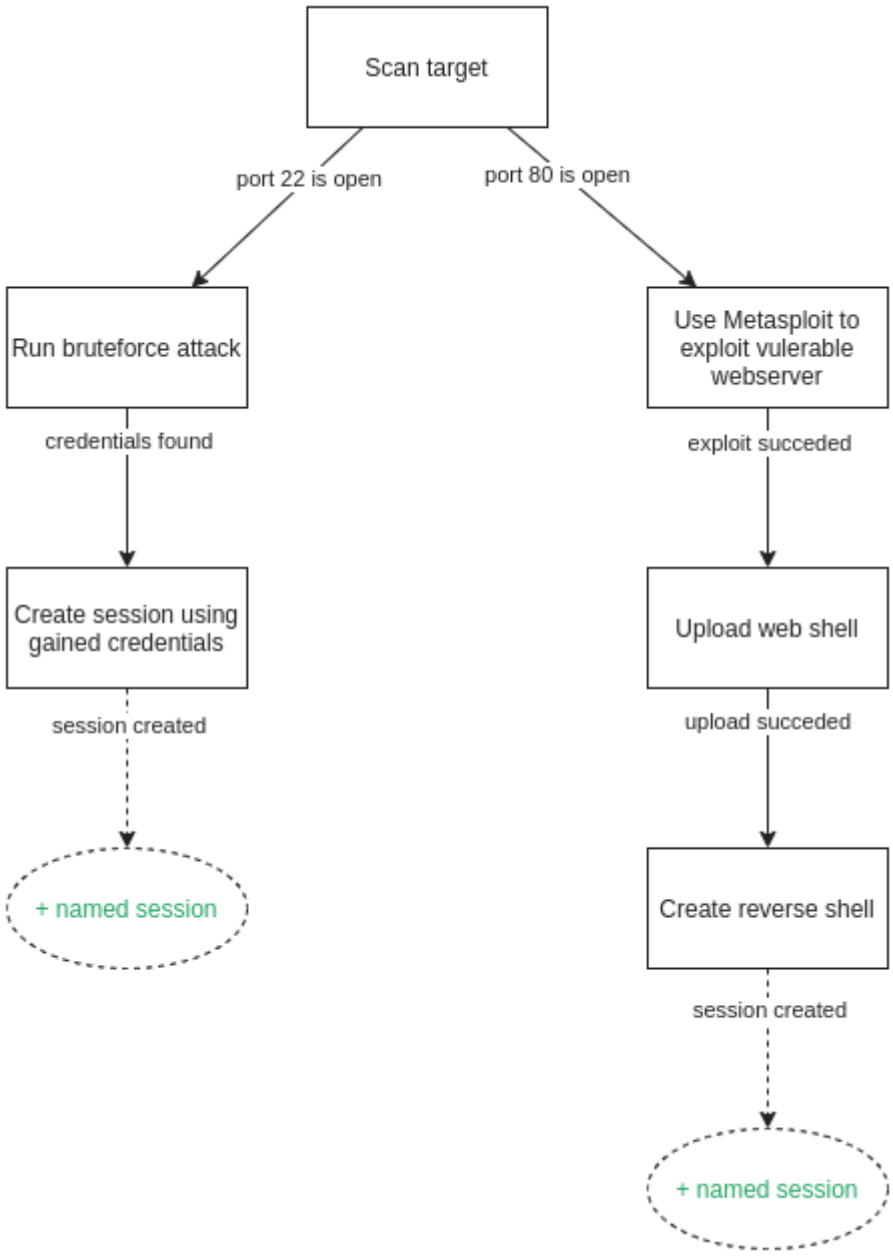
```
plan:
  name: my-plan
  owner: my name
  stages:
    ...
```

To better understand what each argument means and defines, here is a short description:

- **name** - Sets the name of the Plan.
- **owner** - Name of the person who created the Plan.
- **stages** - List of Stages that will be executed during the Plan's execution.

5.2.3 Stage

A stage is a unit defined by a target and its trigger (for example time of start). It contains a list of attack Steps that are related to each other.



Example of defining Stage using YAML:

```
name: my-stage
trigger_type: delta
trigger_args:
minutes: 5
steps:
  ...
depends_on:
  - previous-stage
```

To better understand what each argument means and defines, here is a short description:

- **name** - Sets the name of the Stage, which is mainly used to define its purpose (**must be unique** across the Plan).
- **trigger_type** - Type of the trigger to be used. For more details see triggers.
- **trigger_args** - Dictionary arguments, that are specific for each type of trigger. For more details see triggers.
- **steps** - List of related Steps that will be executed during the Stage's execution.
- **depends_on** - If the Stage depends on other Stages' actions, we can tell the Stage to wait until the other Stages are finished. For more details see dependencies.

Triggers

DELTA

Schedule execution for a specific time after the plan start, e.g. `minutes: 30`.

Examples

```
trigger_type: delta
trigger_args:
  hours: 0  # Wait for x hours.
  minutes: 0  # Wait for x minutes.
  seconds: 0  # Wait for x seconds.
```

DATETIME

Schedule execution for a specific date and time after the plan started.

Trigger arguments:

Notice: One argument from the Date and Time part of arguments (every argument except `timezone`) is required. Besides that, arguments are optional and their default values are used in their absence.

Argument	Description	Default
timezone	Timezone for DateTime trigger. List of available timezones here.	UTC
year	Year in which stage should be executed.	The year of the plan execution in the specified timezone
month	Month in which stage should be executed.	The month of the plan execution in the specified timezone
day	Day in which stage should be executed.	The day of the plan execution in the specified timezone
hour	Hour in which stage should be executed.	00
minute	Minute in which stage should be executed.	00
second	Second in which stage should be executed.	00

Examples

```
# This stage would be executed on 2022-01-01 08:20:00 in Europe/Prague timezone
trigger_type: datetime
trigger_args:
  timezone: Europe/Prague
  year: 2022
  month: 1
  day: 1
  hour: 8
  minute: 20
  second: 00
```

```
# This stage would be executed at 16:00 UTC on the day of the plan execution
trigger_type: datetime
trigger_args:
  hour: 16
```

HTTP LISTENER

The stage will be executed on specific data received in the HTTP request (GET/POST) on the listener.

```
trigger_type: HTTPListener
trigger_args:
  host: localhost  # Address of the listener from the Worker's perspective.
  port: 8082  # Port of the listener from the Worker's perspective.
  routes:  # List of routes listener will check for requests.
    - path: /index  # Request's path.
      method: GET   # Request's allowed method.
      parameters:  # Request's required parameters.
        - name: parameter  # Parameter's name.
          value: value  # Parameter's value.
```

MSF LISTENER

The stage will be executed when a session with the user-defined arguments is returned from Worker. Once the session is saved, it can be accessed using `use_named_session: my-stage-name_session`, where `my-stage-name` is the Stage's name.

Identifiers are arguments that can be used to identify a msf session that we are waiting for to trigger the Stage. Arguments in identifiers use partial 'regex' to get matches. For example, they can use 'handler' to match 'exploit/multi/handler'.

```
trigger_type: MSFListener
trigger_args:
  identifiers: # Optional, by default MsfTrigger will try to find a match using 'via_exploit' and 'via_payload' based on used msf module and payload
    type: 'shell'
    tunnel_local: '192.168.56.50:4444'
    tunnel_peer: '192.168.56.51:35380'
    via_exploit: 'exploit/unix/irc/unreal_ircd_3281_backdoor'
    via_payload: 'payload/cmd/unix/reverse_perl'
    desc: 'Command shell'
    info: ''
    workspace: 'false'
    session_host: '192.168.56.50'
    session_port: '4444'
    target_host: ''
    username: 'vagrant'
    uuid: 'o3mnfksh'
    exploit_uuid: 'vkzl8sib'
    routes: ''
    arch: 'python'
  exploit: unix/irc/unreal_ircd_3281_backdoor  # Exploit to use.
  exploit_arguments:  # Arguments that will be passed to the exploit.
    RHOST: 192.168.56.51
    RPORT: 6697
  payload: cmd/unix/reverse_perl  # Payload to use.
  payload_arguments:  # Arguments that will be passed to payload.
    LHOST: 192.168.56.50
    LPORT: 4444
```

Dependencies

Creating time based triggers can be limiting, since the Stage itself can take more time than expected. To ensure that the Stages will execute in the correct order, you can choose to check if some other Stage has already finished, before its execution. All you have to do is define the `depends_on` argument.

This way you can ensure that the sessions and gathered output from other Stages are available.

```
name: stage-name
depends_on:
- other-stage
```

5.2.4 Step

As the name suggests, a Step is equal to one action. All the possible actions will be described later. Every step can have a successor(s) which execution will follow according to provided conditions.

Example of defining Step using YAML:

```
name: get-credentials
step_type: worker/execute
is_init: true
output_prefix: credentials_from_localhost
arguments:
  module: mod_medusa
  module_arguments:
    target: localhost
    credentials:
      username: admin
      password: admin
next:
  - type: result
    value: OK
    step: create-sesion
```

To better understand what each argument means and defines, here is a short description (sub-arguments are omitted since they will be discussed in more depth in their own section):

- **name** - Sets the name of the Step, which is mainly used to define its purpose (**must be unique** across the Plan).
- **step_type** - Sets what action will the Step perform and what `arguments` will the Step use, more info below.
- **is_init** - Defines if the step is initial (is executed first) and is not a successor.
- **output_prefix** - If you want to use a custom name for sharing Step's results (*serialized_output*) you can define this parameter. By default, the Step's *name* is used. For more details see Output prefix.
- **arguments** - Dictionary of arguments different for each *step_type*. To check out all possible parameters and types see types section.
- **next** - Defines Step's successors, more info below.

Step types

Step types are represented by the mandatory `step_type` parameter which defines what action should be executed in Step. It tells the Worker component what arguments to expect and what functions to run based on them.

Currently, there are 3 types:

Step type	Purpose
`worker/execute`	Execution of attack modules.
`empire/agent-deploy`	Deployment of Empire agent on target.
`empire/execute`	Execution of shell commands or Empire modules on active Empire agent.

Execute attack module on Worker

This functionality uses `step_type: worker/execute` and enables the execution of an attack module on a worker defined by the following parameters.

Argument	Description
`module`	Defines a path (will be added to the path defined in Worker) to the chosen module that will be executed on Worker.
`module_arguments`	Python dictionary (JSON) containing arguments that will be passed to the module.
`create_named_session` (optional)	How to name the session this module will create for later usage.
`use_named_session` (optional)	Name of created msf session through Cryton.
`use_any_session_to_target` (optional)	Ip address of target on which has been created msf session.
`session_id` (optional)	ID of msf session to target.
`ssh_connection` (optional)	Arguments for creating ssh connection to target.

Execution variables can be used only for the `module_arguments` parameter!

ARGUMENTS FOR `SSH_CONNECTION`

Argument	Description
`target`	Ip address for ssh connection.
`username` (optional)	Username for ssh connection.
`password` (optional)	Password for ssh connection if `ssh_key` is not supplied.
`ssh_key` (optional)	Ssh key for ssh connection if `password` is not supplied.
`port` (optional)	Port for ssh connection (default: 22).

Conditional execution

To be able to execute an attack scenario according to some execution tree, Steps provide a way to be executed according to specified conditions. There are many types of conditions that can be used. To use them in designing a Template, a list of dictionaries **containing** params **type**, **value**, **step** must be provided.

Following are types of conditions together with descriptions of possible values.

Type	Value	Description
`result`	OK, FAIL, EXCEPTION	Match final `result` of the Step.
`serialized_output`	Regex expression, for example: `^my_regex_expression.*`	Match regex expression in `serialized_output` of the Step.
`output`	Regex expression, for example: `^my_regex_expression.*`	Match regex expression in `output` of the Step.
`any`	- (value must be omitted)	Run successor(s) in any case.

TYPE

Which value you want to compare, according to the output of the parent Step.

VALUE

The desired value of the selected type. Can be defined as a string (one value) or a list of strings (multiple values).

STEP

Name of the Step's successor(s).
Can be a string (one successor) or a list of strings (multiple successors).

EXAMPLES:

```
next:
- type: result
  value: OK
  step: step-to-execute
```

```
next:
- type: serialized_output
  value:
    - admin
    - root
  step: step-to-execute
```

```
next:
- type: any
  step:
    - step-to-execute-1
    - step-to-execute-2
```

Output sharing

Output sharing is used for sharing gained data (*serialized_output*) between multiple steps. To go through the data we use a modified version of a dot notation.

For example imagine following dictionary (Python data structure)

```
{"credentials": [{"username": "admin", "password": "securePassword"}]}
```

If we wanted to access it and get **password** for *admin* user using our version of dot notation, we would use `credentials[0].password` which would return *securePassword* string.

This brings in some limitations:

- keys are separated using `.`
- key can't be in format `[integer]` (regex representation: `^\[[0-9]+]$`) as it represents list (array) index
- list (array) index can be defined multiple times in the same key for example `myKey[1][1]`, however must be defined at its end (regex representation: `((\[[0-9]+])+$)`)

There are two techniques for sharing outputs of modules between steps:

- **output_prefix**
- **output_mapping**

OUTPUT PREFIX

By default, the prefix string is set to the name of the step. Using its name, any other step can query its output (serialized_output of its attack module execution) and use it in its arguments.

Alternatively, this prefix can be set to a custom string. This way, multiple equivalent steps can return the same prefixed variable value to be used by a dependent step. For example:

```
- name: bruteforce
  step_type: worker/execute
  is_init: true
  output_prefix: custom_prefix
```

```
  arguments:
    module: mod_medusa
    # Should return username and password in a dictionary
    module_arguments:
      target: localhost
      # Default password list in mod_medusa folder will be used for bruteforce
      credentials:
        username: admin
  next:
  - type: result
    value: OK
    step: mod_ssh
- name: mod_ssh
  step_type: worker/execute
  arguments:
    module: mod_ssh
    module_arguments:
      username: $custom_prefix.username
      password: $custom_prefix.password
```

Also, there is a special prefix named **parent**, which simply takes the output from parent step execution (which executed the current step).

```
- name: stepA
  ...
  next:
  - type: ...
    value: ...
    step: stepB
- name: stepB
  ...
  module_arguments:
    username: $parent.var
```

Output prefix **cannot be the name of other steps or the value "parent"**, otherwise it can be any string **that doesn't contain "$" and "." signs**.

OUTPUT MAPPING

Sometimes you do not care from which module you receive information. Step A and Step B can both return a stolen authentication token. For this reason, you can use `output_prefix`. But there is an obvious problem! What if both steps return this value under a different name, e.g. `token` and `auth_token`? Prefix would not help you much in this situation. For this reason there is a `output_mapping` mechanism.

```
- name: step_a
  # Returns 'token'
  output_prefix: steal
  output_mapping:
    - name_from: token
      name_to: stolen_token
  step_type: worker/execute
  arguments:
    module: mod_a
    module_arguments:
      ...
- name: step_b
  # Returns 'auth_token'
  output_prefix: steal
  output_mapping:
    - name_from: auth_token
      name_to: stolen_token
  step_type: worker/execute
  arguments:
    module: mod_b
    module_arguments:
      ...
- name: step_c
  step_type: worker/execute
  arguments:
    module: mod_c
    module_arguments:
      token: $steal.stolen_token
```

Execution variables

To assign different values for each Plan execution in Run, you can use execution variables.
To define execution variables, use Jinja (with some tweaks) instead of filling the arguments with real values while creating a Plan template. For example:

```
module_arguments:
  target: '{{ my_jinja_variable }}'
```

IMPORTANT: Execution variables must be defined as a string using single quotes, otherwise they won't be matched.

And before you execute Run (and its Plan execution(s)), upload your variable(s). (see CLI)

Example of a file with execution variables:

```
variable: localhost
nested:
  variable: value
variable_list:
  - var1
  - var2
```

If a variable cannot be filled, the Step errors out.

LIMITATIONS

- Execution variables must be defined as a string using single quotes. `yaml`
  `foo: '{{ variable }}'`
- In the case of Step type `worker/execute` you can use these variables only for the `module_arguments` parameter and its sub-parameters. For `empire/agent-deploy` or `empire/execute` you can use these variables for the root `arguments` parameter and its sub-parameters.
- Currently, there is support for simple and nested variables only. Examples: `yaml`
  `foo: '{{ variable }}'`
  `foo: '{{ nested.variable }}'`
  `foo: '{{ variable[index] }}'`
  `foo: '{{ nested.variable[index] }}'`
- If you want to use more Jinja goodies, use the raw block: `yaml`
  `foo: {% raw %} '{{ variable + 14 }}' {% endraw %}`

5.3 Plan instance

The second stage is creating a Plan instance. While Template contains unfilled variables (therefore the name "template"), Plan instance fills these things in by combining the template with an **inventory file**. This file contains all information that needs to be filled in the template. After instantiation, everything is ready to create a **Run**.

NOTE: After creating the Plan instance only the Execution variables can be left unfilled and must be explicitly defined as a string.

5.3.1 Inventory files

When you create a template, you don't always have all the information you need for directly executing it. Or you simply want to make it reusable for other people in their own environment. To provide variability in templates we support **inventory files**. These inventory files can be used to provide variable values to templates using **Jinja** language.

A valid Plan file is written in YAML format with variables in the Jinja format, which have to be replaced during the instantiation process.

Example of inventory file:

```
names:
  stage1_target: 192.168.56.102
```

You can use it as in following example:

```
  # Stage two: target is web server
  - target: {{names.stage1_target}}
```

5.4 Session management

One of the unique features of Cryton is the ability to create and use *sessions* - connections to the target systems. When you successfully exploit a running network service on your target machine (victim), you open a connection to it. In Cryton, this connection can be given a name and then used during the Plan execution in any Step (which is executed on the same Worker node and supports this functionality). Metasploit Framework session management is used for storing and interacting with sessions, and therefore must be available and running on the Worker node.

```
- name: step1
  arguments:
    create_named_session: session_to_target_1
    ...
- name: step2
  arguments:
    use_named_session: session_to_target_1
    ...
```

In the example above, step1 creates a named session *session_to_target_1* (in case it succeeds). Its Metasploit ID gets stored in the database and can be used anywhere in the Plan, not only in the following Step (as seen in the example). When the Plan creates multiple sessions to the same target, and the attacker does not care which he is using, the *use_any_session_to_target* parameter can be used.

```
- name: step1
  arguments:
    use_any_session_to_target: 192.168.56.22
    ...
```

5.4.1 Plan example

```
---
plan:
  name: Session example
  owner: test name
  stages:
  - name: stage-one
    trigger_type: delta
    trigger_args:
      seconds: 5
    steps:
    - name: ssh-session
      is_init: true
      step_type: worker/execute
      arguments:
        create_named_session: session_to_target_1
        module: mod_msf
        module_arguments:
          exploit: auxiliary/scanner/ssh/ssh_login
          exploit_arguments:
            RHOSTS: 127.0.0.1
            USERNAME: vagrant
            PASSWORD: vagrant
      next:
        - type: result
          value: OK
          step: session-cmd
    - name: session-cmd
      step_type: worker/execute
      arguments:
        use_named_session: session_to_target_1
        module: mod_cmd
        module_arguments:
          cmd: cat /etc/passwd
```

6. Execution phase

6.1 What is Run

When you finally have a Plan instance stored in the database, you can create a new Run. For every Worker, a new Plan execution is created. The reason behind this is a storage of execution history, so it is possible to compare results in Run and usage of different variables for Plan executions. As mentioned before, you can leave Step argument as an empty Jinja variable. Now, before execution, you can use different values for each Plan execution.

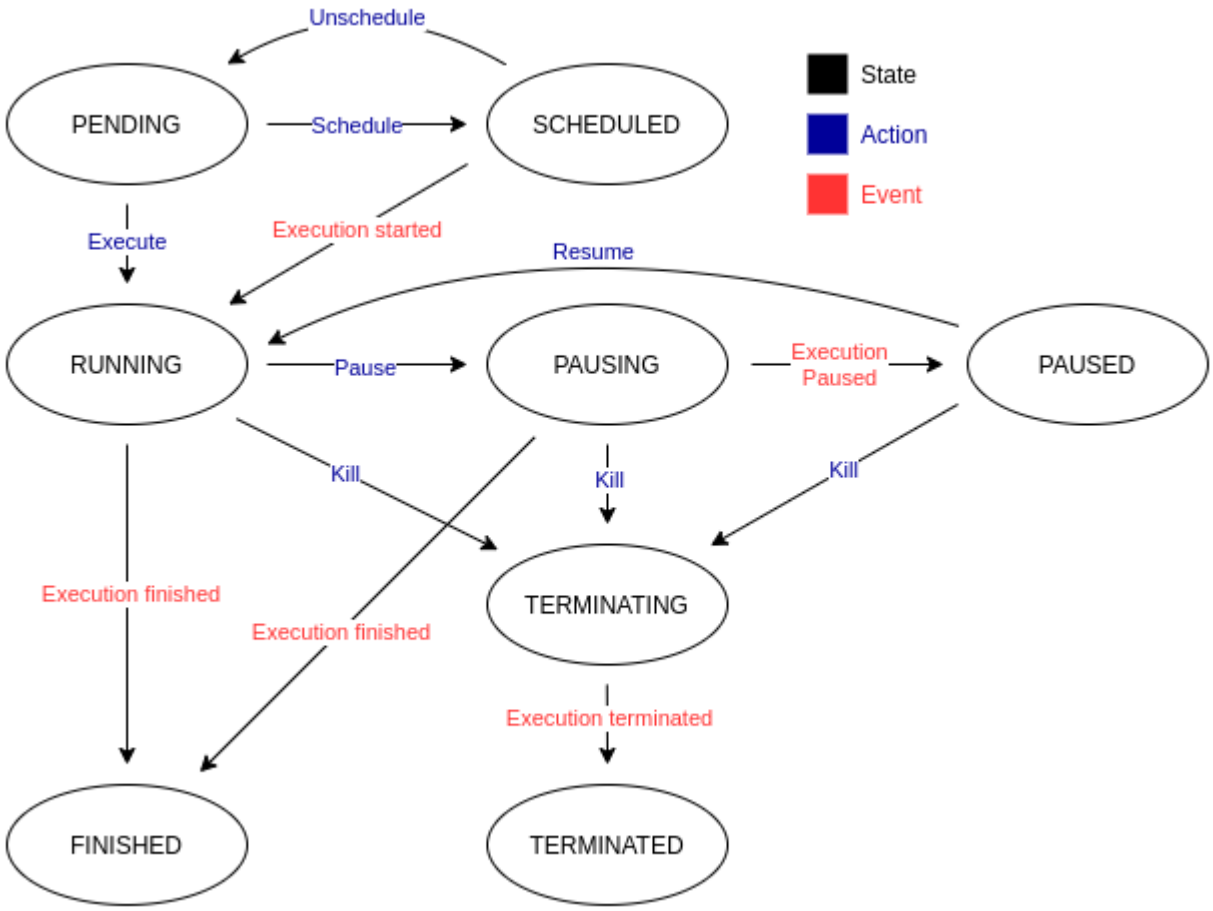
6.1.1 Parameters

The following table contains a list of output parameters.

Name	Description	Type	Example
state	Current state of the execution.	string	PENDING
start_time	When the execution started.	datetime	2022-07-21T20:37:28.343619Z
pause_time	Time of the last pause.	datetime	2022-07-21T20:37:28.343619Z
finish_time	When the execution finished.	datetime	2022-07-21T20:37:28.343619Z
schedule_time	When is the execution supposed to start.	datetime	2022-07-21T20:37:28.343619Z
aps_job_id	ID of the job in scheduler.	string	abcd-1d2c-abcd-1d2c
plan	Which Plan is used for the execution.	int	1

6.1.2 States

Here is a map of allowed states, transitions, and their description.



PENDING - Every execution starts its lifecycle in this state - it is inactive.

SCHEDULED - Execution with this state will be started at the defined time.

RUNNING - Execution is in progress and its sub-executions are being executed.

PAUSING - If the user decides to pause the execution, it will change its state to *PAUSING* and wait until the conditions are met.

PAUSED - Once all the sub-executions are paused or in a final state, the execution is marked as *PAUSED*.

TERMINATING - If the user decides to kill the execution, it will change its state to *TERMINATING* and wait until the conditions are met.

TERMINATED - Once all the sub-executions are stopped, the execution is marked as *TERMINATED*.

FINISHED - In this state the execution, and its sub-executions have reached final states.

6.1.3 Actions

To give you the ability to control your attack plan, here is a list of the supported actions.

Execute

The basic action you can do is **Execute** the Run. This will start it instantly. What does that mean?

- Delta and DateTime triggered Stages are scheduled
- HTTP/MSF triggers are started on Workers
- Run state is set to RUNNING

Schedule, Reschedule, Unschedule, Postpone

Schedule action does just that - it **schedules** an *Execute* action at a given time. If you want to change the time of your execution, you can still **reschedule** it or simply **unschedule** it and leave it in a pending state. You can also **postpone** your scheduled execution.

Pause, Unpause

Anytime during Run execution, you can **pause** it. To continue the Run, simply issue the **unpause** command.

Kill

Terminate (forcefully) Run's execution and its sub-executions.

Validate modules

Check if all the modules used in the Run are present on desired Workers and if their arguments set in the Plan are correct.

6.2 Execution statistics

There are special kinds of objects called *Executions* for every unit in the attack scenario:

- **Plan execution**
- **Stage execution**
- **Step execution**

These are not objects that need to be created by you - instead, they are created when their respective unit is executed. This way the history and results of each execution can be stored.

Every Execution object stores a start and finish time, so it is easy to count the running times of each unit. With Steps the Execution is also a place where the output and results from attack modules are stored.

6.2.1 Plan execution

For every execution of the Plan (on a given Worker) a new Plan execution is created.

The importance of this object is in keeping the history of executions of a given run. This object also connects all other sub-executions (Stage and Step) with the Run.

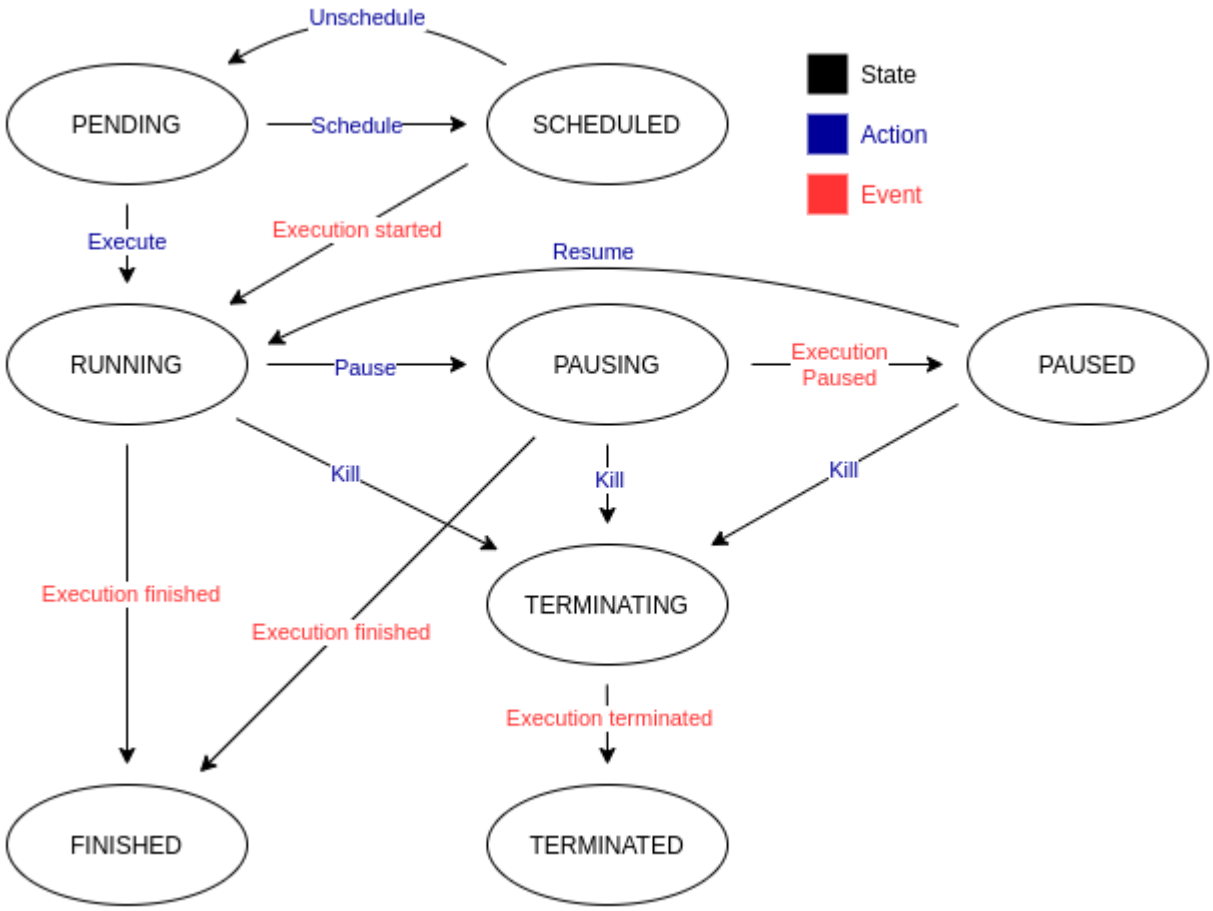
Parameters

The following table contains a list of output parameters.

Name	Description	Type	Example
state	Current state of the execution.	string	PENDING
start_time	When the execution started.	datetime	2022-07-21T20:37:28.343619Z
pause_time	Time of the last pause.	datetime	2022-07-21T20:37:28.343619Z
finish_time	When the execution finished.	datetime	2022-07-21T20:37:28.343619Z
schedule_time	When is the execution supposed to start.	datetime	2022-07-21T20:37:28.343619Z
aps_job_id	ID of the job in scheduler.	string	abcd-1d2c-abcd-1d2c
run	Run of which it is a part of.	int	1
worker	Which Worker is used for the execution.	int	1
evidence_directory	In what directory is the evidence saved.	string	/path/to/evidence/directory

States

Here is a map of allowed states, transitions, and their description.



PENDING - Every execution starts its lifecycle in this state - it is inactive.

SCHEDULED - Execution with this state will be started at the defined time.

RUNNING - Execution is in progress and its sub-executions are being executed.

PAUSING - If the user decides to pause the execution, it will change its state to *PAUSING* and wait until the conditions are met.

PAUSED - Once all the sub-executions are paused or in a final state, the execution is marked as *PAUSED*.

TERMINATING - If the user decides to kill the execution, it will change its state to *TERMINATING* and wait until the conditions are met.

TERMINATED - Once all the sub-executions are stopped, the execution is marked as *TERMINATED*.

FINISHED - In this state the execution, and its sub-executions have reached final states.

6.2.2 Stage execution

Stage execution contains execution data for its Stage counterpart.

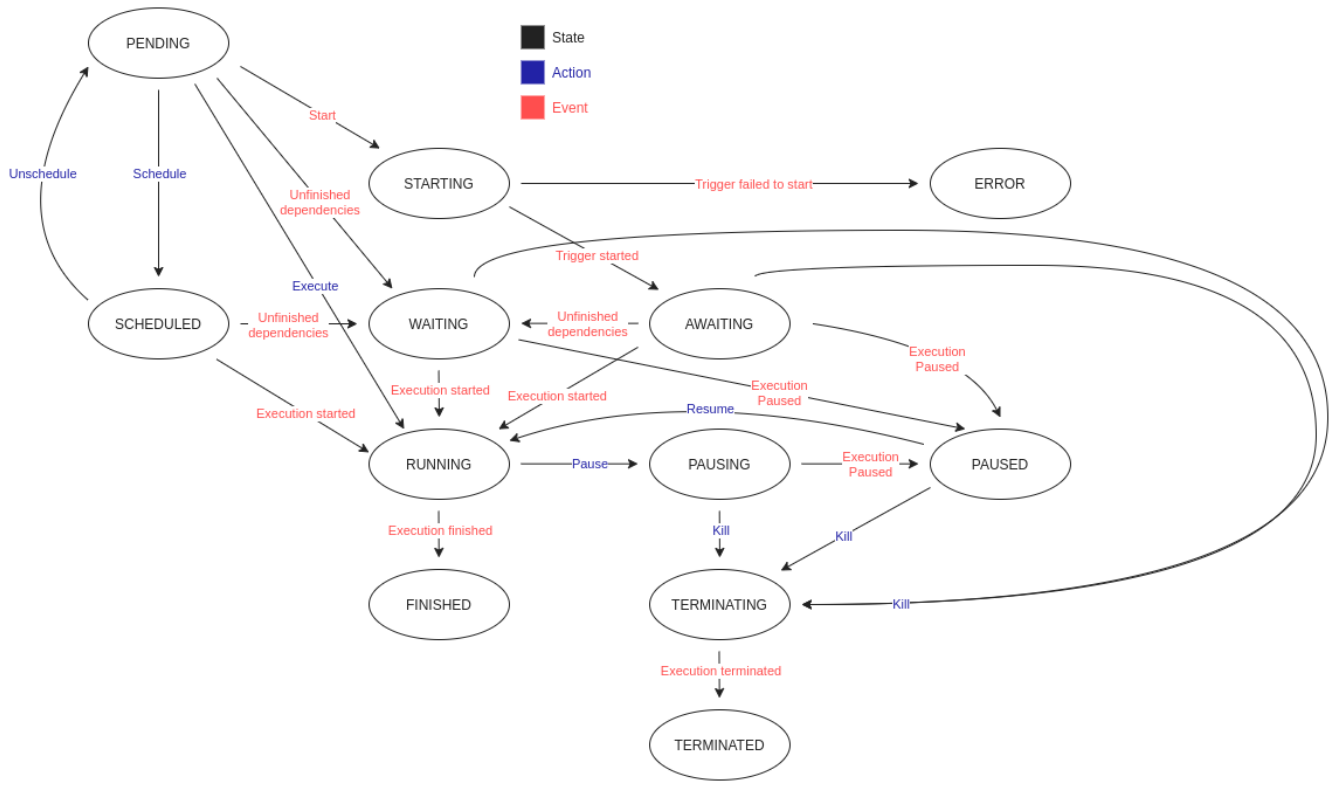
Parameters

The following table contains a list of output parameters.

Name	Description	Type	Example
state	Current state of the execution.	string	PENDING
start_time	When the execution started.	datetime	2022-07-21T20:37:28.343619Z
pause_time	Time of the last pause.	datetime	2022-07-21T20:37:28.343619Z
finish_time	When the execution finished.	datetime	2022-07-21T20:37:28.343619Z
schedule_time	When is the execution supposed to start.	datetime	2022-07-21T20:37:28.343619Z
aps_job_id	ID of the job in scheduler.	string	abcd-1d2c-abcd-1d2c
trigger_id	ID of the trigger on Worker.	string	abcd-1d2c-abcd-1d2c
plan_execution	Plan execution of which it is a part of.	int	1

States

Here is a map of allowed states, transitions, and their description.



PENDING - Every execution starts its lifecycle in this state - it is inactive.

SCHEDULED - Execution with this state will be started at the defined time.

STARTING - Action that requires Worker confirmation is occurring - starting listener.

ERROR - An error occurred during execution start up - unable to start listener.

WAITING - Execution in this state is waiting for its dependencies (other executions) to finish.

AWAITING - Execution is awaiting a trigger activation.

RUNNING - Execution is in progress and its sub-executions are being executed.

PAUSING - If the user decides to pause the execution, it will change its state to *PAUSING* and wait until the conditions are met.

PAUSED - Once all the sub-executions are paused or in a final state, the execution is marked as *PAUSED*.

TERMINATING - If the user decides to kill the execution, it will change its state to *TERMINATING* and wait until the conditions are met.

TERMINATED - Once all the sub-executions are stopped, the execution is marked as *TERMINATED*.

FINISHED - In this state the execution, and its sub-executions have reached final states.

6.2.3 Step execution

Step execution contains execution data for its Step counterpart.

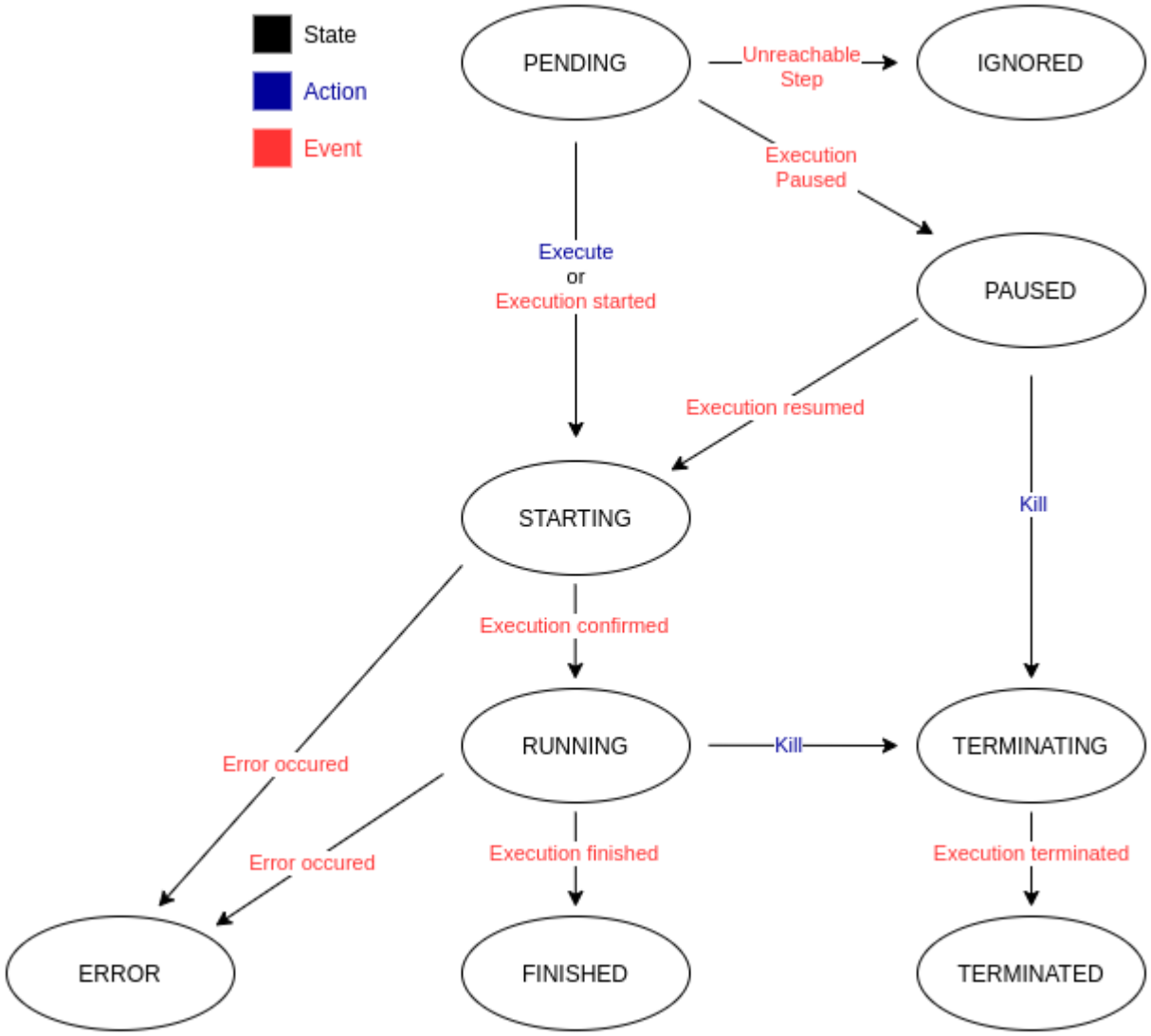
Parameters

The following table contains a list of output parameters.

Name	Description	Type	Example
state	Current state of the execution.	string	PENDING
start_time	When the execution started.	datetime	2022-07-21T20:37:28.343619Z
pause_time	Time of the last pause.	datetime	2022-07-21T20:37:28.343619Z
finish_time	When the execution finished.	datetime	2022-07-21T20:37:28.343619Z
result	Result of the module execution.	string	OK
output	Received output from the module execution.	string	created session with id 1.
serialized_output	Serializable output from the module execution.	dictionary	{"session_id": 1}
valid	Whether the parameters passed to the module are valid or not.	boolean	true
stage_execution	Stage execution of which it is a part of.	int	1

States

Here is a map of allowed states, transitions, and their description.



PENDING - Every execution starts its lifecycle in this state - it is inactive.

STARTING - Action that requires Worker confirmation is occurring - starting the execution.

RUNNING - Module execution is in progress.

PAUSED - Execution is marked as *PAUSED* if its Stage execution is pausing and only if its about to be executed next.

TERMINATING - If the user decides to kill the execution, it will change its state to *TERMINATING* and wait until the conditions are met.

TERMINATED - Once the module execution is stopped, the execution is marked as *TERMINATED*.

FINISHED - In this state the execution has successfully *FINISHED*.

ERROR - An unexpected error occurred during the execution.

IGNORED - The conditions to start the execution weren't met.

6.3 Reporting

After the Run has successfully ended (or not) you can **generate a report** with every Step's output and result. When you have multiple Plan executions in a single Run (when utilizing multiple Workers), you can compare each execution and use this insight to e.g. score each team in a *cybersecurity exercise.*

The easiest way to generate a report is to use CLI:

```
cryton-cli runs report <ID>
```

Optionally, you can also generate a report for Plan/Stage/Step execution.

You can see an example report here:


```
id: 7
plan_id: 6
plan_name: Basic example
state: FINISHED
schedule_time: null
start_time: '2022-07-21T20:37:27.650142Z'
pause_time: null
finish_time: '2022-07-21T20:37:28.527673Z'
plan_executions:
- id: 7
  plan_name: Basic example
  state: FINISHED
  schedule_time: null
  start_time: '2022-07-21T20:37:27.661100Z'
  finish_time: '2022-07-21T20:37:28.517554Z'
  pause_time: null
  worker_id: 1
  worker_name: e2e-1
  evidence_dir: /tmp/run_7/worker_e2e-1
  stage_executions:
  - id: 7
    stage_name: get-localhost-credentials
    state: FINISHED
    start_time: '2022-07-21T20:37:27.862354Z'
    pause_time: null
    finish_time: '2022-07-21T20:37:28.504804Z'
    schedule_time: '2022-07-21T20:37:27.762581Z'
    step_executions:
    - id: 11
      step_name: check-ssh
      state: FINISHED
      start_time: '2022-07-21T20:37:27.898861Z'
      finish_time: '2022-07-21T20:37:28.276521Z'
      output: ''
      serialized_output:
        stats:
          args: /usr/bin/nmap -oX - -sV -p22 192.168.61.13
          start: '1658386108'
          scanner: nmap
          version: '7.80'
          startstr: Thu Jul 21 06:48:28 2022
          xmloutputversion: '1.04'
        runtime:
          exit: success
          time: '1658386109'
          elapsed: '0.31'
          summary: Nmap done at Thu Jul 21 06:48:29 2022; 1 IP address (1 host up)
            scanned in 0.31 seconds
          timestr: Thu Jul 21 06:48:29 2022
        192.168.61.13:
          ports:
          - cpe:
            - cpe: cpe:/a:openbsd:openssh:8.4p1
            - cpe: cpe:/o:linux:linux_kernel
            state: open
            portid: '22'
            reason: syn-ack
            scripts: []
            service:
              conf: '10'
              name: ssh
              method: probed
              ostype: Linux
              product: OpenSSH
              version: 8.4p1 Debian 5
              extrainfo: protocol 2.0
            protocol: tcp
            reason_ttl: '64'
          state:
            state: up
```

```
            reason: arp-response
            reason_ttl: '0'
          osmatch: {}
          hostname:
          - name: 192.168.61.13
            type: PTR
          macaddress:
            addr: 08:00:27:D4:BF:9E
            vendor: Oracle VirtualBox virtual NIC
            addrtype: mac
      evidence_file: 'No evidence '
      result: OK
      valid: false
    - id: 12
      step_name: bruteforce
      state: FINISHED
      start_time: '2022-07-21T20:37:28.343619Z'
      finish_time: '2022-07-21T20:37:28.479002Z'
      output: ''
      serialized_output:
        password: victim
        username: victim
        all_credentials:
        - password: victim
          username: victim
      evidence_file: 'No evidence '
      result: OK
      valid: false
```

7. Interfaces

7.1 CLI

CLI implements capabilities of the Cryton's REST API and can be automated by using custom scripts.

To start the CLI just type `cryton-cli` and the following help page should show:

```
Usage: cryton-cli [OPTIONS] COMMAND [ARGS]...

  A CLI wrapper for Cryton API.

Options:
  -H, --host TEXT     Set Cryton's address (default is localhost).
  -p, --port INTEGER  Set Cryton's address (default is 8000).
  --secure            Set if HTTPS will be used.
  --debug             Show non formatted output.
  --version           Show the version and exit.
  --help              Show this message and exit.

Commands:
  execution-variables  Manage Execution variables from here.
  logs                 Manage Workers from here.
  plan-executions      Manage Plan's executions from here.
  plan-templates       Manage Plan templates from here.
  plans                Manage Plans from here.
  runs                 Manage Runs from here.
  stage-executions     Manage Stage's executions from here.
  stages               Manage Stages from here.
  step-executions      Manage Step's executions from here.
  steps                Manage Steps from here.
  workers              Manage Workers from here.
```

The default Cryton's REST API address and port are **localhost** and **8000**. To override this use `-H` and `-p` options. Optionally use the `--secure` flag to use the *HTTPS* protocol or the `--debug` flag for non-formatted output.

```
cryton-cli -H 127.0.0.1 -p 8000 --secure --debug <your command>
```

To learn about each command's options use:

```
cryton-cli <your command> --help
```

For a better understanding of the results, we highlight the successful ones with **green** and the others with **red**.

7.1.1 execution-variables

Manage Execution variables from here.

Options:
- help (`--help`) - Show this message and exit.

create

Create new execution variable(s) for PLAN_EXECUTION_ID from FILE.

PLAN_EXECUTION_ID IS ID of the desired PlanExecution.

FILE is path (can be multiple) to file(s) containing execution variables.

Arguments:
- PLAN_EXECUTION_ID
- FILE

Options:
- help (`--help`) - Show this message and exit.

delete

Delete Execution variable with EXECUTION_VARIABLE_ID saved in Cryton.

EXECUTION_VARIABLE_ID is ID of the Execution_variable you want to delete.

Arguments:
- EXECUTION_VARIABLE_ID

Options:
- help (`--help`) - Show this message and exit.

list

List existing Execution variables in Cryton.

Options:
- less (`--less`) - Show less like output.
- offset (`-o`, `--offset`) - The initial index from which to return the results.
- limit (`-l`, `--limit`) - Number of results to return per page.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- parent (`-p`, `--parent`) - Filter Execution variables using Plan execution ID.
- parameter_filters (`-f`, `--filter`) - Filter results using returned parameters (for example `id=1`, `name=test`, etc.).
- help (`--help`) - Show this message and exit.

show

Show Execution variable with EXECUTION_VARIABLE_ID saved in Cryton.

EXECUTION_VARIABLE_ID is ID of the Execution variable you want to see.

Arguments:
- EXECUTION_VARIABLE_ID

Options:
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

7.1.2 generate-docs

Generate Markdown documentation for CLI.

FILE is path/to/your/file where you want to save the generated documentation.

Arguments:
- FILE

Options:
- layer (`-l`, `--layer`) - Highest header level.
- help (`--help`) - Show this message and exit.

7.1.3 logs

Manage Workers from here.

Options:
- help (`--help`) - Show this message and exit.

list

List existing Logs in Cryton.

Options:
- less (`--less`) - Show less like output.
- offset (`-o`, `--offset`) - The initial index from which to return the results.
- limit (`-l`, `--limit`) - Number of results to return per page.
- parameter_filters (`-f`, `--filter`) - Phrase to use to filter the results.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

7.1.4 plan-executions

Manage Plan's executions from here.

Options:
- help (`--help`) - Show this message and exit.

delete

Delete Plan's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Plan's execution you want to delete.

Arguments:
- EXECUTION_ID

Options:
- help (`--help`) - Show this message and exit.

kill

Kill Plan's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Plan's execution you want to kill.

Arguments:
- EXECUTION_ID

Options:
- help (`--help`) - Show this message and exit.

list

List existing Plan's executions in Cryton.

Options:
- less (`--less`) - Show less like output.
- offset (`-o`, `--offset`) - The initial index from which to return the results.
- limit (`-l`, `--limit`) - Number of results to return per page.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- parent (`-p`, `--parent`) - Filter Plan executions using Run ID.
- parameter_filters (`-f`, `--filter`) - Filter results using returned parameters (for example `id=1`, `name=test`, etc.).
- help (`--help`) - Show this message and exit.

pause

Pause Plan's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Plan's execution you want to pause.

Arguments:
- EXECUTION_ID

Options:
- help (`--help`) - Show this message and exit.

report

Create report for Plan's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Plan's execution you want to create report for.

Arguments:
- EXECUTION_ID

Options:
- file (`-f`, `--file`) - File to save the report to (default is /tmp).
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

resume

Resume Plan's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Plan's execution you want to resume.

Arguments:
- EXECUTION_ID

Options:
- help (`--help`) - Show this message and exit.

show

Show Plan's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Plan's execution you want to see.

Arguments:
- EXECUTION_ID

Options:
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

validate-modules

Validate modules for Plan's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Plan's execution you want to validate modules for.

Arguments:
- EXECUTION_ID

Options:
- help (`--help`) - Show this message and exit.

7.1.5 plan-templates

Manage Plan templates from here.

Options:
- help (`--help`) - Show this message and exit.

create

Store Plan Template into Cryton.

FILE is path/to/your/file that you want to upload to Cryton.

Arguments:
- FILE

Options:
- help (`--help`) - Show this message and exit.

delete

Delete Template with TEMPLATE_ID saved in Cryton.

TEMPLATE_ID is ID of the Template you want to delete.

Arguments:
- TEMPLATE_ID

Options:
- help (`--help`) - Show this message and exit.

get-template

Get Template with TEMPLATE_ID saved in Cryton.

TEMPLATE_ID is ID of the Template you want to get.

Arguments:
- TEMPLATE_ID

Options:
- file (`-f`, `--file`) - File to save the template to (default is /tmp).
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

list

List existing Plan templates in Cryton.

Options:
- less (`--less`) - Show less like output.
- offset (`-o`, `--offset`) - The initial index from which to return the results.
- limit (`-l`, `--limit`) - Number of results to return per page.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- parameter_filters (`-f`, `--filter`) - Filter results using returned parameters (for example `id=1`, `name=test`, etc.).
- help (`--help`) - Show this message and exit.

show

Show Template with TEMPLATE_ID saved in Cryton.

TEMPLATE_ID is ID of the Template you want to see.

Arguments:
- TEMPLATE_ID

Options:
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

7.1.6 plans

Manage Plans from here.

Options:
- help (`--help`) - Show this message and exit.

create

Fill template PLAN_TEMPLATE_ID with inventory file(s) and save it to Cryton.

PLAN_TEMPLATE_ID is ID of the template you want to fill.

Arguments:
- TEMPLATE_ID

Options:
- inventory_files (`-i` , `--inventory-file`) - Inventory file used to fill the template. Can be used multiple times.
- help (`--help`) - Show this message and exit.

delete

Delete Plan with PLAN_ID saved in Cryton.

PLAN_ID is ID of the Plan you want to delete.

Arguments:
- PLAN_ID

Options:
- help (`--help`) - Show this message and exit.

execute

Execute Plan saved in Cryton with PLAN_ID on Worker with WORKER_ID and attach it to Run with RUN_ID.

PLAN_ID is ID of the Plan you want to execute.

WORKER_ID is ID of the Plan you want to execute.

RUN_ID is ID of the Run you want to attach this execution to.

Arguments:
- PLAN_ID
- WORKER_ID
- RUN_ID

Options:
- help (`--help`) - Show this message and exit.

get-plan

Get Plan with PLAN_ID saved in Cryton.

PLAN_ID is ID of the Plan you want to get.

Arguments:
- PLAN_ID

Options:
- file (`-f`, `--file`) - File to save the plan to (default is /tmp).
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

list

List existing Plans in Cryton.

Options:
- less (`--less`) - Show less like output.
- offset (`-o`, `--offset`) - The initial index from which to return the results.
- limit (`-l`, `--limit`) - Number of results to return per page.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- parameter_filters (`-f`, `--filter`) - Filter results using returned parameters (for example `id=1`, `name=test`, etc.).
- help (`--help`) - Show this message and exit.

show

Show Plan with PLAN_ID saved in Cryton.

PLAN_ID is ID of the Plan you want to see.

Arguments:
- PLAN_ID

Options:
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

validate

Validate (syntax check) your FILE with Plan.

FILE is path/to/your/file that you want to validate.

Arguments:
- FILE

Options:
- inventory_files (`-i`, `--inventory-file`) - Inventory file used to fill the template. Can be used multiple times.
- help (`--help`) - Show this message and exit.

7.1.7 runs

Manage Runs from here.

Options:
- help (`--help`) - Show this message and exit.

create

Create new Run with PLAN_ID and WORKER_IDS.

PLAN_ID is ID of the Plan you want to create Run for. (for example 1)

WORKER_IDS is list of IDs you want to use for Run. (1 2 3)

Arguments:
- PLAN_ID
- WORKER_IDS

Options:
- help (`--help`) - Show this message and exit.

delete

Delete Run with RUN_ID saved in Cryton.

RUN_ID is ID of the Run you want to delete.

Arguments:
- RUN_ID

Options:
- help (`--help`) - Show this message and exit.

execute

Execute Run saved in Cryton with RUN_ID.

RUN_ID is ID of the Run you want to execute.

Arguments:
- RUN_ID

Options:
- help (`--help`) - Show this message and exit.

get-plan

Get plan from Run with RUN_ID saved in Cryton.

RUN_ID is ID of the Run you want to get plan from.

Arguments:
- RUN_ID

Options:
- file (`-f`, `--file`) - File to save the plan to (default is /tmp).
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

kill

Kill Run saved in Cryton with RUN_ID.

RUN_ID is ID of the Run you want to kill.

Arguments:
- RUN_ID

Options:
- help (`--help`) - Show this message and exit.

list

List existing Runs in Cryton.

Options:
- less (`--less`) - Show 'less' like output.
- offset (`-o`, `--offset`) - The initial index from which to return the results.
- limit (`-l`, `--limit`) - Number of results to return per page.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- parameter_filters (`-f`, `--filter`) - Filter results using returned parameters (for example `id=1`, `name=test`, etc.).
- help (`--help`) - Show this message and exit.

pause

Pause Run saved in Cryton with RUN_ID.

RUN_ID is ID of the Run you want to pause.

Arguments:
- RUN_ID

Options:
- help (`--help`) - Show this message and exit.

postpone

Postpone Run saved in Cryton with RUN_ID by HOURS, MINUTES and SECONDS.

RUN_ID is ID of the Run you want to postpone.

HOURS is number of hours.

MINUTES is number of minutes.

SECONDS is number of seconds.

Arguments:
- RUN_ID
- HOURS
- MINUTES
- SECONDS

Options:
- help (`--help`) - Show this message and exit.

report

Create report for Run with RUN_ID saved in Cryton.

RUN_ID is ID of the Run you want to create report for.

Arguments:
- RUN_ID

Options:
- file (`-f`, `--file`) - File to save the report to (default is /tmp).
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

reschedule

Reschedule Run saved in Cryton with RUN_ID to specified DATE and TIME.

RUN_ID is ID of the Run you want to reschedule.

DATE in format year-month-day (Y-m-d).

TIME in format hours:minutes:seconds (H:M:S).

Arguments:
- RUN_ID
- TO_DATE
- TO_TIME

Options:
- utc_timezone (`--utc-timezone`) - Input time in UTC timezone.
- help (`--help`) - Show this message and exit.

resume

Resume Run saved in Cryton with RUN_ID.

RUN_ID is ID of the Run you want to resume.

Arguments:
- RUN_ID

Options:
- help (`--help`) - Show this message and exit.

schedule

Schedule Run saved in Cryton with RUN_ID to specified DATE and TIME.

RUN_ID is ID of the Run you want to schedule.

DATE in format year-month-day (Y-m-d).

TIME in format hours:minutes:seconds (H:M:S).

Arguments:
- RUN_ID
- TO_DATE
- TO_TIME

Options:
- utc_timezone (`--utc-timezone`) - Input time in UTC timezone.
- help (`--help`) - Show this message and exit.

show

Show Run with RUN_ID saved in Cryton.

RUN_ID is ID of the Run you want to see.

Arguments:
- RUN_ID

Options:
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

unschedule

Unschedule Run saved in Cryton with RUN_ID.

RUN_ID is ID of the Run you want to unschedule.

Arguments:
- RUN_ID

Options:
- help (`--help`) - Show this message and exit.

validate-modules

Validate modules for Run with RUN_ID saved in Cryton.

RUN_ID is ID of the Run you want to validate modules for.

Arguments:
- RUN_ID

Options:
- help (`--help`) - Show this message and exit.

7.1.8 stage-executions

Manage Stage's executions from here.

Options:
- help (`--help`) - Show this message and exit.

delete

Delete Stage's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Stage's execution you want to delete.

Arguments:
- EXECUTION_ID

Options:
- help (`--help`) - Show this message and exit.

kill

Kill Stage's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Stage's execution you want to kill.

Arguments:
- EXECUTION_ID

Options:
- help (`--help`) - Show this message and exit.

list

List existing Stage's executions in Cryton.

Options:
- less (`--less`) - Show less like output.
- offset (`-o`, `--offset`) - The initial index from which to return the results.
- limit (`-l`, `--limit`) - Number of results to return per page.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- parent (`-p`, `--parent`) - Filter Stage executions using Plan execution ID.
- parameter_filters (`-f`, `--filter`) - Filter results using returned parameters (for example `id=1`, `name=test`, etc.).
- help (`--help`) - Show this message and exit.

re-execute

Re-execute Stage's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Stage's execution you want to kill.

Arguments:
- EXECUTION_ID

Options:
- immediately (`--immediately`) - Re-execute StageExecution immediately without starting its Trigger.
- help (`--help`) - Show this message and exit.

report

Create report for Stage's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Stage's execution you want to create report for.

Arguments:
- EXECUTION_ID

Options:
- file (`-f`, `--file`) - File to save the report to (default is /tmp).
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

show

Show Stage's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Stage's execution you want to see.

Arguments:
- EXECUTION_ID

Options:
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

7.1.9 stages

Manage Stages from here.

Options:
- help (`--help`) - Show this message and exit.

create

Create Stage from FILE and add it to Plan with PLAN_ID.

PLAN_ID is an ID of the Plan you want to add the Stage to.

FILE is a path to the file containing the Stage template.

Arguments:
- PLAN_ID
- FILE

Options:
- inventory_files (`-i`, `--inventory-file`) - Inventory file used to fill the template. Can be used multiple times.
- help (`--help`) - Show this message and exit.

delete

Delete Stage with STAGE_ID saved in Cryton.

STAGE_ID is ID of the Stage you want to delete.

Arguments:
- STAGE_ID

Options:
- help (`--help`) - Show this message and exit.

list

List existing Stages in Cryton.

Options:
- less (`--less`) - Show less like output.
- offset (`-o`, `--offset`) - The initial index from which to return the results.
- limit (`-l`, `--limit`) - Number of results to return per page.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- parent (`-p`, `--parent`) - Filter Stages using Plan ID.
- parameter_filters (`-f`, `--filter`) - Filter results using returned parameters (for example `id=1`, `name=test`, etc.).
- help (`--help`) - Show this message and exit.

show

Show Stage with STAGE_ID saved in Cryton.

STAGE_ID is ID of the Stage you want to see.

Arguments:
- STAGE_ID

Options:
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

start-trigger

Start Stage's trigger with STAGE_ID under Plan execution with PLAN_EXECUTION_ID.

STAGE_ID is an ID of the Stage you want to start.

PLAN_EXECUTION_ID is an ID of the Plan execution you want to set as a parent of the Stage execution.

Arguments:
- STAGE_ID
- PLAN_EXECUTION_ID

Options:
- help (`--help`) - Show this message and exit.

validate

Validate (syntax check) your FILE with Stage.

FILE is path/to/your/file that you want to validate.

Arguments:
- FILE

Options:
- inventory_files (`-i`, `--inventory-file`) - Inventory file used to fill the template. Can be used multiple times.
- dynamic (`-D`, `--dynamic`) - If Stage will be used with a dynamic Plan.
- help (`--help`) - Show this message and exit.

7.1.10 step-executions

Manage Step's executions from here.

Options:
- help (`--help`) - Show this message and exit.

delete

Delete Step's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Step's execution you want to delete.

Arguments:
- EXECUTION_ID

Options:
- help (`--help`) - Show this message and exit.

kill

Kill Step's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Step's execution you want to kill.

Arguments:
- EXECUTION_ID

Options:
- help (`--help`) - Show this message and exit.

list

List existing Step's executions in Cryton.

Options:
- less (`--less`) - Show less like output.
- offset (`-o` , `--offset`) - The initial index from which to return the results.
- limit (`-l` , `--limit`) - Number of results to return per page.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- parent (`-p` , `--parent`) - Filter Step executions using Stage execution ID.
- parameter_filters (`-f` , `--filter`) - Filter results using returned parameters (for example `id=1` , `name=test` , etc.).
- help (`--help`) - Show this message and exit.

re-execute

Re-execute Step's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Step's execution you want to kill.

Arguments:
- EXECUTION_ID

Options:
- help (`--help`) - Show this message and exit.

report

Create report for Step's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Step's execution you want to create report for.

Arguments:
- EXECUTION_ID

Options:
- file (`-f` , `--file`) - File to save the report to (default is /tmp).
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

show

Show Step's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Step's execution you want to see.

Arguments:
- EXECUTION_ID

Options:
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

7.1.11 steps

Manage Steps from here.

Options:
- help (`--help`) - Show this message and exit.

create

Create Step from FILE and add it to Stage with STAGE_ID.

STAGE_ID is an ID of the Stage you want to add the Stage to.

FILE is a path to the file containing the Step template.

Arguments:
- STAGE_ID
- FILE

Options:
- inventory_files (`-i`, `--inventory-file`) - Inventory file used to fill the template. Can be used multiple times.
- help (`--help`) - Show this message and exit.

delete

Delete Step with STEP_ID saved in Cryton.

STEP_ID is ID of the Step you want to delete.

Arguments:
- STEP_ID

Options:
- help (`--help`) - Show this message and exit.

execute

Execute Step with STEP_ID under Stage execution with STAGE_EXECUTION_ID.

STEP_ID is ID of the Step you want to execute.

STAGE_EXECUTION_ID is an ID of the Stage execution you want to set as a parent of the Step execution.

Arguments:
- STEP_ID
- STAGE_EXECUTION_ID

Options:
- help (`--help`) - Show this message and exit.

list

List existing Steps in Cryton.

Options:
- less (`--less`) - Show less like output.
- offset (`-o`, `--offset`) - The initial index from which to return the results.
- limit (`-l`, `--limit`) - Number of results to return per page.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- parent (`-p`, `--parent`) - Filter Steps using Stage ID.
- parameter_filters (`-f`, `--filter`) - Filter results using returned parameters (for example `id=1`, `name=test`, etc.).
- help (`--help`) - Show this message and exit.

show

Show Step with STEP_ID saved in Cryton.

STEP_ID is ID of the Step you want to see.

Arguments:
- STEP_ID

Options:
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

validate

Validate (syntax check) your FILE with Step.

FILE is path/to/your/file that you want to validate.

Arguments:
- FILE

Options:
- inventory_files (`-i`, `--inventory-file`) - Inventory file used to fill the template. Can be used multiple times.
- help (`--help`) - Show this message and exit.

7.1.12 workers

Manage Workers from here.

Options:
- help (`--help`) - Show this message and exit.

create

Create new Worker with NAME and save it into Cryton.

NAME of your Worker (will be used to match your Worker). For example: "MyCustomName".

Arguments:
- NAME

Options:
- description (`-d`, `--description`) - Description of your Worker (wrap in "").
- force (`-f`, `--force`) - Ignore, if Worker with the same parameter 'name' exists.
- help (`--help`) - Show this message and exit.

delete

Delete Worker with WORKER_ID saved in Cryton.

WORKER_ID is ID of the Worker you want to delete.

Arguments:
- WORKER_ID

Options:
- help (`--help`) - Show this message and exit.

health-check

Check if Worker with WORKER_ID saved in Cryton is online.

WORKER_ID is ID of the Worker you want to check.

Arguments:
- WORKER_ID

Options:
- help (`--help`) - Show this message and exit.

list

List existing Workers in Cryton.

Options:
- less (`--less`) - Show less like output.
- offset (`-o`, `--offset`) - The initial index from which to return the results.
- limit (`-l`, `--limit`) - Number of results to return per page.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- parameter_filters (`-f`, `--filter`) - Filter results using returned parameters (for example `id=1`, `name=test`, etc.).
- help (`--help`) - Show this message and exit.

show

Show Worker with WORKER_ID saved in Cryton.

WORKER_ID is ID of the Worker you want to see.

Arguments:
- WORKER_ID

Options:
- less (`--less`) - Show less like output.
- localize (`--localize`) - Convert UTC datetime to local timezone.
- help (`--help`) - Show this message and exit.

7.2 Frontend

Cryton Frontend provides functionality for interacting with Cryton Core more easily and clearly than by using CLI.

7.2.1 Listing data

You can list all Cryton data by using list pages in the navigation bar. Most important data can be found directly in the dashboard. Each data table provides functionality for sorting and filtering data.

7.2.2 Creating objects

You can create workers, templates, instances, and runs by using create pages in the navigation bar. Every object can be also deleted from its list page.

7.2.3 Template creation

You can create plan templates in the **Plan templates > Create template**. The whole creation process is documented in-app with an introduction page and additional help pages for every creation step.

7.2.4 Run interaction

The front end provides 2 ways to interact with runs. There is a quick interaction menu that you can access in **Runs > List runs** by clicking on a run. The interaction menu will expand under the run. Another way is to click on the eye icon next to the run which will take you to the run's page. There you can also view the current state of the run and its sub-parts, and modify the execution variables for each execution.

Execution timeline

You can view timelines of the run's executions by clicking on the clock icon next to a run on the list runs page or by clicking on the **show timeline** button on the run's page. The timeline shows the start, pause and finish times of the whole execution, stages, and steps. More details can be found in an in-app help page found inside the timeline tab.

7.2.5 Theming

The front end provides two color themes - light and dark. You can switch between them with a toggle button in the top bar.

8. Integrated tools

8.1 Metasploit

Description of Metasploit functionalities supported by Cryton.

8.1.1 Setup

To be able to use MSF, it must be accessible to the Worker. All you need to do is start the msfrpc(d) module in MSF and set Worker's environment `CRYTON_WORKER_MSFRPCD_*` variables. After that, if you start the Worker and a connection is created, you will see the following message: `Connected to msfrpcd.`.

8.1.2 Session management

Cryton allows you to utilize sessions from Metasploit. To learn how, see session management.

8.1.3 MSF listener

Cryton allows creating a Stage that will start an MSF listener on Worker and will wait until it returns a session that matches defined parameters. For more information see MSF listener in Stage.

8.2 Empire

Description of Empire functionalities supported by Cryton.

functionalities:

1. Deploy empire agents through ssh connection or metasploit session
2. Execute shell scripts or Empire modules on active agents

8.2.1 requirements for usage with Core:

- Installed and running Empire server with version 4.1.0 and above. Installation guide here
- Installed all main Cryton components, that is Core, Worker and Cli
- Empire server needs to be able to communicate with Worker component

For Empire usage only with Worker see documentation here.

8.2.2 Step types for Empire functionalities

Empire functionalities supported by Cryton are represented by different Step types. More about the `step_type` argument in here.

8.2.3 Deploy Empire agent on a target

This functionality uses `step_type: empire/agent-deploy` and enables to deploy Empire agent on the given target (executing Empire generated payload with given parameters on target).

Usable arguments for this step type are:

Argument	Description
`listener_name`	Name of listener in Empire for identification. If listener with this name already exists in Empire, it will be used for stager generation.
`listener_port` (optional)	Port on which should be listener communicating with Agents.
`listener_options` (optional)	Additional adjustable parameters for creating listener. More on here.
`listener_type` (optional)	Type of listener (default: http).
`stager_type`	Type of stager that should be generated in form of path (example: `multi/bash'). For stager types look here.
`stager_options` (optional)	Additional adjustable parameters for generating stager. Parameters can be viewed in individual stager python files or through Empire client.
`agent_name`	Name for the deployed agent which is going to be used as a reference to this agent later.
`use_named_session` (optional)	Name of created msf session through Cryton.
`use_any_session_to_target` (optional)	Ip address of target on which has been created msf session
`session_id` (optional)	ID of msf session to target.
`ssh_connection` (optional)	Arguments for creating ssh connection to target.

Arguments for `ssh_connection`

Argument	Description
`target`	Ip address for ssh connection.
`username` (optional)	Username for ssh connection.
`password` (optional)	Password for ssh connection if `ssh_key` is not supplied.
`ssh_key` (optional)	Ssh key for ssh connection if `password` is not supplied.
`port` (optional)	Port for ssh connection (default: 22).

Example

```
- name: deploy-agent
  step_type: empire/agent-deploy
  arguments:
    use_named_session: session_to_target_1 # using named session created in step ssh-session
    listener_name: testing
    listener_port: 80
    stager_type: multi/bash
    agent_name: MyAgent # only lower/upper characters and numbers allowed in name
```

8.2.4 Execute shell script or Empire module on agent

This functionality uses `step_type: empire/execute` and allows the execution of shell commands or Empire modules on active Empire agents.

To execute a Shell command use the following arguments:

Argument	Description
`use_agent`	Name of an active agent that checked on Empire server.
`shell_command`	Shell command that should be executed on an active agent (example: `whoami`).

To execute an Empire module use the following arguments:

Argument	Description
`use_agent`	Name of an active agent that checked on Empire server.
`module`	Name of Empire module in form of a path that should be executed on the active agent (example: `collection/sniffer`). Available Empire modules here.
`module_arguments` (optional)	Additional arguments for Empire module execution.

Example

```
- name: sniffer-on-agent
  step_type: empire/execute
  arguments:
    use_agent: MyAgent
    module: collection/sniffer
    module_arguments: # Optional
      IpFilter: 192.168.33.12
      PortFilter: 1234
```

```
- name: whoami-on-agent
  step_type: empire/execute
  arguments:
    use_agent: MyAgent
    shell_command: whoami
```

8.2.5 Debugging

Empire server connection problems

1. Check that the empire server is running correctly
2. Check that the Worker component has access to the Empire server

Empire Agent cannot connect to the Empire server

1. If you are using metasploit session for agent deployment, check that the session is functioning correctly
2. Check that the Listener Host option is set to an IP address of the machine that the Empire server is running on and that the target you are deploying an Empire agent on has access to that IP address

8.2.6 Deploy Empire agent on Windows

Recommended Empire `stager_type` to use for Windows machines is `multi/launcher` right now.

IMPORTANT!!

For empire stagers to work on newer versions of Windows OS, you need to disable all **firewall** and **antivirus** protection on targeted Windows machine.

9. Dynamic execution

To support dynamic security testing. We've added support for creating dynamic plans. They allow the user to create an empty Plan/Stage and create their agent to control the execution instead of Cryton's advanced scheduler.

9.1 Features

- Create a Plan/Step/Stage for dynamic execution (an empty list of Stages/Steps can be provided)
- Add Step to Stage execution and execute it
- Add Stage to Plan execution and start it
- Added Steps are automatically set as a successor of the last Step (only if the `is_init` variable is **not** set to *True* and a possible parent Step exists)

9.2 Limitations

- Dynamic plan must have a `dynamic` variable set to *True*
- If you don't want to pass any Stages/Steps you must provide an empty list
- Each Stage and Step must have a unique name in the same Plan (utilize inventory variables to overcome this limitation)
- The Stage/Step you're trying to add must be valid
- Run's Plan must contain the instance (Stage/Step) you are trying to execute
- You cannot create multiple executions for an instance (you can execute an instance only once) under the same Plan execution

9.3 Workflow example (using CLI)

For this example we will assume that:

- Cryton Core is running (REST API is accessible at *localhost:8000*)
- Worker is registered in Core and running
- mod_cmd is accessible from the Worker

If it isn't the case, feel free to follow the installation example.

Files used in this guide can be found in the Cryton Core repository.

It's best to switch to the example directory, so we will assume that's true.

```
cd /path/to/cryton-core/examples/dynamic-execution-example/
```

9.3.1 Building a base Plan and executing it

First, we create a template

```
cryton-cli plan-templates create template.yml
```

Create a Plan (instance)

```
cryton-cli plans create <template_id>
```

Add a Stage to the Plan (update the inventory file to your needs)

```
cryton-cli stages create <plan_id> stage.yml -i stage-inventory.yml
```

Add an initial Step to the Stage

```
cryton-cli steps create <stage_id> step-init.yml
```

Add a reusable Step to the Stage (update the inventory file to your needs)

```
cryton-cli steps create <stage_id> step-reusable.yml -i step-reusable-inventory.yml
```

Create a Worker you want to test on

```
cryton-cli workers create local
```

Create a Run

```
cryton-cli runs create <plan_id> <worker_id>
```

Execute the Run

```
cryton-cli runs execute <run_id>
```

9.3.2 Start standalone Stage:

Add your Stage to the desired Plan (**Update the inventory file! Stage names must be unique.**)

```
cryton-cli stages create <plan_id> stage.yml -i stage-inventory.yml
```

Start your Stage (its trigger) under the desired Plan execution

```
cryton-cli stages start-trigger <stage_id> <plan_execution_id>
```

9.3.3 Execute standalone Step:

Add your Step to the desired Stage (**Update the inventory file! Step names must be unique.**)

```
cryton-cli steps create <stage_id> step-reusable.yml -i step-reusable-inventory.yml
```

Execute your Step under the desired Stage execution

```
cryton-cli steps execute <step_id> <stage_execution_id>
```

9.3.4 Check results - works only once the Run is created:

```
cryton-cli runs report 1 --less
```

9.4 Automation using Python

You will probably want to automate these actions rather than using CLI to do them. For this purpose, we will create a simple Python script that will:

1. Create a template
2. Create a Plan
3. Add a Stage
4. Add a Step
5. Create a Run
6. Execute the Run
7. Create a new Step
8. Execute the new Step
9. Get the Run report

For this example we will assume that:

- Cryton Core is running (REST API is accessible at *localhost:8000*)
- Worker is registered in Core and running
- mod_cmd is accessible from the Worker

If it isn't the case, feel free to follow the installation example.

Once we have met all the requirements, we can copy the following script, **update the `WORKER_ID` variable**, and run it using `python3 my_automation_script.py`.

```
import requests
import yaml
import time

WORKER_ID = 0

TEMPLATE = {
    "plan": {
        "name": "example",
        "owner": "Cryton",
        "dynamic": True,
        "stages": []
    }
}

STAGE = {
    "name": "no delay stage {{ id }}",
    "trigger_type": "delta",
    "trigger_args": {
        "seconds": 0
    },
    "steps": []
}

STEP = {
    "name": "initial step",
    "step_type": "worker/execute",
    "is_init": True,
    "arguments": {
        "module": "mod_cmd",
        "module_arguments": {
            "cmd": "whoami"
        }
    }
}

STEP_REUSABLE = {
    "name": "reusable step {{ id }}",
    "step_type": "worker/execute",
    "arguments": {
        "module": "mod_cmd",
        "module_arguments": {
            "cmd": "{{ command }}"
        }
    }
}


def get_api_root():
```

```
    api_address = "localhost"
    api_port = 8000
    return f"http://{api_address}:{api_port}/api/"


if __name__ == "__main__":
    # Check if the Worker is specified
    if WORKER_ID < 1:
        raise Exception("Please specify a correct Worker ID at the top of the file.")
    print(f"Worker id: {WORKER_ID}")

    # Get api root
    api_root = get_api_root()

    # 1. Create a template
    r_create_template = requests.post(f"{api_root}templates/", files={"file": yaml.dump(TEMPLATE)})
    template_id = r_create_template.json()['id']
    print(f"Template id: {template_id}")

    # 2. Create a Plan
    r_create_plan = requests.post(f"{api_root}plans/", data={'template_id': template_id})
    plan_id = r_create_plan.json()['id']
    print(f"Plan id: {plan_id}")

    # 3. Add a Stage
    stage_inventory = {"id": 1}
    r_create_stage = requests.post(f"{api_root}stages/", data={'plan_id': plan_id},
                                   files={"file": yaml.dump(STAGE), "inventory_file": yaml.dump(stage_inventory)})
    stage_id = r_create_stage.json()['id']
    print(f"Stage id: {stage_id}")

    # 4. Add a Step
    r_create_step = requests.post(f"{api_root}steps/", data={'stage_id': stage_id}, files={"file": yaml.dump(STEP)})
    step_id = r_create_step.json()['id']
    print(f"Step id: {step_id}")

    # 5. Create a new Run
    r_create_run = requests.post(f"{api_root}runs/", data={'plan_id': plan_id, "worker_ids": [WORKER_ID]})
    run_id = r_create_run.json()["id"]
    print(f"Run id: {run_id}")

    # 6. Execute the Run
    r_execute_run = requests.post(f"{api_root}runs/{run_id}/execute/", data={'run_id': run_id})
    print(f"Run response: {r_execute_run.text}")

    # 7. Create a new Step
    step_inventory = {"id": 1, "command": "echo test"}
    r_create_step2 = requests.post(f"{api_root}steps/", data={'stage_id': stage_id},
                                   files={"file": yaml.dump(STEP_REUSABLE),
                                          "inventory_file": yaml.dump(step_inventory)})
    step_id2 = r_create_step2.json()['id']
    print(f"Second step id: {step_id2}")

    # 8. Execute the new Step (First, get Stage execution's id)
    stage_execution_id = requests.get(f"{api_root}runs/{run_id}/report/")\
        .json()["detail"]["plan_executions"][0]["stage_executions"][0]["id"]
    r_execute_step = requests.post(f"{api_root}steps/{step_id2}/execute/",
                                   data={'stage_execution_id': stage_execution_id})
    print(f"Second Step response: {r_execute_step.text}")

    # 9. Get Run report
    for i in range(5):
        time.sleep(3)
        current_state = requests.get(f"{api_root}runs/{run_id}/").json()["state"]
        if current_state == "FINISHED":
            break
        print(f"Waiting for a final state. Current state: {current_state}")

    print()
    print("Report: ")
    print(yaml.dump(requests.get(f"{api_root}runs/{run_id}/report/").json()["detail"]))
```

10. Logging

The logs adhere to the following format:

```
{"queue": "cryton_core.control.request", "event": "Queue declared and consuming", "logger": "cryton-debug", "level": "info", "timestamp":
"2021-05-18T11:19:20.012152Z"}
{"plan_name": "Example scenario", "plan_id": 129, "status": "success", "event": "plan created", "logger": "cryton", "level": "info", "timestamp":
"2021-05-18T06:17:39.753017Z"}
```

When running in Docker, you can always check the logs by:

```
docker logs CONTAINER_NAME
```

10.1 Core

Every change of state is logged for later analysis. Every Step the result is also logged, although output is not. It can be found in the database.

10.1.1 Loggers

You can choose from two loggers - **debug** or **production**, which you can set by environment variable *CRYTON_CORE_DEBUG*.

For **production**: - RotatingFileHandler *CRYTON_CORE_APP_DIRECTORY*/log/cryton-core.log

For **debug**: - RotatingFileHandler *CRYTON_CORE_APP_DIRECTORY*/log/cryton-core-debug.log - Console (std_out)

For running tests the **cryton-core-test** logger is used.

10.2 Worker

Each request and its processing are logged for later analysis.

10.2.1 Loggers

You can choose from two loggers - **debug** or **production**, which you can set by environment variable *CRYTON_WORKER_DEBUG*.

For **production**: - RotatingFileHandler *CRYTON_WORKER_APP_DIRECTORY*/log/cryton-worker.log

For **debug**: - RotatingFileHandler *CRYTON_WORKER_APP_DIRECTORY*/log/cryton-worker-debug.log - Console (std_out)

For running tests the **cryton-worker-test** logger is used.

11. How to contribute

11.1 Fixing and reporting bugs

Any identified bugs should be posted as an issue in the respective gitlab repository. Please, include as much detail as possible for the developers, to be able to reproduce the erroneous behavior.

11.2 Writing Attack modules

To make attack scenario automation easier we need to create and maintain attack modules. To support project development checkout section How to create Attack module.

12. License

Cryton is open-source software developed by **Masaryk University**, and distributed under **MIT license**.

12.1 License Terms

CONTENTS:

CHAPTER

ONE

API REFERENCE

This page contains auto-generated API reference documentation[1].

1.1 cryton_core

1.1.1 Subpackages

cryton_core.cryton_app

Subpackages

cryton_core.cryton_app.management

Subpackages

cryton_core.cryton_app.management.commands

Submodules

cryton_core.cryton_app.management.commands.runserver

Module Contents

Classes

Command

class cryton_core.cryton_app.management.commands.runserver.Command

Bases: django.core.management.commands.runserver.Command

handle(**args*, ***options*)

[1] Created with sphinx-autoapi

cryton_core.cryton_app.management.commands.startgunicorn

Module Contents

Classes

GunicornApplication
Command

class cryton_core.cryton_app.management.commands.startgunicorn.GunicornApplication(*app*, *options=None*)

Bases: gunicorn.app.base.BaseApplication

init(*parser*, *opts*, *args*)

load_config()

load()

class cryton_core.cryton_app.management.commands.startgunicorn.Command

Bases: django.core.management.base.BaseCommand

add_arguments(*parser*)

handle(**args*, ***options*)

cryton_core.cryton_app.management.commands.startlistener

Module Contents

Classes

Command

class cryton_core.cryton_app.management.commands.startlistener.Command

Bases: django.core.management.base.BaseCommand

handle(**args*, ***options*)

cryton_core.cryton_app.management.commands.startmonitoring

Module Contents

Classes

Command	

Functions

monitor_health()	

cryton_core.cryton_app.management.commands.startmonitoring.monitor_health()

class cryton_core.cryton_app.management.commands.startmonitoring.Command

Bases: django.core.management.base.BaseCommand

handle(**args*, ***options*)

cryton_core.cryton_app.views

Submodules

cryton_core.cryton_app.views.execution_variable_views

Module Contents

Classes

ExecutionVariableViewSet	ExecutionVariable ViewSet.

class cryton_core.cryton_app.views.execution_variable_views.ExecutionVariableViewSet

Bases: cryton_core.cryton_app.util.InstanceFullViewSet

ExecutionVariable ViewSet.

queryset

http_method_names = ['get', 'post', 'delete']

serializer_class

create(*request: rest_framework.request.Request*)

destroy(*request*, **args*, ***kwargs*)

cryton_core.cryton_app.views.log_views

Module Contents

Classes

LogViewSet	Log ViewSet.

class cryton_core.cryton_app.views.log_views.LogViewSet

Bases: cryton_core.cryton_app.util.BaseViewSet

Log ViewSet.

http_method_names = ['get']

serializer_class

list(*request: rest_framework.request.Request*)

cryton_core.cryton_app.views.plan_execution_views

Module Contents

Classes

PlanExecutionViewSet	PlanExecution ViewSet.

class cryton_core.cryton_app.views.plan_execution_views.PlanExecutionViewSet

Bases: cryton_core.cryton_app.util.ExecutionViewSet

PlanExecution ViewSet.

queryset

http_method_names = ['get', 'post', 'delete']

serializer_class

destroy(*request: rest_framework.request.Request*, **args*, ***kwargs*)

report(_, ***kwargs*)

pause(_, ***kwargs*)

unpause(_, ***kwargs*)

validate_modules(_, ***kwargs*)

kill(_, ***kwargs*)

cryton_core.cryton_app.views.plan_template_views

Module Contents

Classes

PlanTemplateViewSet	Plan's template ViewSet.

class cryton_core.cryton_app.views.plan_template_views.PlanTemplateViewSet

Bases: cryton_core.cryton_app.util.InstanceFullViewSet

Plan's template ViewSet.

queryset

http_method_names = ['get', 'post', 'delete']

serializer_class

get_template(_, **kwargs)

destroy(*request: rest_framework.request.Request*, **args*, ***kwargs*)

cryton_core.cryton_app.views.plan_views

Module Contents

Classes

PlanViewSet	Plan ViewSet.

class cryton_core.cryton_app.views.plan_views.PlanViewSet

Bases: cryton_core.cryton_app.util.InstanceFullViewSet

Plan ViewSet.

queryset

http_method_names = ['get', 'post', 'delete']

serializer_class

create(*request: rest_framework.request.Request*, ***kwargs*)

destroy(*request: rest_framework.request.Request*, **args*, ***kwargs*)

validate(*request: rest_framework.request.Request*)

execute(*request: rest_framework.request.Request*, ***kwargs*)

get_plan(_, **kwargs)

cryton_core.cryton_app.views.run_views

Module Contents

Classes

RunViewSet	Run ViewSet.

class cryton_core.cryton_app.views.run_views.RunViewSet

Bases: cryton_core.cryton_app.util.ExecutionFullViewSet

Run ViewSet.

queryset

http_method_names = ['get', 'post', 'delete']

serializer_class

create(*request: rest_framework.request.Request*, ***kwargs*)

destroy(*request: rest_framework.request.Request*, **args*, ***kwargs*)

report(_, ***kwargs*)

pause(_, ***kwargs*)

unpause(_, ***kwargs*)

schedule(*request: rest_framework.request.Request*, ***kwargs*)

execute(_, ***kwargs*)

reschedule(*request: rest_framework.request.Request*, ***kwargs*)

postpone(*request: rest_framework.request.Request*, ***kwargs*)

unschedule(_, ***kwargs*)

kill(_, ***kwargs*)

validate_modules(_, ***kwargs*)

get_plan(_, ***kwargs*)

cryton_core.cryton_app.views.stage_execution_views

Module Contents

Classes

StageExecutionViewSet	StageExecution ViewSet.

class cryton_core.cryton_app.views.stage_execution_views.StageExecutionViewSet

Bases: cryton_core.cryton_app.util.ExecutionViewSet

StageExecution ViewSet.

queryset

http_method_names = ['get', 'post', 'delete']

serializer_class

destroy(*request: rest_framework.request.Request*, **args*, ***kwargs*)

report(_, ***kwargs*)

kill(_, ***kwargs*)

re_execute(*request: rest_framework.request.Request*, ***kwargs*)

cryton_core.cryton_app.views.stage_views

Module Contents

Classes

StageViewSet	Stage ViewSet.

class cryton_core.cryton_app.views.stage_views.StageViewSet

Bases: cryton_core.cryton_app.util.InstanceFullViewSet

Stage ViewSet.

queryset

http_method_names = ['get', 'post', 'delete']

serializer_class

create(*request: rest_framework.request.Request*, **args*, ***kwargs*)

destroy(*request: rest_framework.request.Request*, **args*, ***kwargs*)

validate(*request: rest_framework.request.Request*)

start_trigger(*request: rest_framework.request.Request*, ***kwargs*)

cryton_core.cryton_app.views.step_execution_views

Module Contents

Classes

StepExecutionViewSet	StepExecution ViewSet.

class cryton_core.cryton_app.views.step_execution_views.StepExecutionViewSet

Bases: cryton_core.cryton_app.util.ExecutionViewSet

StepExecution ViewSet.

queryset

http_method_names = ['get', 'post', 'delete']

serializer_class

destroy(*request: rest_framework.request.Request*, **args*, ***kwargs*)

report(_, ***kwargs*)

kill(_, ***kwargs*)

re_execute(_, ***kwargs*)

cryton_core.cryton_app.views.step_views

Module Contents

Classes

StepViewSet	Step ViewSet.

class cryton_core.cryton_app.views.step_views.StepViewSet

Bases: cryton_core.cryton_app.util.InstanceFullViewSet

Step ViewSet.

queryset

http_method_names = ['get', 'post', 'delete']

serializer_class

create(*request: rest_framework.request.Request*, **args*, ***kwargs*)

destroy(*request: rest_framework.request.Request*, **args*, ***kwargs*)

validate(*request: rest_framework.request.Request*)

execute(*request: rest_framework.request.Request*, ***kwargs*)

cryton_core.cryton_app.views.worker_views

Module Contents

Classes

WorkerViewSet	Worker ViewSet.

class cryton_core.cryton_app.views.worker_views.WorkerViewSet

Bases: cryton_core.cryton_app.util.InstanceFullViewSet

Worker ViewSet.

queryset

http_method_names = ['get', 'post', 'delete']

serializer_class

create(*request: rest_framework.request.Request*)

healthcheck(_, ***kwargs*)

destroy(*request: rest_framework.request.Request*, **args*, ***kwargs*)

Submodules

cryton_core.cryton_app.admin

cryton_core.cryton_app.apps

Module Contents

Classes

CrytonCoreAppConfig

class cryton_core.cryton_app.apps.CrytonCoreAppConfig

Bases: django.apps.AppConfig

default_auto_field = 'django.db.models.BigAutoField'

name = 'cryton_core.cryton_app'

cryton_core.cryton_app.exceptions

Module Contents

Classes

ApiWrongObjectState
RpcTimeout

class cryton_core.cryton_app.exceptions.ApiWrongObjectState

Bases: rest_framework.exceptions.APIException

status_code

default_detail

default_code = 'wrong_state'

class cryton_core.cryton_app.exceptions.RpcTimeout

Bases: rest_framework.exceptions.APIException

status_code

default_detail

default_code = 'error'

cryton_core.cryton_app.models

Module Contents

Classes

AdvancedModel
InstanceModel
ExecutionModel
ExtendedExecutionModel
PlanModel
StageModel
StepModel
WorkerModel
RunModel
PlanExecutionModel
StageExecutionModel
StepExecutionModel
SessionModel
ExecutionVariableModel
SuccessorModel
CorrelationEventModel
PlanTemplateModel
OutputMappingModel
DependencyModel

class cryton_core.cryton_app.models.AdvancedModel

Bases: django.db.models.Model

class Meta

abstract = True

created_at

updated_at

class cryton_core.cryton_app.models.InstanceModel

Bases: AdvancedModel

class Meta

abstract = True

name

class cryton_core.cryton_app.models.ExecutionModel

Bases: AdvancedModel

class Meta

abstract = True

state

start_time

pause_time

finish_time

class cryton_core.cryton_app.models.ExtendedExecutionModel

Bases: ExecutionModel

class Meta

abstract = True

schedule_time

aps_job_id

class cryton_core.cryton_app.models.PlanModel

Bases: InstanceModel

owner

dynamic

class cryton_core.cryton_app.models.StageModel

Bases: InstanceModel

plan_model

trigger_type

trigger_args

class cryton_core.cryton_app.models.StepModel

Bases: InstanceModel

stage_model

step_type

arguments

is_init

is_final

output_prefix

class cryton_core.cryton_app.models.WorkerModel

Bases: InstanceModel

description

state

class cryton_core.cryton_app.models.RunModel

Bases: ExtendedExecutionModel

plan_model

class cryton_core.cryton_app.models.PlanExecutionModel

Bases: ExtendedExecutionModel

run

plan_model

worker

evidence_directory

class cryton_core.cryton_app.models.StageExecutionModel

Bases: ExtendedExecutionModel

plan_execution

stage_model

trigger_id

class cryton_core.cryton_app.models.StepExecutionModel

Bases: ExecutionModel

stage_execution

step_model

result

serialized_output

output

valid

parent_id

class cryton_core.cryton_app.models.SessionModel

Bases: InstanceModel

plan_execution

msf_id

class cryton_core.cryton_app.models.ExecutionVariableModel

Bases: InstanceModel

plan_execution

value

class cryton_core.cryton_app.models.SuccessorModel

Bases: django.db.models.Model

type

value

parent

successor

class cryton_core.cryton_app.models.CorrelationEventModel

Bases: django.db.models.Model

correlation_id

step_execution

class cryton_core.cryton_app.models.PlanTemplateModel

Bases: django.db.models.Model

file

class cryton_core.cryton_app.models.OutputMappingModel

Bases: django.db.models.Model

step_model

name_from

name_to

class cryton_core.cryton_app.models.DependencyModel

Bases: django.db.models.Model

stage_model

dependency

cryton_core.cryton_app.serializers

Module Contents

Classes

BaseSerializer
ListSerializer
DetailStringSerializer
DetailDictionarySerializer
CreateDetailSerializer
ExecutionCreateDetailSerializer
CreateWithFilesSerializer
CreateMultipleDetailSerializer
LogListResponseSerializer
LogSerializer
PlanSerializer
PlanCreateSerializer
PlanExecuteSerializer
StageSerializer
StageCreateSerializer
StageValidateSerializer
StageStartTriggerSerializer
StepSerializer
StepCreateSerializer
StepExecuteSerializer
RunSerializer
RunCreateSerializer

continues on next page

Table 1 – continued from previous page

RunCreateDetailSerializer
RunScheduleSerializer
RunPostponeSerializer
PlanExecutionSerializer
PlanExecutionListSerializer
StageExecutionSerializer
StageExecutionReExecuteSerializer
StageExecutionListSerializer
StepExecutionSerializer
StepExecutionListSerializer
WorkerSerializer
WorkerCreateSerializer
SessionSerializer
ExecutionVariableSerializer
ExecutionVariableCreateSerializer
ExecutionVariableListSerializer
SuccessorSerializer
CorrelationEventSerializer
PlanTemplateSerializer
OutputMappingSerializer
DependencySerializer

class cryton_core.cryton_app.serializers.BaseSerializer

Bases: rest_framework.serializers.Serializer

create(*validated_data*)

update(*instance*, *validated_data*)

class cryton_core.cryton_app.serializers.ListSerializer

Bases: BaseSerializer

order_by

any_returned_parameter

class cryton_core.cryton_app.serializers.DetailStringSerializer

Bases: BaseSerializer

detail

class cryton_core.cryton_app.serializers.DetailDictionarySerializer

Bases: BaseSerializer

detail

class cryton_core.cryton_app.serializers.CreateDetailSerializer

Bases: DetailStringSerializer

id

class cryton_core.cryton_app.serializers.ExecutionCreateDetailSerializer

Bases: DetailStringSerializer

execution_id

class cryton_core.cryton_app.serializers.CreateWithFilesSerializer

Bases: BaseSerializer

file

inventory_file

class cryton_core.cryton_app.serializers.CreateMultipleDetailSerializer

Bases: DetailStringSerializer

ids

class cryton_core.cryton_app.serializers.LogListResponseSerializer

Bases: BaseSerializer

count

results

class cryton_core.cryton_app.serializers.LogSerializer

Bases: BaseSerializer

detail

class cryton_core.cryton_app.serializers.PlanSerializer

Bases: rest_framework.serializers.ModelSerializer

class Meta

model

exclude = []

class cryton_core.cryton_app.serializers.PlanCreateSerializer

Bases: BaseSerializer

template_id

file

class cryton_core.cryton_app.serializers.PlanExecuteSerializer

Bases: BaseSerializer

run_id

worker_id

class cryton_core.cryton_app.serializers.StageSerializer

Bases: rest_framework.serializers.ModelSerializer

class Meta

model

exclude = []

class cryton_core.cryton_app.serializers.StageCreateSerializer

Bases: CreateWithFilesSerializer

plan_id

class cryton_core.cryton_app.serializers.StageValidateSerializer

Bases: CreateWithFilesSerializer

dynamic

class cryton_core.cryton_app.serializers.StageStartTriggerSerializer

Bases: BaseSerializer

plan_execution_id

class cryton_core.cryton_app.serializers.StepSerializer

Bases: rest_framework.serializers.ModelSerializer

class Meta

model

exclude = []

class cryton_core.cryton_app.serializers.StepCreateSerializer

Bases: CreateWithFilesSerializer

stage_id

class cryton_core.cryton_app.serializers.StepExecuteSerializer

Bases: BaseSerializer

stage_execution_id

class cryton_core.cryton_app.serializers.RunSerializer

Bases: rest_framework.serializers.ModelSerializer

class Meta

model

exclude = []

class cryton_core.cryton_app.serializers.RunCreateSerializer

Bases: BaseSerializer

plan_id

worker_ids

class cryton_core.cryton_app.serializers.RunCreateDetailSerializer

Bases: CreateDetailSerializer

plan_execution_ids

class cryton_core.cryton_app.serializers.RunScheduleSerializer

Bases: BaseSerializer

start_time

class cryton_core.cryton_app.serializers.RunPostponeSerializer

Bases: BaseSerializer

delta

class cryton_core.cryton_app.serializers.PlanExecutionSerializer

Bases: rest_framework.serializers.ModelSerializer

class Meta

model

exclude = []

class cryton_core.cryton_app.serializers.PlanExecutionListSerializer

Bases: ListSerializer

run_id

plan_model_id

class cryton_core.cryton_app.serializers.StageExecutionSerializer

Bases: rest_framework.serializers.ModelSerializer

class Meta

model

exclude = []

class cryton_core.cryton_app.serializers.StageExecutionReExecuteSerializer

Bases: BaseSerializer

immediately

class cryton_core.cryton_app.serializers.StageExecutionListSerializer

Bases: ListSerializer

plan_execution_id

stage_model_id

class cryton_core.cryton_app.serializers.StepExecutionSerializer

Bases: rest_framework.serializers.ModelSerializer

class Meta

model

exclude = []

class cryton_core.cryton_app.serializers.StepExecutionListSerializer

Bases: ListSerializer

stage_execution_id

step_model_id

class cryton_core.cryton_app.serializers.WorkerSerializer

Bases: rest_framework.serializers.ModelSerializer

class Meta

model

exclude = []

class cryton_core.cryton_app.serializers.WorkerCreateSerializer

Bases: BaseSerializer

name

description

force

class cryton_core.cryton_app.serializers.SessionSerializer

Bases: rest_framework.serializers.ModelSerializer

class Meta

model

exclude = []

class cryton_core.cryton_app.serializers.ExecutionVariableSerializer

Bases: rest_framework.serializers.ModelSerializer

class Meta

model

exclude = []

class cryton_core.cryton_app.serializers.ExecutionVariableCreateSerializer

Bases: BaseSerializer

plan_execution_id

file

class cryton_core.cryton_app.serializers.ExecutionVariableListSerializer

Bases: ListSerializer

plan_execution_id

class cryton_core.cryton_app.serializers.SuccessorSerializer

Bases: rest_framework.serializers.ModelSerializer

class Meta

model

exclude = []

class cryton_core.cryton_app.serializers.CorrelationEventSerializer

Bases: rest_framework.serializers.ModelSerializer

class Meta

model

exclude = []

class cryton_core.cryton_app.serializers.PlanTemplateSerializer

Bases: rest_framework.serializers.ModelSerializer

class Meta

model

exclude = []

class cryton_core.cryton_app.serializers.OutputMappingSerializer

Bases: rest_framework.serializers.ModelSerializer

class Meta

model

exclude = []

class cryton_core.cryton_app.serializers.DependencySerializer

Bases: rest_framework.serializers.ModelSerializer

class Meta

model

exclude = []

cryton_core.cryton_app.urls

Module Contents

cryton_core.cryton_app.urls.router

cryton_core.cryton_app.urls.urlpatterns

cryton_core.cryton_app.util

Module Contents

Classes

IgnoreNestedUndefined	An undefined that is chainable, where both __getattr__ and
BaseViewSet	A ViewSet that provides default list() action.
InstanceViewSet	A ViewSet that provides default retrieve(), destroy(), and list() actions.
InstanceFullViewSet	A ViewSet that provides default retrieve(), destroy(), list(), and create() actions.
ExecutionViewSet	A ViewSet that provides default retrieve(), destroy(), and list() actions.
ExecutionFullViewSet	A ViewSet that provides default retrieve(), destroy(), list(), and create() actions.

Functions

filter_decorator(func)	Decorator for filtering of serializer results.
get_start_time(→ datetime.datetime)	Parse start time and its timezone.
parse_inventory(→ dict)	Reads inventory file (JSON, YAML, INI) and returns it as a dictionary
get_inventory_variables_from_files(→ dict)	Get all inventory variables from input.
fill_template(→ str)	Fill Jinja variables in the template (if there are any inventory variables).
parse_object_from_files(→ dict)	Get serialized object from input - parse template and fill it with inventory variables.

cryton_core.cryton_app.util.filter_decorator(*func*)

Decorator for filtering of serializer results. :param func: :return: Filtered queryset

cryton_core.cryton_app.util.get_start_time(*request_data: dict*) → datetime.datetime

Parse start time and its timezone. :param request_data: Incoming request data :return: Normalized start time

cryton_core.cryton_app.util.parse_inventory(*inventory: str*) → dict

Reads inventory file (JSON, YAML, INI) and returns it as a dictionary :param inventory: Inventory file content :return: Inventory variables

cryton_core.cryton_app.util.get_inventory_variables_from_files(*files: django.utils.datastructures.MultiValueDict*) → dict

Get all inventory variables from input. :param files: Files to parse :return: All inventory variables

class cryton_core.cryton_app.util.IgnoreNestedUndefined(*hint: str | None = None, obj: Any = missing, name: str | None = None, exc: Type[jinja2.exceptions.TemplateRuntimeError] = UndefinedError*)

Bases: jinja2.ChainableUndefined, jinja2.DebugUndefined

An undefined that is chainable, where both __getattr__ and __getitem__ return itself rather than raising an UndefinedError.

```
>>> foo = ChainableUndefined(name='foo')
>>> str(foo.bar['baz'])
''
>>> foo.bar['baz'] + 42
Traceback (most recent call last):
  ...
jinja2.exceptions.UndefinedError: 'foo' is undefined
```

New in version 2.11.0.

__getattr__(*attr: str*) → *IgnoreNestedUndefined*

__getitem__(*item: str*) → *IgnoreNestedUndefined*

cryton_core.cryton_app.util.fill_template(*inventory_variables: dict, template: str*) → str

Fill Jinja variables in the template (if there are any inventory variables). :param inventory_variables: Template variables to fill the template with :param template: Template to fill :return: Filled template

cryton_core.cryton_app.util.parse_object_from_files(*files: django.utils.datastructures.MultiValueDict*) → dict

Get serialized object from input - parse template and fill it with inventory variables. :param files: Input files :return: Serialized object

class cryton_core.cryton_app.util.BaseViewSet

Bases: rest_framework.viewsets.mixins.ListModelMixin, rest_framework.viewsets.GenericViewSet

A ViewSet that provides default list() action.

class cryton_core.cryton_app.util.InstanceViewSet

Bases: rest_framework.viewsets.mixins.RetrieveModelMixin, rest_framework.viewsets.mixins.DestroyModelMixin, BaseViewSet

A ViewSet that provides default retrieve(), destroy(), and list() actions.

get_queryset()

class cryton_core.cryton_app.util.InstanceFullViewSet

Bases: rest_framework.viewsets.mixins.CreateModelMixin, InstanceViewSet

A ViewSet that provides default retrieve(), destroy(), list(), and create() actions.

class cryton_core.cryton_app.util.ExecutionViewSet

Bases: InstanceViewSet

A ViewSet that provides default retrieve(), destroy(), and list() actions.

class cryton_core.cryton_app.util.ExecutionFullViewSet

Bases: InstanceFullViewSet

A ViewSet that provides default retrieve(), destroy(), list(), and create() actions.

cryton_core.etc

Submodules

cryton_core.etc.config

Module Contents

cryton_core.etc.config.APP_DIRECTORY

cryton_core.etc.config.REPORT_DIRECTORY

cryton_core.etc.config.EVIDENCE_DIRECTORY

cryton_core.etc.config.LOG_DIRECTORY

cryton_core.etc.config.LOG_FILE_PATH

cryton_core.etc.config.LOG_FILE_PATH_DEBUG

cryton_core.etc.config.MISFIRE_GRACE_TIME = 180

cryton_core.etc.config.TIME_ZONE

cryton_core.etc.config.DEBUG

cryton_core.etc.config.DB_NAME

cryton_core.etc.config.DB_USERNAME

cryton_core.etc.config.DB_PASSWORD

cryton_core.etc.config.DB_HOST

cryton_core.etc.config.DB_PORT

cryton_core.etc.config.RABBIT_USERNAME

cryton_core.etc.config.RABBIT_PASSWORD

cryton_core.etc.config.RABBIT_HOST

cryton_core.etc.config.RABBIT_PORT

cryton_core.etc.config.Q_ATTACK_RESPONSE_NAME

cryton_core.etc.config.Q_AGENT_RESPONSE_NAME

cryton_core.etc.config.Q_EVENT_RESPONSE_NAME

cryton_core.etc.config.Q_CONTROL_REQUEST_NAME

cryton_core.etc.config.CPU_CORES

cryton_core.etc.config.EXECUTION_THREADS_PER_PROCESS

cryton_core.etc.config.DEFAULT_RPC_TIMEOUT

cryton_core.etc.config.DJANGO_API_ROOT_URL = 'api/'

cryton_core.etc.config.DJANGO_ALLOWED_HOSTS

cryton_core.etc.config.DJANGO_ALLOWED_HOSTS

cryton_core.etc.config.DJANGO_STATIC_ROOT

cryton_core.etc.config.DJANGO_SECRET_KEY

cryton_core.etc.config.DJANGO_USE_STATIC_FILES

cryton_core.etc.config.UPLOAD_DIRECTORY_RELATIVE = 'uploads/'

cryton_core.lib

Subpackages

cryton_core.lib.models

Submodules

cryton_core.lib.models.plan

Module Contents

Classes

Plan	
PlanExecution	

Functions

execution(→ None)	Create PlanExecution object and call its execute method

class cryton_core.lib.models.plan.Plan(**kwargs)

property model: Type[cryton_core.cryton_app.models.PlanModel] | cryton_core.cryton_app.models.PlanModel

property name: str

property owner: str

property dynamic: bool

delete()

static filter(**kwargs) → django.db.models.query.QuerySet

List PlanModel objects fulfilling fields specified in kwargs. If no such fields are specified all objects are returned. :param kwargs: dict of field-value pairs to filter by :raises WrongParameterError: invalid field is specified :return: Queryset of PlanModel objects

static validate_unique_values(*plan_dict*) → None

Check if there are any duplicate names in Plan that should be unique :param plan_dict: Plan dictionary :raises exceptions.DuplicateNameInPlan: :return: None

static validate(*plan_dict*, *dynamic: bool = False*) → None

Check if plan's dictionary is valid :param plan_dict: Plan information :param dynamic: If the Plan is static or dynamic :raises

exceptions.PlanValidationError: exceptions.StageValidationError exceptions.StepValidationError

Returns

True if dictionary is valid

generate_plan() → dict

Get Plan's YAML from database. :return: Plan and its Stages/Steps

class cryton_core.lib.models.plan.PlanExecution(**kwargs)

property model: Type[cryton_core.cryton_app.models.PlanExecutionModel] | cryton_core.cryton_app.models.PlanExecutionModel

property state: str

property schedule_time: datetime.datetime | None

property start_time: datetime.datetime | None

property pause_time: datetime.datetime | None

property finish_time: datetime.datetime | None

property aps_job_id: str

property evidence_directory: str

property all_stages_finished: bool

delete()

_create_evidence_directory() → None

Generate directory for storing execution evidence. :return: None

_prepare_executions()

Prepare execution for each Stage. :return: None

schedule(*schedule_time: datetime.datetime*) → None

Schedule all plan's stages. :param schedule_time: Time to schedule to :return: None :raises

:exception RuntimeError

execute() → None

Execute Plan. This method starts triggers. :return: None

unschedule() → None

Unschedule plan execution. :return: None

reschedule(*new_time: datetime.datetime*) → None

Reschedule plan execution. :param new_time: Time to reschedule to :raises UserInputError: when provided time < present :return: None

pause() → None

Pause plan execution. :return: None

unpause() → None

validate_modules()

For each stage validate if worker is up, all modules are present and module args are correct.

start_triggers() → None

Start triggers for all execution stages. :return: None

static filter(***kwargs*) → django.db.models.query.QuerySet

List PlanExecutionModel objects fulfilling fields specified in kwargs. If no such fields are specified all objects are returned. :param kwargs: dict of field-value pairs to filter by :return: Queryset of PlanExecutionModel objects :raises WrongParameterError: invalid field is specified

report() → dict

Generate a report from Plan execution. :return: report from Plan execution

kill() → None

Kill current PlanExecution and its StageExecutions :return: None

finish() → None

Finish execution - update necessary variables. :return: None

cryton_core.lib.models.plan.execution(*plan_execution_id: int*) → None

Create PlanExecution object and call its execute method :param plan_execution_id: desired PlanExecutionModel's ID :return: None

cryton_core.lib.models.run

Module Contents

Classes

RunReport
Run

Functions

execution(→ None)	Create Run object and call its execute method

class cryton_core.lib.models.run.RunReport

id: int

plan_id: int

plan_name: str

state: str

schedule_time: datetime.datetime

start_time: datetime.datetime

pause_time: datetime.datetime

finish_time: datetime.datetime

plan_executions: list

class cryton_core.lib.models.run.Run(**kwargs)

property model: Type[cryton_core.cryton_app.models.RunModel] | cryton_core.cryton_app.models.RunModel

Get or set the RunModel.

property schedule_time: datetime.datetime | None

property start_time: datetime.datetime | None

Get or set start time of RunModel.

property pause_time: datetime.datetime | None

Get or set pause time of RunModel.

property finish_time: datetime.datetime | None

Get or set pause time of RunModel.

property state

Get or set pause time of RunModel.

property aps_job_id: str

property workers: List[cryton_core.cryton_app.models.WorkerModel]

property all_plans_finished: bool

delete()

Delete RunModel :return:

static filter(**kwargs) → django.db.models.query.QuerySet

Get list of RunModel according to no or specified conditions :param kwargs: dict of parameters to filter by :return:

_prepare_executions()

Prepare execution for each worker. :return: None

report() → dict

schedule(*schedule_time: datetime.datetime*) → None

Schedules Run for specific time. :param schedule_time: Desired start time :return: None :raises

:exception RuntimeError

unschedule() → None

Unschedules Run on specified workers :return: None

reschedule(*schedule_time: datetime.datetime*) → None

Reschedules Run on specified WorkerModels :param schedule_time: Desired start time :return: None

pause() → None

Pauses Run on specified WorkerModels :return:

unpause() → None

Unpauses Run on specified WorkerModels :return:

postpone(*delta: datetime.timedelta*) → None

Postpones Run on specified WorkerModels :param delta: Time delta :return:

execute() → None

Executes Run :return:

kill() → None

Kill current Run and its PlanExecutions :return: None

validate_modules()

For each Plan validate if worker is up, all modules are present and module args are correct.

finish() → None

Finish execution - update necessary variables. :return: None

cryton_core.lib.models.run.execution(*run_model_id: int*) → None

Create Run object and call its execute method :param run_model_id: desired RunModel's ID :return: None

cryton_core.lib.models.session

Module Contents

Functions

create_session(...)	param plan_execution_id
get_msf_session_id(→ str)	Get a Metasploit session ID by the defined session name
set_msf_session_id(→ int)	Update metasploit session ID
get_session_ids(→ list)	Get list of session IDs to specified IP

cryton_core.lib.models.session.create_session(*plan_execution_id: Type[int] | int*, *session_id: str*, *session_name: str = None*) → *cryton_core.cryton_app.models.SessionModel*

Parameters

- plan_execution_id –
- session_id –
- session_name –

Returns

cryton_core.lib.models.session.get_msf_session_id(*session_name: str*, *plan_execution_id: Type[int] | int*) → str

Get a Metasploit session ID by the defined session name

Parameters

- session_name (str) – Session name provided in input file
- plan_execution_id (int) – ID of the desired plan execution

Raises
SessionObjectDoesNotExist: If Session doesn't exist

Returns
Metasploit session ID

cryton_core.lib.models.session.set_msf_session_id(*session_name: str*, *msf_session_id: str*, *plan_execution_id: Type[int] | int*) → int

Update metasploit session ID

Parameters

- plan_execution_id (int) – ID of the desired plan execution
- msf_session_id – Metasploit session ID
- session_name (str) – Session name

Returns
ID of the named session

cryton_core.lib.models.session.get_session_ids(*target_ip: str*, *plan_execution_id: Type[int] | int*) → list

Get list of session IDs to specified IP

Parameters

- target_ip (str) – Target IP
- plan_execution_id (int) – ID of the desired Plan execution

Returns
List of session IDs

cryton_core.lib.models.stage

Module Contents

Classes

StageReport	
Stage	
StageExecution	

Functions

execution(→ None)	Create StageExecution object and call its execute method

class cryton_core.lib.models.stage.StageReport

id: int

stage_name: str

state: str

start_time: datetime.datetime

pause_time: datetime.datetime

finish_time: datetime.datetime

schedule_time: datetime.datetime

step_executions: list

class cryton_core.lib.models.stage.Stage(**kwargs)

property model: Type[cryton_core.cryton_app.models.StageModel] | cryton_core.cryton_app.models.StageModel

property name: str

property trigger_type: str

property trigger_args: dict

property final_steps: django.db.models.query.QuerySet

property execution_list: django.db.models.query.QuerySet

Returns StageExecutionModel QuerySet. If the latest is needed, use '.latest()' on result. :return: QuerySet of StageExecutionModel

delete()

static filter(**kwargs) → django.db.models.query.QuerySet

Parameters
kwargs – dict of parameters to filter by

Returns
QuerySet of StageModel

static _dfs_reachable(*visited: set*, *completed: set*, *nodes_pairs: dict*, *node: str*) → set

Depth first search of reachable nodes

Parameters

- visited – set of visited nodes
- completed – set of completed nodes
- nodes_pairs – stage successors representation ({parent: [successors]})
- node – current node

Returns

static validate(*stage_dict*, *dynamic: bool = False*) → bool

Check if Stage's dictionary is valid :param stage_dict: Stage information :param dynamic: If the Plan is static or dynamic :raises

exceptions.StageValidationError exceptions.StepValidationError

:return True if Stage's dictionary is valid

add_dependency(*dependency_id: int*) → int

Create dependency object :param dependency_id: Stage ID :return: ID of the dependency object

class cryton_core.lib.models.stage.StageExecution(**kwargs)

property model: Type[cryton_core.cryton_app.models.StageExecutionModel] | cryton_core.cryton_app.models.StageExecutionModel

property state: str

property aps_job_id: str

property start_time: datetime.datetime | None

property schedule_time: datetime.datetime | None

property pause_time: datetime.datetime | None

property finish_time: datetime.datetime | None

property trigger_id: str

property trigger: cryton_core.lib.triggers.TriggerDelta | cryton_core.lib.triggers.TriggerHTTP | cryton_core.lib.triggers.TriggerMSF | cryton_core.lib.triggers.TriggerDateTime

property all_steps_finished: bool

property all_dependencies_finished: bool

delete()

static filter(**kwargs) → django.db.models.query.QuerySet

Get list of StageExecutionModel according to no or specified conditions :param kwargs: dict of parameters to filter by :return: Desired QuerySet

_prepare_executions()

Prepare execution for each Step. :return: None

static _run_step_executions(*step_executions: List[*cryton_core.lib.models.step.StepExecution*])

Evenly distribute Step executions and run each batch in a Process. :param step_executions: Step executions to be distributed into Processes :return: None

execute() → None

Check if all requirements for execution are met, get init steps and execute them. :return: None

validate_modules() → None

Check if module is present and module args are correct for each Step

report() → dict

kill() → None

Kill current StageExecution and its StepExecutions. :return: None

execute_subjects_to_dependency() → None

Execute WAITING StageExecution subjects to specified StageExecution dependency. :return: None

re_execute(*immediately: bool = False*) → None

Reset execution data and re-execute StageExecution. :return: None

reset_execution_data() → None

Reset changeable data to defaults and reset StepExecutions. :return: None

finish() → None

Finish execution - update necessary variables, execute dependencies, and stop trigger. :return: None

cryton_core.lib.models.stage.execution(*execution_id: int*) → None

Create StageExecution object and call its execute method :param execution_id: desired StageExecution's ID :return: None

cryton_core.lib.models.step

Module Contents

Classes

StepReport	
Step	
StepWorkerExecute	
StepEmpireAgentDeploy	
StepEmpireExecute	
StepExecution	
StepExecutionWorkerExecute	
StepExecutionEmpireExecute	
StepExecutionEmpireAgentDeploy	
StepTypeMeta	Overrides base metaclass of Enum in order to support custom exception when accessing not present item.
StepType	Keys according to cryton_core.lib.util.constants
StepExecutionType	Keys according to cryton_core.lib.util.constants

class cryton_core.lib.models.step.StepReport

id: int

step_name: str

state: str

start_time: datetime.datetime

finish_time: datetime.datetime

output: str

serialized_output: dict

result: str

valid: bool

class cryton_core.lib.models.step.Step(**kwargs)

property model: Type[cryton_core.cryton_app.models.StepModel] | cryton_core.cryton_app.models.StepModel

property stage_model_id: Type[int] | int

property name: str

property step_type: str

property is_init: bool

property is_final: bool

property arguments: dict

property output_prefix: str

property execution_stats_list: django.db.models.query.QuerySet

Returns StepExecutionStatsModel QuerySet. If the latest is needed, use '.latest()' on result. :return: QuerySet of StepExecutionStatsModel

property parents: django.db.models.query.QuerySet

property successors: django.db.models.query.QuerySet

delete()

static filter(***kwargs*) → django.db.models.query.QuerySet

Get list of StepInstances according to no or specified conditions :param kwargs: dict of parameters to filter by :return:

classmethod validate(*step_dict*) → bool

Validate a step dictionary :raises:

exceptions.StepValidationError

Returns
True

classmethod validate_ssh_connection(*ssh_connection_dict*)

Validate ssh_connection dictionary

Raises
exceptions.StepValidationError

Returns
True

classmethod validate_next_parameter(*next_dict*)

add_successor(*successor_id: int*, *successor_type: str*, *successor_value: str | None*) → int

Check if successor's parameters are correct and save it. :param successor_id: :param successor_type: One of valid types :param successor_value: One of valid values for specified type :raises:

InvalidSuccessorType InvalidSuccessorValue

Returns
SuccessorModel id

class cryton_core.lib.models.step.StepWorkerExecute(***kwargs*)

Bases: Step

classmethod validate(*step_arguments*)

Validate arguments in 'worker/execute' step_type

class cryton_core.lib.models.step.StepEmpireAgentDeploy(***kwargs*)

Bases: Step

classmethod validate(*step_arguments*)

Validate arguments in 'empire/agent-deploy' step type

class cryton_core.lib.models.step.StepEmpireExecute(**kwargs)

Bases: Step

classmethod validate(*step_arguments*)

Validate arguments in 'empire/execute' step type

class cryton_core.lib.models.step.StepExecution(**kwargs)

property model: Type[cryton_core.cryton_app.models.StepExecutionModel] | cryton_core.cryton_app.models.StepExecutionModel

property state: str

property result: str

property output: str

property serialized_output: list | dict

property start_time: datetime.datetime | None

property finish_time: datetime.datetime | None

property valid: bool

property parent_id: int

delete()

static filter(**kwargs) → django.db.models.query.QuerySet

Get list of StepExecutionStatsModel according to specified conditions. :param kwargs: dict of parameters to filter by :return: Desired QuerySet

validate()

Validates Step Execution.

Returns

save_output(*step_output: dict*) → None

Save Step execution output to StepExecutionModel.

Parameters

step_output – dictionary with keys: output, serialized_output

Returns

None

_update_dynamic_variables(*arguments: dict*, *parent_step_ex_id: int | None*) → dict

Update dynamic variables in mod_args (even with special $parent prefix) :param arguments: arguments that should be updated :param parent_step_ex_id: ID of the parent step of the current step execution :return: Arguments updated for dynamic variables

static _update_arguments_with_execution_variables(*arguments: dict*, *execution_variables: list*) → dict

Fill Jinja variables in the arguments with execution variables. :param arguments: Arguments to fill :param execution_variables: Execution variables to fill the template (arguments) with :return: Filled arguments

update_step_arguments(*arguments: dict*, *plan_execution_id: Type[int]*) → dict

Update Step arguments with execution and dynamic variables. :param arguments: Arguments to be updated :param plan_execution_id: ID of the parent step of the current step execution :return: Updated arguments with execution and dynamic variables

get_msf_session_id(*step_obj: Step*, *plan_execution_id: Type[int]*) → str | None

Check if there should be used any session stored in database and get its session_id. :param step_obj: Step instance of current Step Execution :param plan_execution_id: Plan Execution ID of current Step Execution. :return: MSF Session id

send_step_execution_request(*rabbit_channel: amqpstorm.Channel*, *message_body: dict*, *reply_queue: str*, *target_queue: str*) → None

Sends RPC request to execute step using RabbitMQ. :param rabbit_channel: Rabbit channel :param message_body: data that should be sent to worker :param reply_queue: Queue that worker should reply to :param target_queue: Queue on which should data be sent to worker(for example) :return: None

_prepare_execution() → [Type[int], *cryton_core.lib.models.worker.Worker*]

Execute necessary actions and return variables needed for individual execution of each type of StepExecution. :return: Plan execution id and worker instance for current Plan execution

execute()

Execute current Step Execution.

Returns

report() → dict

Generate report containing output from Step Execution. :return: Step Execution report

get_successors_to_execute() → django.db.models.query.QuerySet

Get Successors based on evaluated dependency.

Returns
QuerySet of StepModel objects

ignore() → None

Set IGNORE state to Step Execution and to all of its successors.

Returns
None

postprocess(*ret_vals: dict*) → None

Perform necessary things after executing Step like creating named sessions, update state, update successors and save Step Execution Output.

Parameters
ret_vals – output from Step Execution

Returns
None

ignore_successors() → None

Ignor/skip all successor Steps of current Step Execution. :return: None

execute_successors() → None

Execute all successors of current Step Execution. :return: None

pause_successors() → None

Pause successor Steps of current Step Execution. :return: None

kill()

Kill current Step Execution on Worker.

Returns

Dictionary containing return_code and output

re_execute() → None

Reset execution data and re-execute StepExecution. :return: None

reset_execution_data() → None

Reset changeable data to defaults. :return: None

process_error_state() → None

If an error state is set, ignore successors and send an event that an error occurred. :return: None

class cryton_core.lib.models.step.StepExecutionWorkerExecute(***kwargs*)

Bases: StepExecution

validate() → bool

Validate cryton attack module arguments.

Returns

execute(*rabbit_channel: amqpstorm.Channel = None*) → None

Execute Step on worker specified in execution stats. :param rabbit_channel: Rabbit channel :return: None

class cryton_core.lib.models.step.StepExecutionEmpireExecute(***kwargs*)

Bases: StepExecution

execute(*rabbit_channel: amqpstorm.Channel = None*) → None

Execute Step on worker specified in execution stats. :param rabbit_channel: Rabbit channel :return: None

validate()

Validates StepExecutionEmpireExecute.

Returns

class cryton_core.lib.models.step.StepExecutionEmpireAgentDeploy(***kwargs*)

Bases: StepExecution

execute(*rabbit_channel: amqpstorm.Channel = None*) → None

Execute Step on worker specified in execution stats. :param rabbit_channel: Rabbit channel :return: None

validate()

Validates StepExecutionEmpireAgentDeploy.

Returns

class cryton_core.lib.models.step.StepTypeMeta

Bases: enum.EnumMeta

Overrides base metaclass of Enum in order to support custom exception when accessing not present item.

step_types

__getitem__(*item*)

class cryton_core.lib.models.step.StepType

Bases: enum.Enum

Keys according to cryton_core.lib.util.constants

empire_agent_deploy

worker_execute

empire_execute

class cryton_core.lib.models.step.StepExecutionType

Bases: enum.Enum

Keys according to cryton_core.lib.util.constants

empire_agent_deploy

worker_execute

empire_execute

cryton_core.lib.models.worker

Module Contents

Classes

Worker

class cryton_core.lib.models.worker.Worker(**kwargs)

property model: Type[cryton_core.cryton_app.models.WorkerModel] | cryton_core.cryton_app.models.WorkerModel

property name: str

property description: str

property state: str

property attack_q_name

property agent_q_name

property control_q_name

delete()

healthcheck() → bool

Check if Worker is consuming its attack queue :return:

prepare_rabbit_queues() → None

Declare Rabbit queues in case the Worker is not online yet. :return: None

cryton_core.lib.services

Submodules

cryton_core.lib.services.listener

Module Contents

Classes

ChannelConsumer
Consumer
Listener

class cryton_core.lib.services.listener.ChannelConsumer(*identifier: int*, *connection: amqpstorm.Connection*, *queues: dict*)

start()

class cryton_core.lib.services.listener.Consumer(*identifier: int*, *queues: dict*, *channel_consumer_count: int*)

start() → None

Establish connection, start channel consumers in thread and keep self alive. :return: None

stop() → None

Stop Consumer (self). Close connection and its channels. :return: None

_update_connection() → bool

Check existing connection for errors and optionally reconnect. :return: True if connection was updated :raises: amqpstorm.AMQPConnectionError if connection can't be established

_start_channel_consumers() → None

Start channel consumers in threads. :return: None

class cryton_core.lib.services.listener.Listener

start() → None

Start consumers and keep self alive. :return: None

stop() → None

Stop Listener and it's Consumers. :return: None

_create_consumers() → None

Create consumers and their processes. :return: None

_start_consumers() → None

Start each consumer in process. :return: None

step_response_callback(*message: amqpstorm.Message*) → None

Callback for processing Step execution responses. :param message: Received RabbitMQ message :return: None

static event_callback(*message: amqpstorm.Message*) → None

Callback for processing events. :param message: Received RabbitMQ message :return: None

control_request_callback(*message: amqpstorm.Message*) → None

Callback for processing control requests. :param message: Received RabbitMQ message :return: None

static _get_correlation_event(*correlation_id: str*) → *cryton_core.cryton_app.models.CorrelationEventModel*

Find correlation event. :param correlation_id: ID of the correlation event :return: correlation event :raises: CorrelationEventModel.DoesNotExist

static _handle_pausing(*step_ex_obj:* cryton_core.lib.models.step.StepExecution) → None

Check for PAUSED states. :param step_ex_obj: StepExecution object to check :return: None

static _send_response(*original_message: amqpstorm.Message*, *message_body: dict*) → None

Send a response to *reply_to* from the original message. :param original_message: Received message :param message_body: Message content :return: None

cryton_core.lib.services.scheduler

Module Contents

Classes

SchedulerService

Attributes

SCHED_MAX_THREADS
SCHED_MAX_PROCESSES
JOB_MAX_INSTANCES

cryton_core.lib.services.scheduler.SCHED_MAX_THREADS = 20

cryton_core.lib.services.scheduler.SCHED_MAX_PROCESSES = 5

cryton_core.lib.services.scheduler.JOB_MAX_INSTANCES = 3

class cryton_core.lib.services.scheduler.SchedulerService

__del__()

exposed_add_job(*execute_function: str*, *function_args: list*, *start_time: datetime.datetime*) → str

Parameters

- execute_function – Function/method to be scheduled

- function_args – Function arguments
- start_time – Function start time

Returns
Scheduled job ID

exposed_add_repeating_job(*execute_function: str*, *seconds: int*) → str

Parameters

- execute_function – Function/method to be scheduled
- seconds – Function interval in seconds

Returns
Scheduled job ID

exposed_reschedule_job(*job_id: str*)

exposed_pause_job(*job_id: str*)

exposed_resume_job(*job_id: str*)

exposed_remove_job(*job_id: str*)

exposed_get_job(*job_id: str*)

exposed_get_jobs()

exposed_pause_scheduler()

exposed_resume_scheduler()

cryton_core.lib.triggers

Submodules

cryton_core.lib.triggers.trigger_base

Module Contents

Classes

TriggerBase
TriggerTime
TriggerWorker

class cryton_core.lib.triggers.trigger_base.TriggerBase(*stage_execution*)

arg_schema

start() → None

stop() → None

pause() → None

unpause() → None

Unpause stage execution. :return: None

class cryton_core.lib.triggers.trigger_base.TriggerTime(*stage_execution*)

Bases: TriggerBase

start() → None

Runs schedule() method. :return: None

stop() → None

Runs unschedule() method. :return: None

schedule() → None

Schedule stage execution. :return: None

unschedule() → None

Unschedule StageExecution from a APScheduler. :raises:

ConnectionRefusedError

Returns

None

pause() → None

Pause stage execution. :return: None

_create_schedule_time() → datetime.datetime

class cryton_core.lib.triggers.trigger_base.TriggerWorker(*stage_execution*)

Bases: TriggerBase

_rpc_start(*event_v: dict*) → None

Start trigger's listener on worker. :param event_v: Special trigger parameters for listener on worker :return: None

_rpc_stop() → None

Stop trigger's listener on worker. :return: None

pause() → None

Pause stage execution. :return: None

cryton_core.lib.triggers.trigger_datetime

Module Contents

Classes

TriggerDateTime

class cryton_core.lib.triggers.trigger_datetime.TriggerDateTime(*stage_execution*)

Bases: cryton_core.lib.triggers.trigger_base.TriggerTime

arg_schema

_create_schedule_time() → datetime.datetime

Create Stage's start time. :return: Stage's start time

cryton_core.lib.triggers.trigger_delta

Module Contents

Classes

TriggerDelta

class cryton_core.lib.triggers.trigger_delta.TriggerDelta(*stage_execution*)

Bases: cryton_core.lib.triggers.trigger_base.TriggerTime

arg_schema

start()

Runs schedule() method. :return: None

_create_schedule_time() → datetime.datetime

Create Stage's start time. :return: Stage's start time

cryton_core.lib.triggers.trigger_http

Module Contents

Classes

TriggerHTTP

class cryton_core.lib.triggers.trigger_http.TriggerHTTP(*stage_execution*)

Bases: cryton_core.lib.triggers.trigger_base.TriggerWorker

arg_schema

start() → None

Start HTTP listener. :return: None

stop() → None

Stop HTTP listener. :return: None

cryton_core.lib.triggers.trigger_msf

Module Contents

Classes

TriggerMSF

class cryton_core.lib.triggers.trigger_msf.TriggerMSF(*stage_execution*)

Bases: cryton_core.lib.triggers.trigger_base.TriggerWorker

arg_schema

start() → None

Start MSF listener. :return: None

stop() → None

Stop MSF listener. :return: None

Package Contents

Classes

TriggerBase	
TriggerDelta	
TriggerHTTP	
TriggerMSF	
TriggerDateTime	
TriggerTypeMeta	Overrides base metaclass of Enum in order to support custom exception when accessing not present item.
TriggerType	Keys according to cryton_core.lib.util.constants

exception cryton_core.lib.triggers.TriggerTypeDoesNotExist(*trigger_type: str*, *supported_triggers: List*)

Bases: Error

Exception raised if the trigger type doesn't exist

class cryton_core.lib.triggers.TriggerBase(*stage_execution*)

arg_schema

start() → None

stop() → None

pause() → None

unpause() → None

Unpause stage execution. :return: None

class cryton_core.lib.triggers.TriggerDelta(*stage_execution*)

Bases: cryton_core.lib.triggers.trigger_base.TriggerTime

arg_schema

start()

Runs schedule() method. :return: None

_create_schedule_time() → datetime.datetime

Create Stage's start time. :return: Stage's start time

class cryton_core.lib.triggers.TriggerHTTP(*stage_execution*)

Bases: cryton_core.lib.triggers.trigger_base.TriggerWorker

arg_schema

start() → None

Start HTTP listener. :return: None

stop() → None

Stop HTTP listener. :return: None

class cryton_core.lib.triggers.TriggerMSF(*stage_execution*)

Bases: cryton_core.lib.triggers.trigger_base.TriggerWorker

arg_schema

start() → None

Start MSF listener. :return: None

stop() → None

Stop MSF listener. :return: None

class cryton_core.lib.triggers.TriggerDateTime(*stage_execution*)

Bases: cryton_core.lib.triggers.trigger_base.TriggerTime

arg_schema

_create_schedule_time() → datetime.datetime

Create Stage's start time. :return: Stage's start time

class cryton_core.lib.triggers.TriggerTypeMeta

Bases: enum.EnumMeta

Overrides base metaclass of Enum in order to support custom exception when accessing not present item.

__getitem__(*item*)

class cryton_core.lib.triggers.TriggerType

Bases: enum.Enum

Keys according to cryton_core.lib.util.constants

delta

HTTPListener

MSFListener

datetime

cryton_core.lib.util

Submodules

cryton_core.lib.util.constants

Module Contents

cryton_core.lib.util.constants.RESULT_OK = 'OK'

cryton_core.lib.util.constants.RESULT_FAIL = 'FAIL'

cryton_core.lib.util.constants.RESULT_UNKNOWN = 'UNKNOWN'

cryton_core.lib.util.constants.CODE_OK = 0

cryton_core.lib.util.constants.CODE_FAIL

cryton_core.lib.util.constants.CODE_ERROR

cryton_core.lib.util.constants.CODE_TERMINATED

cryton_core.lib.util.constants.RETURN_CODE = 'return_code'

cryton_core.lib.util.constants.RETURN_VALUE = 'return_value'

cryton_core.lib.util.constants.RESULT = 'result'

cryton_core.lib.util.constants.OUTPUT = 'output'

cryton_core.lib.util.constants.SERIALIZED_OUTPUT = 'serialized_output'

cryton_core.lib.util.constants.ANY = 'any'

cryton_core.lib.util.constants.REGEX_TYPES

cryton_core.lib.util.constants.ARGUMENTS = 'arguments'

cryton_core.lib.util.constants.STEP_TYPE = 'step_type'

cryton_core.lib.util.constants.NEXT = 'next'

cryton_core.lib.util.constants.MODULE = 'module'

cryton_core.lib.util.constants.MODULE_ARGUMENTS = 'module_arguments'

cryton_core.lib.util.constants.LISTENER_NAME = 'listener_name'

cryton_core.lib.util.constants.LISTENER_PORT = 'listener_port'

cryton_core.lib.util.constants.LISTENER_OPTIONS = 'listener_options'

cryton_core.lib.util.constants.LISTENER_TYPE = 'listener_type'

cryton_core.lib.util.constants.STAGER_TYPE = 'stager_type'

cryton_core.lib.util.constants.STAGER_OPTIONS = 'stager_options'

cryton_core.lib.util.constants.AGENT_NAME = 'agent_name'

cryton_core.lib.util.constants.USE_AGENT = 'use_agent'

cryton_core.lib.util.constants.SHELL_COMMAND = 'shell_command'

cryton_core.lib.util.constants.SESSION_ID = 'session_id'

cryton_core.lib.util.constants.CREATE_NAMED_SESSION = 'create_named_session'

cryton_core.lib.util.constants.USE_NAMED_SESSION = 'use_named_session'

cryton_core.lib.util.constants.USE_ANY_SESSION_TO_TARGET = 'use_any_session_to_target'

cryton_core.lib.util.constants.SSH_CONNECTION = 'ssh_connection'

cryton_core.lib.util.constants.STEP_TYPE_WORKER_EXECUTE = 'worker/execute'

cryton_core.lib.util.constants.STEP_TYPE_DEPLOY_AGENT = 'empire/agent-deploy'

cryton_core.lib.util.constants.STEP_TYPE_EMPIRE_EXECUTE = 'empire/execute'

cryton_core.lib.util.constants.STEP_TYPES_LIST

cryton_core.lib.util.constants.DELTA = 'delta'

cryton_core.lib.util.constants.DATETIME = 'datetime'

cryton_core.lib.util.constants.HTTP_LISTENER = 'HTTPListener'

cryton_core.lib.util.constants.MSF_LISTENER = 'MSFListener'

cryton_core.lib.util.constants.RETURN_CODE_ENUM

cryton_core.lib.util.constants.VALID_SUCCESSOR_TYPES

cryton_core.lib.util.constants.VALID_SUCCESSOR_RESULTS

cryton_core.lib.util.constants.SUCCESSOR_TYPES_WITHOUT_ANY

cryton_core.lib.util.constants.EVENT_T = 'event_t'

cryton_core.lib.util.constants.EVENT_V = 'event_v'

cryton_core.lib.util.constants.ACK_QUEUE = 'ack_queue'

cryton_core.lib.util.constants.REPLY_TO = 'reply_to'

cryton_core.lib.util.constants.EVENT_ACTION = 'action'

cryton_core.lib.util.constants.TRIGGER_TYPE = 'trigger_type'

cryton_core.lib.util.constants.TRIGGER_ID = 'trigger_id'

cryton_core.lib.util.constants.EVENT_VALIDATE_MODULE = 'VALIDATE_MODULE'

cryton_core.lib.util.constants.EVENT_LIST_MODULES = 'LIST_MODULES'

cryton_core.lib.util.constants.EVENT_LIST_SESSIONS = 'LIST_SESSIONS'

cryton_core.lib.util.constants.EVENT_KILL_STEP_EXECUTION = 'KILL_STEP_EXECUTION'

cryton_core.lib.util.constants.EVENT_HEALTH_CHECK = 'HEALTH_CHECK'

cryton_core.lib.util.constants.EVENT_ADD_TRIGGER = 'ADD_TRIGGER'

cryton_core.lib.util.constants.EVENT_REMOVE_TRIGGER = 'REMOVE_TRIGGER'

cryton_core.lib.util.constants.EVENT_TRIGGER_STAGE = 'TRIGGER_STAGE'

cryton_core.lib.util.constants.EVENT_STEP_EXECUTION_ERROR = 'STEP_EXECUTION_ERROR'

cryton_core.lib.util.constants.EVENT_UPDATE_SCHEDULER = 'UPDATE_SCHEDULER'

cryton_core.lib.util.constants.ADD_REPEATING_JOB = 'add_repeating_job'

cryton_core.lib.util.constants.ADD_JOB = 'add_job'

cryton_core.lib.util.constants.REMOVE_JOB = 'remove_job'

cryton_core.lib.util.constants.RESUME_SCHEDULER = 'resume_scheduler'

cryton_core.lib.util.constants.PAUSE_SCHEDULER = 'pause_scheduler'

cryton_core.lib.util.constants.GET_JOBS = 'get_jobs'

cryton_core.lib.util.constants.RESUME_JOB = 'resume_job'

cryton_core.lib.util.constants.PAUSE_JOB = 'pause_job'

cryton_core.lib.util.constants.RESCHEDULE_JOB = 'reschedule_job'

cryton_core.lib.util.constants.TIME_FORMAT = '%Y-%m-%dT%H:%M:%SZ'

cryton_core.lib.util.constants.TIME_FORMAT_DETAILED = '%Y-%m-%dT%H:%M:%S.%fZ'

cryton_core.lib.util.constants.LOGGER_CRYTON_PRODUCTION = 'cryton-core'

cryton_core.lib.util.constants.LOGGER_CRYTON_DEBUG = 'cryton-core-debug'

cryton_core.lib.util.constants.LOGGER_CRYTON_TESTING = 'cryton-core-testing'

cryton_core.lib.util.constants.VALID_CRYTON_LOGGERS

cryton_core.lib.util.constants.BLOCK_START_STRING = "'{%"

cryton_core.lib.util.constants.BLOCK_END_STRING = "%}'"

cryton_core.lib.util.constants.VARIABLE_START_STRING = "'{{"

cryton_core.lib.util.constants.VARIABLE_END_STRING = "}}'"

cryton_core.lib.util.constants.COMMENT_START_STRING = "'{#"

cryton_core.lib.util.constants.COMMENT_END_STRING = "#}'"

cryton_core.lib.util.creator

Module Contents

Functions

create_plan(→ int)	Check if Plan structure is correct and add it to DB.
create_stage(→ int)	Add Stage to DB.
create_step(→ int)	Add Step to DB.
create_successor(parent_step, stage_id, ...)	Add successor and its links between parent and successor Step to DB.
create_output_mapping(output_mapping, step_model_id)	Add output mapping for Step to DB.
create_worker(→ int)	Update prefix and add Worker to DB.

cryton_core.lib.util.creator.create_plan(*template: dict*) → int

Check if Plan structure is correct and add it to DB. :param template: Plan from file ({"plan": ... }) :return: Added Plan object ID :raises

exceptions.ValidationError exceptions.PlanCreationFailedError

cryton_core.lib.util.creator.create_stage(*stage_dict: dict*, *plan_model_id: int*) → int

Add Stage to DB. :param stage_dict: Stage dictionary :param plan_model_id: Plan ID :return: Added Stage object ID :raises

exceptions.StageCreationFailedError

cryton_core.lib.util.creator.create_step(*step_dict: dict*, *stage_model_id: int*) → int

Add Step to DB. :param step_dict: Step dictionary :param stage_model_id: Stage ID :return: Added Step object ID

cryton_core.lib.util.creator.create_successor(*parent_step:* cryton_core.lib.models.step.Step, *stage_id: int*, *successor_name: str*, *successor_type: str*, *successor_values: list | str*)

Add successor and its links between parent and successor Step to DB. :param parent_step: Parent Step :param stage_id: Stage ID :param successor_name: Successor's name :param successor_type: Successor's type :param successor_values: Successor's values (or just one) :return: None :raises:

exceptions.InvalidSuccessorType exceptions.InvalidSuccessorValue exceptions.SuccessorCreationFailedError

cryton_core.lib.util.creator.create_output_mapping(*output_mapping: dict*, *step_model_id: int*)

Add output mapping for Step to DB. :param output_mapping: Output mapping :param step_model_id: Step ID :return: None

cryton_core.lib.util.creator.create_worker(*name: str*, *description: str*, *force: bool = False*) → int

Update prefix and add Worker to DB. :param name: Worker's name :param description: Worker's description :param force: If True, name won't have to be unique :return: Added Worker object ID

cryton_core.lib.util.event

Module Contents

Classes

Event

class cryton_core.lib.util.event.Event(*event_details: dict*)

trigger_stage() → None

Process trigger trying to start Stage execution. :return: None

update_scheduler() → int

Process scheduler control request. :return: -1 if failed, otherwise relevant data (e.g. ID of scheduled job)

handle_finished_step() → None

Check for FINISHED states. :return: None

cryton_core.lib.util.exceptions

Module Contents

exception cryton_core.lib.util.exceptions.Error

Bases: Exception

Base class for exceptions in this module.

exception cryton_core.lib.util.exceptions.ValidationError(*message: dict | Exception | str*)

Bases: Error

Exception raised for errors in the validation process.

exception cryton_core.lib.util.exceptions.PlanValidationError(*message: Exception | str*, *plan_name: str = None*)

Bases: ValidationError

Exception raised for errors in the Plan validation process.

exception cryton_core.lib.util.exceptions.DuplicateNameInPlan(*unique_argument: str*, *duplicate_name: str*, *plan_name: str*)

Bases: PlanValidationError

Exception raised when multiple instances have the same name in one Plan.

exception cryton_core.lib.util.exceptions.StageValidationError(*message: Exception | str*, *stage_name: str = None*)

Bases: ValidationError

Exception raised for errors in the Stage validation process.

exception cryton_core.lib.util.exceptions.StepValidationError(*message: Exception | str, step_name: str = None*)

Bases: ValidationError

Exception raised for errors in the Step validation process.

exception cryton_core.lib.util.exceptions.UserInputError(*message: Exception | str, user_input: str | int = None*)

Bases: Error

Exception raised for errors when user inputs an invalid input.

exception cryton_core.lib.util.exceptions.ObjectDoesNotExist

Bases: Error

Exception raised if trying to use an object that doesn't exist.

exception cryton_core.lib.util.exceptions.RunObjectDoesNotExist(*run_id: int = None, plan_id: int = None, stage_id: int = None, step_id: int = None, worker_id: int = None*)

Bases: ObjectDoesNotExist

Exception raised if trying to use a Run object that doesn't exist.

exception cryton_core.lib.util.exceptions.PlanObjectDoesNotExist(*plan_id: int = None*)

Bases: ObjectDoesNotExist

Exception raised if trying to use a Plan object that doesn't exist.

exception cryton_core.lib.util.exceptions.StageObjectDoesNotExist(*message: Exception | str, stage_id: int = None, plan_id: int = None*)

Bases: ObjectDoesNotExist

Exception raised if trying to use a Stage object that doesn't exist.

exception cryton_core.lib.util.exceptions.StageExecutionObjectDoesNotExist(*message: Exception | str, stage_id: int = None, plan_id: int = None*)

Bases: ObjectDoesNotExist

Exception raised if trying to use a StageExecution object that doesn't exist.

exception cryton_core.lib.util.exceptions.StepObjectDoesNotExist(*message: Exception | str, step_id: int = None, step_name: str = None, stage_id: int | Type[int] = None, plan_id: int = None*)

Bases: ObjectDoesNotExist

Exception raised if trying to use a Step object that doesn't exist.

exception cryton_core.lib.util.exceptions.StepExecutionObjectDoesNotExist(*message: Exception | str = 'StepExecution not found', step_execution_id: str = None*)

Bases: Error

Exception raised if trying to set invalid type of successor.

exception cryton_core.lib.util.exceptions.SuccessorObjectDoesNotExist(*message: Exception | str, step_id: int = None*)

Bases: ObjectDoesNotExist

Exception raised if trying to use a Successor object that doesn't exist.

exception cryton_core.lib.util.exceptions.WorkerObjectDoesNotExist(*worker_id: int*)

Bases: ObjectDoesNotExist

Exception raised if trying to use a Worker object that doesn't exist.

exception cryton_core.lib.util.exceptions.SessionObjectDoesNotExist(*message: Exception | str, session_name: str = None, session_id: int = None, plan_execution_id: Type[int] | int = None, step_id: int = None, plan_id: int = None, target=None, target_id=None*)

Bases: ObjectDoesNotExist

Exception raised if trying to use a Session object that doesn't exist.

exception cryton_core.lib.util.exceptions.SessionIsNotOpen(*message: Exception | str, session_name: str, plan_execution_id: Type[int] | int, step_id: int*)

Bases: Error

Exception raised if the desired session isn't open.

exception cryton_core.lib.util.exceptions.ArgumentsObjectDoesNotExist(*message: Exception | str, step_id: int = None*)

Bases: ObjectDoesNotExist

Exception raised if trying to use an Arguments object that doesn't exist.

exception cryton_core.lib.util.exceptions.InvalidStateError(*message: Exception | str, execution_id: int, state: str, allowed_states: list*)

Bases: Error

Exception raised if an invalid state is detected.

exception cryton_core.lib.util.exceptions.RunInvalidStateError(*message: Exception | str, execution_id: int, state: str, allowed_states: list*)

Bases: InvalidStateError

Exception raised if Run's invalid state is detected.

exception cryton_core.lib.util.exceptions.PlanInvalidStateError(*message: Exception | str, execution_id: int, state: str, allowed_states: list*)

Bases: InvalidStateError

Exception raised if PlanExecution's invalid state is detected.

exception cryton_core.lib.util.exceptions.StageInvalidStateError(*message: Exception | str, execution_id: int, state: str, allowed_states: list*)

Bases: InvalidStateError

Exception raised if StageExecution's invalid state is detected.

exception cryton_core.lib.util.exceptions.StepInvalidStateError(*message: Exception | str, execution_id: int, state: str, allowed_states: list*)

Bases: InvalidStateError

Exception raised if StepExecution's invalid state is detected.

exception cryton_core.lib.util.exceptions.StateTransitionError(*message: Exception | str, execution_id: int, state_from: str, state_to: str, allowed_transitions: list*)

Bases: Error

Exception raised if an invalid state transition is made.

exception cryton_core.lib.util.exceptions.RunStateTransitionError(*message: Exception | str*, *execution_id: int*, *state_from: str*, *state_to: str*, *allowed_transitions: list*)

Bases: StateTransitionError

Raised if an invalid Run's state transition is made.

exception cryton_core.lib.util.exceptions.PlanStateTransitionError(*message: Exception | str*, *execution_id: int*, *state_from: str*, *state_to: str*, *allowed_transitions: list*)

Bases: StateTransitionError

Raised if an invalid PlanExecution's state transition is made.

exception cryton_core.lib.util.exceptions.StageStateTransitionError(*message: Exception | str*, *execution_id: int*, *state_from: str*, *state_to: str*, *allowed_transitions: list*)

Bases: StateTransitionError

Raised if an invalid StageExecution's state transition is made.

exception cryton_core.lib.util.exceptions.StepStateTransitionError(*message: Exception | str*, *execution_id: int*, *state_from: str*, *state_to: str*, *allowed_transitions: list*)

Bases: StateTransitionError

Raised if an invalid StepExecution's state transition is made.

exception cryton_core.lib.util.exceptions.UnexpectedValue(*message: Exception | str*, *wrong_value: str | int = None*)

Bases: Error

Raised if an invalid value is used.

exception cryton_core.lib.util.exceptions.WrongParameterError(*message: Exception | str = 'Wrong parameter name'*, *param_name: str = None*)

Bases: Error

Exception raised if trying to filter in model by wrong parameter.

exception cryton_core.lib.util.exceptions.ParameterMissingError(*message: Exception | str = 'Missing parameter'*, *param_name: str = None*)

Bases: Error

Exception raised if compulsory parameter is missing.

exception cryton_core.lib.util.exceptions.DependencyDoesNotExist(*message: Exception | str*, *stage_name: str = None*)

Bases: Error

Exception raised if could not create dependency between Stages.

exception cryton_core.lib.util.exceptions.InvalidSuccessorType(*message: Exception | str = 'Wrong successor type'*, *successor_type: str = None*)

Bases: Error

Exception raised if trying to set invalid type of successor.

exception cryton_core.lib.util.exceptions.InvalidSuccessorValue(*message: Exception | str = 'Wrong successor value', successor_value: str = None*)

Bases: Error

Exception raised if trying to set invalid value for successor.

exception cryton_core.lib.util.exceptions.PlanExecutionDoesNotExist(*message: Exception | str = 'PlanExecution not found', plan_execution_id: str = None*)

Bases: Error

Exception raised if Execution does not exist.

exception cryton_core.lib.util.exceptions.CreationFailedError(*message: dict*)

Bases: Error

Exception raised if object creation failed.

exception cryton_core.lib.util.exceptions.PlanCreationFailedError(*message: Exception | str, plan_name: str = None*)

Bases: CreationFailedError

Exception raised if Plan creation failed.

exception cryton_core.lib.util.exceptions.StageCreationFailedError(*message: Exception | str, stage_name: str = None*)

Bases: CreationFailedError

Exception raised if Stage creation failed.

exception cryton_core.lib.util.exceptions.StepCreationFailedError(*message: Exception | str, step_name: str = None*)

Bases: CreationFailedError

Exception raised if Step creation failed.

exception cryton_core.lib.util.exceptions.SuccessorCreationFailedError(*message: Exception | str, successor_name: str = None*)

Bases: CreationFailedError

Exception raised if Successor creation failed.

exception cryton_core.lib.util.exceptions.PlanExecutionCreationFailedError(*message: Exception | str = 'Bad argument', param_name: str = None, param_type: str = None*)

Bases: CreationFailedError

Exception raised if PlanExecution creation failed.

exception cryton_core.lib.util.exceptions.RunCreationFailedError(*message: Exception | str*)

Bases: CreationFailedError

Exception raised if Run creation failed.

exception cryton_core.lib.util.exceptions.RabbitError(*message: Exception | str*)

Bases: Error

Exception raised when there is some problem with RabbitMQ

exception cryton_core.lib.util.exceptions.RpcTimeoutError(*message: Exception | str*)

Bases: RabbitError

Exception raised if the RPC request timeouts

exception cryton_core.lib.util.exceptions.StageCycleDetected(*message: Exception | str*)

Bases: Error

Exception raised if there was Stage cycle detected

exception cryton_core.lib.util.exceptions.TriggerTypeDoesNotExist(*trigger_type: str*, *supported_triggers: List*)

Bases: Error

Exception raised if the trigger type doesn't exist

exception cryton_core.lib.util.exceptions.StepTypeDoesNotExist(*step_type: str*, *supported_step_types: List*)

Bases: Error

Exception raised if the step type doesn't exist

cryton_core.lib.util.logger

Module Contents

Classes

Logger	Default logger
LoggedProcess	Process objects represent activity that is run in a separate process

Attributes

config_dict
logger_object
logger

class cryton_core.lib.util.logger.Logger(*logger_config*)

Default logger

property logger_type

property logger_config

log_handler()

Simple function that takes logs from Processes and handles them instead. :return: None

class cryton_core.lib.util.logger.LoggedProcess(*logg_queue*, **args*, ***kwargs*)

Bases: multiprocessing.Process

Process objects represent activity that is run in a separate process

The class is analogous to *threading.Thread*

cryton_core.lib.util.logger.config_dict

cryton_core.lib.util.logger.logger_object

cryton_core.lib.util.logger.logger

cryton_core.lib.util.rabbit_client

Module Contents

Classes

RpcClient
Client

class cryton_core.lib.util.rabbit_client.RpcClient(*channel: amqpstorm.Channel = None*)

__enter__()

__exit__(*exc_type*, *exc_val*, *exc_tb*)

open() → None

Setup connection, channel, and callback_queue. :return: None

close() → None

Delete the callback_queue, optionally close created channel and connection. :return: None

call(*target_queue: str*, *message_body: dict*, *properties: dict = None*, *custom_reply_queue: str = None*) → dict

Create RPC call and wait for response. :param target_queue: Target RabbitMQ queue to send the message :param message_body: Message contents :param properties: Message properties :param custom_reply_queue: Custom queue to send the reply to (moves self.callback_queue to msg_body["ack_queue"] and is only used for message received acknowledgment) :return: Serialized response :raises: exceptions.RpcTimeoutError

_clean_up() → None

Remove message specific information. :return: None

_create_message(*message_body: dict*, *properties: dict = None*, *custom_reply_queue: str = None*) → amqpstorm.Message

Create message. Optionally use custom reply queue. :param message_body: Message contents :param properties: Message properties :param custom_reply_queue: Custom queue to send the reply to (moves self.callback_queue to msg_body["ack_queue"] and is only used for message received acknowledgment) :return: Rabbit message

_wait_for_response() → None

Wait for response. :return: None :raises: exceptions.RpcTimeoutError

_on_response(*message: amqpstorm.Message*) → None

Check if the correlation_id matches and save the response. :param message: Received RabbitMQ message :return: None

class cryton_core.lib.util.rabbit_client.Client(*channel: amqpstorm.Channel = None*)

__enter__()

__exit__(*exc_type*, *exc_val*, *exc_tb*)

open() → None

Setup connection and channel. :return: None

close() → None

Close created channel and connection. :return: None

send_message(*target_queue: str*, *message_body: dict*, *properties: dict = None*) → None

Declare the target queue and send a message. :param target_queue: Target RabbitMQ queue to send the message :param message_body: Message contents :param properties: Message properties :return: none

cryton_core.lib.util.scheduler_client

Module Contents

Functions

schedule_function(→ str)	Schedule a job
schedule_repeating_function(→ str)	Schedule a job
remove_job(→ int)	Removes s job

cryton_core.lib.util.scheduler_client.schedule_function(*execute_function: callable*, *function_args: list*, *start_time: datetime.datetime*) → str

Schedule a job

Parameters

- execute_function – Function/method to be scheduled
- function_args – Function arguments
- start_time – Start time of function

Returns

ID of the scheduled job

cryton_core.lib.util.scheduler_client.schedule_repeating_function(*execute_function: callable*, *seconds: int*) → str

Schedule a job

Parameters

- execute_function – Function/method to be scheduled

- seconds – Interval in seconds

Returns
ID of the scheduled job

cryton_core.lib.util.scheduler_client.remove_job(*job_id: str*) → int

Removes s job :param job_id: APS job ID :return: 0

cryton_core.lib.util.states

Module Contents

Classes

StateMachine	Base state machine
RunStateMachine	Run state machine
PlanStateMachine	Plan state machine
StageStateMachine	Stage state machine
StepStateMachine	Step state machine

Attributes

PENDING
SCHEDULED
STARTING
RUNNING
PAUSING
PAUSED
FINISHED
IGNORED
ERROR
WAITING
TERMINATED
TERMINATING
AWAITING

continues on next page

Table 2 – continued from previous page

UP
DOWN
WORKER_STATES
WORKER_TRANSITIONS
RUN_STATES
RUN_TRANSITIONS
RUN_PREPARE_STATES
RUN_SCHEDULE_STATES
RUN_UNSCHEDULE_STATES
RUN_RESCHEDULE_STATES
RUN_POSTPONE_STATES
RUN_EXECUTE_STATES
RUN_EXECUTE_NOW_STATES
RUN_PAUSE_STATES
RUN_UNPAUSE_STATES
RUN_KILL_STATES
RUN_PLAN_PAUSE_STATES
PLAN_STATES
PLAN_TRANSITIONS
PLAN_SCHEDULE_STATES
PLAN_EXECUTE_STATES
PLAN_UNSCHEDULE_STATES
PLAN_RESCHEDULE_STATES
PLAN_POSTPONE_STATES
PLAN_PAUSE_STATES
PLAN_UNPAUSE_STATES

continues on next page

Table 2 – continued from previous page

PLAN_KILL_STATES
PLAN_FINAL_STATES
PLAN_STAGE_PAUSE_STATES
PLAN_RUN_EXECUTE_STATES
STAGE_STATES
STAGE_TRANSITIONS
STAGE_EXECUTE_STATES
STAGE_SCHEDULE_STATES
STAGE_UNSCHEDULE_STATES
STAGE_PAUSE_STATES
STAGE_UNPAUSE_STATES
STAGE_KILL_STATES
STAGE_FINAL_STATES
STAGE_PLAN_EXECUTE_STATES
STEP_STATES
STEP_TRANSITIONS
STEP_EXECUTE_STATES
STEP_KILL_STATES
STEP_FINAL_STATES
STEP_STAGE_EXECUTE_STATES

cryton_core.lib.util.states.PENDING = 'PENDING'

cryton_core.lib.util.states.SCHEDULED = 'SCHEDULED'

cryton_core.lib.util.states.STARTING = 'STARTING'

cryton_core.lib.util.states.RUNNING = 'RUNNING'

cryton_core.lib.util.states.PAUSING = 'PAUSING'

cryton_core.lib.util.states.PAUSED = 'PAUSED'

cryton_core.lib.util.states.FINISHED = 'FINISHED'

cryton_core.lib.util.states.IGNORED = 'IGNORED'

cryton_core.lib.util.states.ERROR = 'ERROR'

cryton_core.lib.util.states.WAITING = 'WAITING'

cryton_core.lib.util.states.TERMINATED = 'TERMINATED'

cryton_core.lib.util.states.TERMINATING = 'TERMINATING'

cryton_core.lib.util.states.AWAITING = 'AWAITING'

cryton_core.lib.util.states.UP = 'UP'

cryton_core.lib.util.states.DOWN = 'DOWN'

cryton_core.lib.util.states.WORKER_STATES

cryton_core.lib.util.states.WORKER_TRANSITIONS = [(), ()]

cryton_core.lib.util.states.RUN_STATES

cryton_core.lib.util.states.RUN_TRANSITIONS = [(), (), (), (), (), (), (), (), (), (), (), (), ()]

cryton_core.lib.util.states.RUN_PREPARE_STATES

cryton_core.lib.util.states.RUN_SCHEDULE_STATES

cryton_core.lib.util.states.RUN_UNSCHEDULE_STATES

cryton_core.lib.util.states.RUN_RESCHEDULE_STATES

cryton_core.lib.util.states.RUN_POSTPONE_STATES

cryton_core.lib.util.states.RUN_EXECUTE_STATES

cryton_core.lib.util.states.RUN_EXECUTE_NOW_STATES

cryton_core.lib.util.states.RUN_PAUSE_STATES

cryton_core.lib.util.states.RUN_UNPAUSE_STATES

cryton_core.lib.util.states.RUN_KILL_STATES

cryton_core.lib.util.states.RUN_PLAN_PAUSE_STATES

cryton_core.lib.util.states.PLAN_STATES

cryton_core.lib.util.states.PLAN_TRANSITIONS

cryton_core.lib.util.states.PLAN_SCHEDULE_STATES

cryton_core.lib.util.states.PLAN_EXECUTE_STATES

cryton_core.lib.util.states.PLAN_UNSCHEDULE_STATES

cryton_core.lib.util.states.PLAN_RESCHEDULE_STATES

cryton_core.lib.util.states.PLAN_POSTPONE_STATES

cryton_core.lib.util.states.PLAN_PAUSE_STATES

cryton_core.lib.util.states.PLAN_UNPAUSE_STATES

cryton_core.lib.util.states.PLAN_KILL_STATES

cryton_core.lib.util.states.PLAN_FINAL_STATES

cryton_core.lib.util.states.PLAN_STAGE_PAUSE_STATES

cryton_core.lib.util.states.PLAN_RUN_EXECUTE_STATES

cryton_core.lib.util.states.STAGE_STATES

cryton_core.lib.util.states.STAGE_TRANSITIONS = [(), ()]

cryton_core.lib.util.states.STAGE_EXECUTE_STATES

cryton_core.lib.util.states.STAGE_SCHEDULE_STATES

cryton_core.lib.util.states.STAGE_UNSCHEDULE_STATES

cryton_core.lib.util.states.STAGE_PAUSE_STATES

cryton_core.lib.util.states.STAGE_UNPAUSE_STATES

cryton_core.lib.util.states.STAGE_KILL_STATES

cryton_core.lib.util.states.STAGE_FINAL_STATES

cryton_core.lib.util.states.STAGE_PLAN_EXECUTE_STATES

cryton_core.lib.util.states.STEP_STATES

cryton_core.lib.util.states.STEP_TRANSITIONS = [(), (), (), (), (), (), (), (), (), (), ()]

cryton_core.lib.util.states.STEP_EXECUTE_STATES

cryton_core.lib.util.states.STEP_KILL_STATES

cryton_core.lib.util.states.STEP_FINAL_STATES

cryton_core.lib.util.states.STEP_STAGE_EXECUTE_STATES

class cryton_core.lib.util.states.StateMachine(*execution_id: int*, *valid_states: list*, *valid_transitions: list*)

Base state machine

_check_transition_validity(*state_from: str*, *state_to: str*) → bool

Check if the transition is valid :param state_from: from what state will be the transition made :param state_to: to what state will be the transition made :raises:

InvalidStateError StateTransitionError

Returns

True if transition is valid, False if transition is duplicate

_check_valid_state(*state*, *valid_states*)

Check if the state is in valid states, if valid_states are empty don't check :param state: state to check :param valid_states: list of valid states :raises:

InvalidStateError

Returns
None

class cryton_core.lib.util.states.RunStateMachine(*execution_id: int*)

Bases: StateMachine

Run state machine

validate_transition(*state_from: str*, *state_to: str*) → bool

Check if the transition is valid :param state_from: from what state will be the transition made :param state_to: to what state will be the transition made :raises:

RunStateTransitionError RunInvalidStateError

Returns
True if transition is valid, False if transition is duplicate

validate_state(*state: str*, *valid_states: list*) → None

Check if the state is in valid states, if valid_states are empty don't check :param state: state to check :param valid_states: list of valid states :raises:

RunInvalidStateError

Returns
None

class cryton_core.lib.util.states.PlanStateMachine(*execution_id: int*)

Bases: StateMachine

Plan state machine

validate_transition(*state_from: str*, *state_to: str*) → bool

Check if the transition is valid :param state_from: from what state will be the transition made :param state_to: to what state will be the transition made :raises:

PlanStateTransitionError PlanInvalidStateError

Returns
True if transition is valid, False if transition is duplicate

validate_state(*state: str*, *valid_states: list*) → None

Check if the state is in valid states, if valid_states are empty don't check :param state: state to check :param valid_states: list of valid states :raises:

PlanInvalidStateError

Returns
None

class cryton_core.lib.util.states.StageStateMachine(*execution_id: int*)

Bases: StateMachine

Stage state machine

validate_transition(*state_from: str*, *state_to: str*) → bool

Check if the transition is valid :param state_from: from what state will be the transition made :param state_to: to what state will be the transition made :raises:

StageStateTransitionError StageInvalidStateError

Returns
True if transition is valid, False if transition is duplicate

validate_state(*state: str*, *valid_states: list*) → None

Check if the state is in valid states, if valid_states are empty don't check :param state: state to check :param valid_states: list of valid states :raises:

StageInvalidStateError

Returns
None

class cryton_core.lib.util.states.StepStateMachine(*execution_id: int*)

Bases: StateMachine

Step state machine

validate_transition(*state_from: str*, *state_to: str*) → bool

Check if the transition is valid :param state_from: from what state will be the transition made :param state_to: to what state will be the transition made :raises:

StepStateTransitionError StepInvalidStateError

Returns
True if transition is valid, False if transition is duplicate

validate_state(*state: str*, *valid_states: list*) → None

Check if the state is in valid states, if valid_states are empty don't check :param state: state to check :param valid_states: list of valid states :raises:

StepInvalidStateError

Returns
None

cryton_core.lib.util.util

Module Contents

Functions

convert_to_utc(→ datetime.datetime)	Convert datetime in specified timezone to UTC timezone
convert_from_utc(→ datetime.datetime)	Convert datetime in UTC timezone to specified timezone
parse_delta_to_datetime(→ datetime.timedelta)	
split_into_lists(→ List[List])	Evenly splits list into n lists.
run_executions_in_threads(→ None)	Creates new Rabbit connection, distributes step executions into threads and runs the threads.
run_step_executions(→ None)	Creates new Rabbit channel and runs step executions.
getitem(obj, key)	Get item from object using key.
parse_dot_argument(→ List[str])	Takes a single argument (Dict key) from dot notation and checks if it also contains list indexes.
get_from_dict(dict_in, value)	Get value from dict_in dict
_finditem(obj, key)	Check if giben key exists in an object
update_inner(obj, dict_in, startswith)	Update value inside the object with one specified by prefix and path from dict_in
replace_value_in_dict(dict_to_repl, dict_in[, startswith])	Replace value in dictionary
fill_dynamic_variables(in_dict, var_dict)	Fill variables in in_dict with var_dict.
get_all_values(input_container)	Get all values (recursively) from a dict
get_dynamic_variables(in_dict[, prefix])	Get list of dynamic variables for input dict
get_prefixes(vars_list)	Get list of prefixes from list of dynamic variables
pop_key(in_dict, val)	Pop key at specified position (eg. 'k1.k2.k3')
add_key(in_dict, path, val)	Add value on specified key position
rename_key(in_dict, rename_from, rename_to)	Rename key (= move to different place)
get_logs(→ list)	Retrieve logs from log file.

cryton_core.lib.util.util.convert_to_utc(*original_datetime: datetime.datetime*, *time_zone: str = 'utc'*, *offset_aware: bool = False*) → datetime.datetime

Convert datetime in specified timezone to UTC timezone :param original_datetime: datetime to convert :param time_zone: timezone of the original datetime (examples: "utc"; "Europe/Prague") :param offset_aware: if True, utc_datetime will be offset-aware, else it will be offset-naive :return: datetime in UTC timezone

cryton_core.lib.util.util.convert_from_utc(*utc_datetime: datetime.datetime*, *time_zone: str*, *offset_aware: bool = False*) → datetime.datetime

Convert datetime in UTC timezone to specified timezone :param utc_datetime: datetime in UTC timezone to convert :param time_zone: timezone of the new datetime (examples: "utc"; "Europe/Prague") :param offset_aware: if True, utc_datetime will be offset-aware, else it will be offset-naive :return: datetime with the specified timezone

cryton_core.lib.util.util.parse_delta_to_datetime(*time_str: str*) → datetime.timedelta

cryton_core.lib.util.util.split_into_lists(*input_list: List*, *target_number_of_lists: int*) → List[List]

Evenly splits list into n lists. E.g. split_into_lists([1,2,3,4], 4) returns [[1], [2], [3], [4]]. :param input_list: object to split :param target_number_of_lists: how many lists to split into :returns: list of lists containing original items

cryton_core.lib.util.util.run_executions_in_threads(*step_executions: List*) → None

Creates new Rabbit connection, distributes step executions into threads and runs the threads. To set desired number of threads/process, see "CRYTON_CORE_EXECUTION_THREADS_PER_PROCESS" variable in config.

Parameters

step_executions – list of step execution objects

cryton_core.lib.util.util.run_step_executions(*rabbit_connection: amqpstorm.Connection*, *step_execution_list: List*) → None

Creates new Rabbit channel and runs step executions.

Parameters

- rabbit_connection – Rabbit connection
- step_execution_list – list of step execution objects to execute

cryton_core.lib.util.util.getitem(*obj: List | Dict*, *key: str*)

Get item from object using key. :param obj: Iterable accessible using key :param key: Key to use to get (match) Item :return: Matched item

cryton_core.lib.util.util.parse_dot_argument(*dot_argument: str*) → List[str]

Takes a single argument (Dict key) from dot notation and checks if it also contains list indexes. :param dot_argument: Dict key from dot notation possibly containing list indexes :return: Dict key and possible List indexes

cryton_core.lib.util.util.get_from_dict(*dict_in: dict*, *value: str*)

Get value from dict_in dict eg:

dict_in: {'output': {'username': 'admin'}} value: '$dict_in.output.username' return: 'admin'

Parameters

- value – value defined in template
- dict_in – dict_in for this step

Returns

value from dict_in

cryton_core.lib.util.util._finditem(*obj*, *key*)

Check if giben key exists in an object :param obj: dictionary/list :param key: key :return: value at the key position

cryton_core.lib.util.util.update_inner(*obj: dict*, *dict_in: dict*, *startswith: str*)

Update value inside the object with one specified by prefix and path from dict_in eg.: $dict_in.test replaces with dict_in.get('test')

Parameters

- obj – Object
- dict_in – dict_in dictioanry
- startswith – prefix, eg. $

Returns

cryton_core.lib.util.util.replace_value_in_dict(*dict_to_repl: dict*, *dict_in: dict*, *startswith: str = '$'*)

Replace value in dictionary :param dict_to_repl: :param dict_in: dict_in :param startswith: prefix :return:

cryton_core.lib.util.util.fill_dynamic_variables(*in_dict*, *var_dict*)

Fill variables in in_dict with var_dict.

Parameters

- in_dict –
- var_dict –

Returns

cryton_core.lib.util.util.get_all_values(*input_container*)

Get all values (recursively) from a dict :param input_container: input dict or list :return: yields elements, use as list(get_all_values(d))

cryton_core.lib.util.util.get_dynamic_variables(*in_dict*, *prefix='$'*)

Get list of dynamic variables for input dict :param in_dict: :param prefix: :return:

cryton_core.lib.util.util.get_prefixes(*vars_list*)

Get list of prefixes from list of dynamic variables :param vars_list: :return:

cryton_core.lib.util.util.pop_key(*in_dict*, *val*)

Pop key at specified position (eg. 'k1.k2.k3') :param in_dict: :param val: :return: Nothing, changes in_dict inplace

cryton_core.lib.util.util.add_key(*in_dict*, *path*, *val*)

Add value on specified key position :param in_dict: eg. {a: 1, b:2} :param path: 'c.d.e' :param val: '3 :return: changes in place, eg. {a:1, b:2, c:{d:{e:3}}}

cryton_core.lib.util.util.rename_key(*in_dict*, *rename_from*, *rename_to*)

Rename key (= move to different place)

eg. in_dict = {a: 1, b: 2, c: {d: 3}} rename_from = 'c.d' rename_to = 'e.f.g'

result = {a: 1, b: 2, e: {f: {g: 3}}}

Parameters

- in_dict –
- rename_from –
- rename_to –

Returns

Changes inplace

:raises KeyError, if rename_from key is not found

cryton_core.lib.util.util.get_logs() → list

Retrieve logs from log file. :return: Parsed Logs

1.1.2 Submodules

cryton_core.asgi

ASGI config for cryton_core project.

It exposes the ASGI callable as a module-level variable named application.

For more information on this file, see https://docs.djangoproject.com/en/4.0/howto/deployment/asgi/

Module Contents

cryton_core.asgi.application

cryton_core.manage

Django's command-line utility for administrative tasks.

Module Contents

Functions

main()	Run administrative tasks.

cryton_core.manage.main()

Run administrative tasks.

cryton_core.settings

Django settings for cryton_core project.

Generated by 'django-admin startproject' using Django 4.0.1.

For more information on this file, see https://docs.djangoproject.com/en/4.0/topics/settings/

For the full list of settings and their values, see https://docs.djangoproject.com/en/4.0/ref/settings/

Module Contents

cryton_core.settings.BASE_DIR

cryton_core.settings.SECRET_KEY

cryton_core.settings.DEBUG

cryton_core.settings.ALLOWED_HOSTS

cryton_core.settings.CORS_ALLOW_ALL_ORIGINS = True

cryton_core.settings.INSTALLED_APPS = ['cryton_core.cryton_app', 'django.contrib.admin', 'django.contrib.auth',...

cryton_core.settings.MIDDLEWARE = ['corsheaders.middleware.CorsMiddleware', 'django.middleware.security.SecurityMiddleware',...

cryton_core.settings.ROOT_URLCONF = 'cryton_core.urls'

cryton_core.settings.TEMPLATES

cryton_core.settings.WSGI_APPLICATION = 'cryton_core.wsgi.application'

cryton_core.settings.DATABASES

cryton_core.settings.AUTH_PASSWORD_VALIDATORS

cryton_core.settings.LANGUAGE_CODE = 'en-us'

cryton_core.settings.TIME_ZONE

cryton_core.settings.USE_I18N = True

cryton_core.settings.USE_TZ = True

cryton_core.settings.STATIC_URL = 'static/'

cryton_core.settings.STATIC_ROOT

cryton_core.settings.DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

cryton_core.settings.REST_FRAMEWORK

cryton_core.settings.SPECTACULAR_SETTINGS

cryton_core.settings.DATA_UPLOAD_MAX_MEMORY_SIZE = 100000000

cryton_core.settings.FILE_UPLOAD_MAX_MEMORY_SIZE = 10000000

cryton_core.urls

cryton_core URL Configuration

The *urlpatterns* list routes URLs to views. For more information please see:
https://docs.djangoproject.com/en/4.0/topics/http/urls/

Examples: Function views

1. Add an import: from my_app import views
2. Add a URL to urlpatterns: path('', views.home, name='home')

Class-based views

1. Add an import: from other_app.views import Home
2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')

Including another URLconf

1. Import the include() function: from django.urls import include, path
2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))

Module Contents

cryton_core.urls.urlpatterns

cryton_core.wsgi

WSGI config for cryton_core project.

It exposes the WSGI callable as a module-level variable named application.

For more information on this file, see https://docs.djangoproject.com/en/4.0/howto/deployment/wsgi/

Module Contents

cryton_core.wsgi.application

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

CONTENTS:

CHAPTER

ONE

API REFERENCE

This page contains auto-generated API reference documentation[1].

1.1 cryton_cli

1.1.1 Subpackages

cryton_cli.etc

Submodules

cryton_cli.etc.config

Module Contents

cryton_cli.etc.config.APP_DIRECTORY

cryton_cli.etc.config.TIME_ZONE

cryton_cli.etc.config.TIME_ZONE

cryton_cli.etc.config.API_HOST

cryton_cli.etc.config.API_PORT

cryton_cli.etc.config.API_SSL

cryton_cli.etc.config.API_ROOT

[1] Created with sphinx-autoapi

cryton_cli.lib

Subpackages

cryton_cli.lib.commands

Submodules

cryton_cli.lib.commands.dynamic_runs

cryton_cli.lib.commands.execution_variable

Module Contents

Functions

execution_variable(→ None)	Manage Execution variables from here.
execution_variable_list(→ None)	List existing Execution variables in Cryton.
execution_variable_create(→ None)	Create new execution variable(s) for PLAN_EXECUTION_ID from FILE.
execution_variable_read(→ None)	Show Execution variable with EXECUTION_VARIABLE_ID saved in Cryton.
execution_variable_delete(→ None)	Delete Execution variable with EXECUTION_VARIABLE_ID saved in Cryton.

cryton_cli.lib.commands.execution_variable.execution_variable(_) → None

Manage Execution variables from here.

Parameters

_ – Click ctx object

Returns

None

cryton_cli.lib.commands.execution_variable.execution_variable_list(*ctx:* cryton_cli.lib.util.util.CliContext, *less: bool*, *offset: int*, *limit: int*, *localize: bool*, *parent: int*, *parameter_filters: List[str] | None*) → None

List existing Execution variables in Cryton.

Parameters

- ctx – Click ctx object
- less – Show less like output
- offset – Initial index from which to return the results
- limit – Number of results per page
- localize – If datetime variables should be converted to local timezone
- parent – Plan execution ID used to filter returned Execution variables
- parameter_filters – Filter results using returned parameters (for example *id*, *name*, etc.)

Returns
None

cryton_cli.lib.commands.execution_variable.execution_variable_create(*ctx:* cryton_cli.lib.util.util.CliContext, *plan_execution_id: int*, *file: str*) → None

Create new execution variable(s) for PLAN_EXECUTION_ID from FILE.

PLAN_EXECUTION_ID IS ID of the desired PlanExecution.

FILE is path (can be multiple) to file(s) containing execution variables.

Parameters

- ctx – Click ctx object
- plan_execution_id – ID of the desired PlanExecution.
- file – Path(s) to file(s) containing execution variables.

Returns
None

cryton_cli.lib.commands.execution_variable.execution_variable_read(*ctx:* cryton_cli.lib.util.util.CliContext, *execution_variable_id: int*, *less: bool*, *localize: bool*) → None

Show Execution variable with EXECUTION_VARIABLE_ID saved in Cryton.

EXECUTION_VARIABLE_ID is ID of the Execution variable you want to see.

Parameters

- ctx – Click ctx object
- execution_variable_id – ID of the desired execution variable
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.execution_variable.execution_variable_delete(*ctx:* cryton_cli.lib.util.util.CliContext, *execution_variable_id: int*) → None

Delete Execution variable with EXECUTION_VARIABLE_ID saved in Cryton.

EXECUTION_VARIABLE_ID is ID of the Execution_variable you want to delete.

Parameters

- ctx – Click ctx object
- execution_variable_id – ID of the desired execution variable

Returns
None

cryton_cli.lib.commands.log

Module Contents

Functions

log(→ None)	Manage Workers from here.
log_list(→ None)	List existing Logs in Cryton.

cryton_cli.lib.commands.log.log(_) → None

Manage Workers from here.

Parameters
_ – Click ctx object

Returns
None

cryton_cli.lib.commands.log.log_list(*ctx: cryton_cli.lib.util.util.CliContext*, *less: bool*, *offset: int*, *limit: int*, *localize: bool*, *parameter_filters*) → None

List existing Logs in Cryton.

Parameters

- ctx – Click ctx object
- less – Show less like output
- offset – Initial index from which to return the results
- limit – Number of results per page
- localize – If datetime variables should be converted to local timezone
- parameter_filters – Phrase to use to filter the logs

Returns
None

cryton_cli.lib.commands.plan

Module Contents

Functions

plan(→ None)	Manage Plans from here.
plan_list(→ None)	List existing Plans in Cryton.
plan_create(→ None)	Fill template PLAN_TEMPLATE_ID with inventory file(s) and save it to Cryton.
plan_read(→ None)	Show Plan with PLAN_ID saved in Cryton.
plan_delete(→ None)	Delete Plan with PLAN_ID saved in Cryton.
plan_validate(→ None)	Validate (syntax check) your FILE with Plan.
plan_execute(→ None)	Execute Plan saved in Cryton with PLAN_ID on Worker with WORKER_ID and attach it to Run with RUN_ID.
plan_get_plan(→ None)	Get Plan with PLAN_ID saved in Cryton.
plan_execution(→ None)	Manage Plan's executions from here.
plan_execution_list(→ None)	List existing Plan's executions in Cryton.
plan_execution_delete(→ None)	Delete Plan's execution with EXECUTION_ID saved in Cryton.
plan_execution_read(→ None)	Show Plan's execution with EXECUTION_ID saved in Cryton.
plan_execution_pause(→ None)	Pause Plan's execution with EXECUTION_ID saved in Cryton.
plan_execution_report(→ None)	Create report for Plan's execution with EXECUTION_ID saved in Cryton.
plan_execution_unpause(→ None)	Resume Plan's execution with EXECUTION_ID saved in Cryton.
plan_execution_validate_modules(→ None)	Validate modules for Plan's execution with EXECUTION_ID saved in Cryton.
plan_execution_kill(→ None)	Kill Plan's execution with EXECUTION_ID saved in Cryton.

cryton_cli.lib.commands.plan.plan(_) → None

Manage Plans from here.

Parameters
_ – Click ctx object

Returns
None

cryton_cli.lib.commands.plan.plan_list(*ctx: cryton_cli.lib.util.util.CliContext*, *less: bool*, *offset: int*, *limit: int*, *localize: bool*, *parameter_filters: List[str] | None*) → None

List existing Plans in Cryton.

Parameters

- ctx – Click ctx object
- less – Show less like output
- offset – Initial index from which to return the results
- limit – Number of results per page
- localize – If datetime variables should be converted to local timezone
- parameter_filters – Filter results using returned parameters (for example *id*, *name*, etc.)

Returns
None

cryton_cli.lib.commands.plan.plan_create(*ctx:* cryton_cli.lib.util.util.CliContext, *template_id: int*, *inventory_files: list*) → None

Fill template PLAN_TEMPLATE_ID with inventory file(s) and save it to Cryton.

PLAN_TEMPLATE_ID is ID of the template you want to fill.

Parameters

- ctx – Click ctx object
- template_id – ID of the Plan's template to use
- inventory_files – Inventory file(s) used to fill the template

Returns
None

cryton_cli.lib.commands.plan.plan_read(*ctx:* cryton_cli.lib.util.util.CliContext, *plan_id: int*, *less: bool*, *localize: bool*) → None

Show Plan with PLAN_ID saved in Cryton.

PLAN_ID is ID of the Plan you want to see.

Parameters

- ctx – Click ctx object
- plan_id – ID of the desired Plan
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.plan.plan_delete(*ctx:* cryton_cli.lib.util.util.CliContext, *plan_id: int*) → None

Delete Plan with PLAN_ID saved in Cryton.

PLAN_ID is ID of the Plan you want to delete.

Parameters

- ctx – Click ctx object
- plan_id – ID of the desired Plan

Returns
None

cryton_cli.lib.commands.plan.plan_validate(*ctx:* cryton_cli.lib.util.util.CliContext, *file: str*, *inventory_files: list*) → None

Validate (syntax check) your FILE with Plan.

FILE is path/to/your/file that you want to validate.

Parameters

- ctx – Click ctx object
- file – File containing your Plan in yaml
- inventory_files – Inventory file(s) used to fill the template

Returns
None

cryton_cli.lib.commands.plan.plan_execute(*ctx: cryton_cli.lib.util.util.CliContext*, *plan_id: int*, *worker_id: int*, *run_id: int*) → None

Execute Plan saved in Cryton with PLAN_ID on Worker with WORKER_ID and attach it to Run with RUN_ID.

PLAN_ID is ID of the Plan you want to execute.

WORKER_ID is ID of the Plan you want to execute.

RUN_ID is ID of the Run you want to attach this execution to.

Parameters

- ctx – Click ctx object
- plan_id – ID of the desired Plan
- worker_id – ID of the desired Worker
- run_id – ID of the desired Run

Returns
None

cryton_cli.lib.commands.plan.plan_get_plan(*ctx: cryton_cli.lib.util.util.CliContext*, *plan_id: int*, *file: str*, *less: bool*, *localize: bool*) → None

Get Plan with PLAN_ID saved in Cryton.

PLAN_ID is ID of the Plan you want to get.

Parameters

- ctx – Click ctx object
- plan_id – ID of the desired Plan
- file – File to save the plan to (default is /tmp)
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.plan.plan_execution(_) → None

Manage Plan's executions from here.

Parameters
_ – Click ctx object

Returns
None

cryton_cli.lib.commands.plan.plan_execution_list(*ctx: cryton_cli.lib.util.util.CliContext*, *less: bool*, *offset: int*, *limit: int*, *localize: bool*, *parent: int*, *parameter_filters: List[str] | None*) → None

List existing Plan's executions in Cryton.

Parameters

- ctx – Click ctx object
- less – Show less like output

- offset – Initial index from which to return the results
- limit – Number of results per page
- localize – If datetime variables should be converted to local timezone
- parent – Run ID used to filter returned Plan executions
- parameter_filters – Filter results using returned parameters (for example *id*, *name*, etc.)

Returns
None

cryton_cli.lib.commands.plan.plan_execution_delete(*ctx: cryton_cli.lib.util.util.CliContext*, *execution_id: int*) → None

Delete Plan's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Plan's execution you want to delete.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Plan's execution

Returns
None

cryton_cli.lib.commands.plan.plan_execution_read(*ctx: cryton_cli.lib.util.util.CliContext*, *execution_id: int*, *less: bool*, *localize: bool*) → None

Show Plan's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Plan's execution you want to see.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Plan's execution
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.plan.plan_execution_pause(*ctx: cryton_cli.lib.util.util.CliContext*, *execution_id: int*) → None

Pause Plan's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Plan's execution you want to pause.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Plan's execution

Returns
None

cryton_cli.lib.commands.plan.plan_execution_report(*ctx: cryton_cli.lib.util.util.CliContext*, *execution_id: int*, *file: str*, *less: bool*, *localize: bool*) → None

Create report for Plan's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Plan's execution you want to create report for.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Plan's execution
- file – File to save the report to (default is /tmp)
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.plan.plan_execution_unpause(*ctx: cryton_cli.lib.util.util.CliContext*, *execution_id: int*) → None

Resume Plan's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Plan's execution you want to resume.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Plan's execution

Returns
None

cryton_cli.lib.commands.plan.plan_execution_validate_modules(*ctx: cryton_cli.lib.util.util.CliContext*, *execution_id: int*) → None

Validate modules for Plan's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Plan's execution you want to validate modules for.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Plan's execution

Returns
None

cryton_cli.lib.commands.plan.plan_execution_kill(*ctx: cryton_cli.lib.util.util.CliContext*, *execution_id: int*) → None

Kill Plan's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Plan's execution you want to kill.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Plan's execution

Returns
None

cryton_cli.lib.commands.plan_template

Module Contents

Functions

template(→ None)	Manage Plan templates from here.
template_list(→ None)	List existing Plan templates in Cryton.
template_create(→ None)	Store Plan Template into Cryton.
template_read(→ None)	Show Template with TEMPLATE_ID saved in Cryton.
template_delete(→ None)	Delete Template with TEMPLATE_ID saved in Cryton.
template_get_template(→ None)	Get Template with TEMPLATE_ID saved in Cryton.

cryton_cli.lib.commands.plan_template.template(_) → None

Manage Plan templates from here.

Parameters
_ – Click ctx object

Returns
None

cryton_cli.lib.commands.plan_template.template_list(*ctx: cryton_cli.lib.util.util.CliContext*, *less: bool*, *offset: int*, *limit: int*, *localize: bool*, *parameter_filters: List[str] | None*) → None

List existing Plan templates in Cryton.

Parameters

- ctx – Click ctx object
- less – Show less like output
- offset – Initial index from which to return the results
- limit – Number of results per page
- localize – If datetime variables should be converted to local timezone
- parameter_filters – Filter results using returned parameters (for example *id*, *name*, etc.)

Returns
None

cryton_cli.lib.commands.plan_template.template_create(*ctx: cryton_cli.lib.util.util.CliContext*, *file: str*) → None

Store Plan Template into Cryton.

FILE is path/to/your/file that you want to upload to Cryton.

Parameters

- ctx – Click ctx object
- file – File containing your Plan Template in yaml

Returns
None

cryton_cli.lib.commands.plan_template.template_read(*ctx: cryton_cli.lib.util.util.CliContext*, *template_id: int*, *less: bool*, *localize: bool*) → None

Show Template with TEMPLATE_ID saved in Cryton.

TEMPLATE_ID is ID of the Template you want to see.

Parameters

- ctx – Click ctx object
- template_id – ID of the desired Template
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.plan_template.template_delete(*ctx: cryton_cli.lib.util.util.CliContext*, *template_id: int*) → None

Delete Template with TEMPLATE_ID saved in Cryton.

TEMPLATE_ID is ID of the Template you want to delete.

Parameters

- ctx – Click ctx object
- template_id – ID of the desired Template

Returns
None

cryton_cli.lib.commands.plan_template.template_get_template(*ctx: cryton_cli.lib.util.util.CliContext*, *template_id: int*, *file: str*, *less: bool*, *localize: bool*) → None

Get Template with TEMPLATE_ID saved in Cryton.

TEMPLATE_ID is ID of the Template you want to get.

Parameters

- ctx – Click ctx object
- template_id – ID of the desired Template
- file – File to save the template to (default is /tmp)
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.run

Module Contents

Functions

run(→ None)	Manage Runs from here.
run_list(→ None)	List existing Runs in Cryton.
run_create(→ None)	Create new Run with PLAN_ID and WORKER_IDS.
run_read(→ None)	Show Run with RUN_ID saved in Cryton.
run_delete(→ None)	Delete Run with RUN_ID saved in Cryton.
run_execute(→ None)	Execute Run saved in Cryton with RUN_ID.
run_pause(→ None)	Pause Run saved in Cryton with RUN_ID.
run_postpone(→ None)	Postpone Run saved in Cryton with RUN_ID by HOURS, MINUTES and SECONDS.
run_report(→ None)	Create report for Run with RUN_ID saved in Cryton.
run_reschedule(→ None)	Reschedule Run saved in Cryton with RUN_ID to specified DATE and TIME.
run_schedule(→ None)	Schedule Run saved in Cryton with RUN_ID to specified DATE and TIME.
run_unpause(→ None)	Resume Run saved in Cryton with RUN_ID.
run_unschedule(→ None)	Unschedule Run saved in Cryton with RUN_ID.
run_kill(→ None)	Kill Run saved in Cryton with RUN_ID.
run_validate_modules(→ None)	Validate modules for Run with RUN_ID saved in Cryton.
run_get_plan(→ None)	Get plan from Run with RUN_ID saved in Cryton.

cryton_cli.lib.commands.run.run(_) → None

Manage Runs from here.

Parameters
_ – Click ctx object

Returns
None

cryton_cli.lib.commands.run.run_list(*ctx: cryton_cli.lib.util.util.CliContext*, *less: bool*, *offset: int*, *limit: int*, *localize: bool*, *parameter_filters: List[str] | None*) → None

List existing Runs in Cryton.

Parameters

- ctx – Click ctx object
- less – Show less like output
- offset – Initial index from which to return the results
- limit – Number of results per page
- localize – If datetime variables should be converted to local timezone
- parameter_filters – Filter results using returned parameters (for example *id*, *name*, etc.)

Returns
None

cryton_cli.lib.commands.run.run_create(*ctx: cryton_cli.lib.util.util.CliContext*, *plan_id: int*, *worker_ids: list*) → None

Create new Run with PLAN_ID and WORKER_IDS.

PLAN_ID is ID of the Plan you want to create Run for. (for example 1)

WORKER_IDS is list of IDs you want to use for Run. (1 2 3)

Parameters

- ctx – Click ctx object
- plan_id – ID of the Plan that will be used in Run
- worker_ids – List of IDs you want to use for Run

Returns
None

cryton_cli.lib.commands.run.run_read(*ctx: cryton_cli.lib.util.util.CliContext*, *run_id: int*, *less: bool*, *localize: bool*) → None

Show Run with RUN_ID saved in Cryton.

RUN_ID is ID of the Run you want to see.

Parameters

- ctx – Click ctx object
- run_id – ID of the desired Run
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.run.run_delete(*ctx: cryton_cli.lib.util.util.CliContext*, *run_id: int*) → None

Delete Run with RUN_ID saved in Cryton.

RUN_ID is ID of the Run you want to delete.

Parameters

- ctx – Click ctx object
- run_id – ID of the desired Run

Returns
None

cryton_cli.lib.commands.run.run_execute(*ctx: cryton_cli.lib.util.util.CliContext*, *run_id: int*) → None

Execute Run saved in Cryton with RUN_ID.

RUN_ID is ID of the Run you want to execute.

Parameters

- ctx – Click ctx object
- run_id – ID of the desired Run

Returns
None

cryton_cli.lib.commands.run.run_pause(*ctx: cryton_cli.lib.util.util.CliContext*, *run_id: int*) → None

Pause Run saved in Cryton with RUN_ID.

RUN_ID is ID of the Run you want to pause.

Parameters

- ctx – Click ctx object
- run_id – ID of the desired Run

Returns
None

cryton_cli.lib.commands.run.run_postpone(*ctx: cryton_cli.lib.util.util.CliContext*, *run_id: int*, *hours: int*, *minutes: int*, *seconds: int*) → None

Postpone Run saved in Cryton with RUN_ID by HOURS, MINUTES and SECONDS.

RUN_ID is ID of the Run you want to postpone.

HOURS is number of hours.

MINUTES is number of minutes.

SECONDS is number of seconds.

Parameters

- ctx – Click ctx object
- run_id – ID of the desired Run
- hours – how many hours should be added to Run's start time
- minutes – how many hours should be added to Run's start time
- seconds – how many hours should be added to Run's start time

Returns
None

cryton_cli.lib.commands.run.run_report(*ctx: cryton_cli.lib.util.util.CliContext*, *run_id: int*, *file: str*, *less: bool*, *localize: bool*) → None

Create report for Run with RUN_ID saved in Cryton.

RUN_ID is ID of the Run you want to create report for.

Parameters

- ctx – Click ctx object
- run_id – ID of the desired Run
- file – File to save the report to (default is /tmp)
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.run.run_reschedule(*ctx: cryton_cli.lib.util.util.CliContext*, *run_id: int*, *to_date: str*, *to_time: str*, *utc_timezone: bool*) → None

Reschedule Run saved in Cryton with RUN_ID to specified DATE and TIME.

RUN_ID is ID of the Run you want to reschedule.

DATE in format year-month-day (Y-m-d).

TIME in format hours:minutes:seconds (H:M:S).

Parameters

- ctx – Click ctx object
- run_id – ID of the desired Run
- to_date – to what date you want to reschedule Run
- to_time – to what time you want to reschedule Run
- utc_timezone – Use UTC timezone instead of local timezone

Returns
None

cryton_cli.lib.commands.run.run_schedule(*ctx: cryton_cli.lib.util.util.CliContext*, *run_id: int*, *to_date: str*, *to_time: str*, *utc_timezone: bool*) → None

Schedule Run saved in Cryton with RUN_ID to specified DATE and TIME.

RUN_ID is ID of the Run you want to schedule.

DATE in format year-month-day (Y-m-d).

TIME in format hours:minutes:seconds (H:M:S).

Parameters

- ctx – Click ctx object
- run_id – ID of the desired Run
- to_date – to what date you want to reschedule Run
- to_time – to what time you want to reschedule Run
- utc_timezone – Use UTC timezone instead of local timezone

Returns
None

cryton_cli.lib.commands.run.run_unpause(*ctx: cryton_cli.lib.util.util.CliContext*, *run_id: int*) → None

Resume Run saved in Cryton with RUN_ID.

RUN_ID is ID of the Run you want to resume.

Parameters

- ctx – Click ctx object
- run_id – ID of the desired Run

Returns
None

cryton_cli.lib.commands.run.run_unschedule(*ctx: cryton_cli.lib.util.util.CliContext*, *run_id: int*) → None

Unschedule Run saved in Cryton with RUN_ID.

RUN_ID is ID of the Run you want to unschedule.

Parameters

- ctx – Click ctx object
- run_id – ID of the desired Run

Returns
None

cryton_cli.lib.commands.run.run_kill(*ctx: cryton_cli.lib.util.util.CliContext*, *run_id: int*) → None

Kill Run saved in Cryton with RUN_ID.

RUN_ID is ID of the Run you want to kill.

Parameters

- ctx – Click ctx object
- run_id – ID of the desired Run

Returns
None

cryton_cli.lib.commands.run.run_validate_modules(*ctx: cryton_cli.lib.util.util.CliContext*, *run_id: int*) → None

Validate modules for Run with RUN_ID saved in Cryton.

RUN_ID is ID of the Run you want to validate modules for.

Parameters

- ctx – Click ctx object
- run_id – ID of the desired Run

Returns
None

cryton_cli.lib.commands.run.run_get_plan(*ctx: cryton_cli.lib.util.util.CliContext*, *run_id: int*, *file: str*, *less: bool*, *localize: bool*) → None

Get plan from Run with RUN_ID saved in Cryton.

RUN_ID is ID of the Run you want to get plan from.

Parameters

- ctx – Click ctx object
- run_id – ID of the desired Run
- file – File to save the plan to (default is /tmp)
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.stage

Module Contents

Functions

stage(→ None)	Manage Stages from here.
stage_list(→ None)	List existing Stages in Cryton.
stage_create(→ None)	Create Stage from FILE and add it to Plan with PLAN_ID.
stage_read(→ None)	Show Stage with STAGE_ID saved in Cryton.
stage_delete(→ None)	Delete Stage with STAGE_ID saved in Cryton.
stage_validate(→ None)	Validate (syntax check) your FILE with Stage.
stage_start_trigger(→ None)	Start Stage's trigger with STAGE_ID under Plan execution with PLAN_EXECUTION_ID.
stage_execution(→ None)	Manage Stage's executions from here.
stage_execution_list(→ None)	List existing Stage's executions in Cryton.
stage_execution_delete(→ None)	Delete Stage's execution with EXECUTION_ID saved in Cryton.
stage_execution_read(→ None)	Show Stage's execution with EXECUTION_ID saved in Cryton.
stage_execution_report(→ None)	Create report for Stage's execution with EXECUTION_ID saved in Cryton.
stage_execution_kill(→ None)	Kill Stage's execution with EXECUTION_ID saved in Cryton.
stage_execution_re_execute(→ None)	Re-execute Stage's execution with EXECUTION_ID saved in Cryton.

cryton_cli.lib.commands.stage.stage(_) → None

Manage Stages from here.

Parameters
_ – Click ctx object

Returns
None

cryton_cli.lib.commands.stage.stage_list(*ctx: cryton_cli.lib.util.util.CliContext*, *less: bool*, *offset: int*, *limit: int*, *localize: bool*, *parent: int*, *parameter_filters: List[str] | None*) → None

List existing Stages in Cryton.

Parameters

- ctx – Click ctx object
- less – Show less like output
- offset – Initial index from which to return the results
- limit – Number of results per page
- localize – If datetime variables should be converted to local timezone
- parent – Plan ID used to filter returned Stages
- parameter_filters – Filter results using returned parameters (for example *id*, *name*, etc.)

Returns
None

cryton_cli.lib.commands.stage.stage_create(*ctx: cryton_cli.lib.util.util.CliContext*, *plan_id: int*, *file: str*, *inventory_files: list*) → None

Create Stage from FILE and add it to Plan with PLAN_ID.

PLAN_ID is an ID of the Plan you want to add the Stage to. FILE is a path to the file containing the Stage template.

Parameters

- ctx – Click ctx object
- plan_id – ID of the Plan to use
- file – File used as the Stage template
- inventory_files – Inventory file(s) used to fill the template

Returns
None

cryton_cli.lib.commands.stage.stage_read(*ctx: cryton_cli.lib.util.util.CliContext*, *stage_id: int*, *less: bool*, *localize: bool*) → None

Show Stage with STAGE_ID saved in Cryton.

STAGE_ID is ID of the Stage you want to see.

Parameters

- ctx – Click ctx object
- stage_id – ID of the desired Stage
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.stage.stage_delete(*ctx: cryton_cli.lib.util.util.CliContext*, *stage_id: int*) → None

Delete Stage with STAGE_ID saved in Cryton.

STAGE_ID is ID of the Stage you want to delete.

Parameters

- ctx – Click ctx object
- stage_id – ID of the desired Stage

Returns
None

cryton_cli.lib.commands.stage.stage_validate(*ctx: cryton_cli.lib.util.util.CliContext*, *file: str*, *inventory_files: list*, *dynamic: bool*) → None

Validate (syntax check) your FILE with Stage.

FILE is path/to/your/file that you want to validate.

Parameters

- ctx – Click ctx object

- file – File containing your Stage in yaml
- inventory_files – Inventory file(s) used to fill the template
- dynamic – If Stage will be used with a dynamic Plan

Returns
None

cryton_cli.lib.commands.stage.stage_start_trigger(*ctx: cryton_cli.lib.util.util.CliContext*, *stage_id: int*, *plan_execution_id: int*) → None

Start Stage's trigger with STAGE_ID under Plan execution with PLAN_EXECUTION_ID.

STAGE_ID is an ID of the Stage you want to start. PLAN_EXECUTION_ID is an ID of the Plan execution you want to set as a parent of the Stage execution.

Parameters

- ctx – Click ctx object
- stage_id – ID of the Stage that will be used to create execution
- plan_execution_id – ID of the Plan execution that will be set as a parent of the Stage execution

Returns
None

cryton_cli.lib.commands.stage.stage_execution(_) → None

Manage Stage's executions from here.

Parameters
_ – Click ctx object

Returns
None

cryton_cli.lib.commands.stage.stage_execution_list(*ctx: cryton_cli.lib.util.util.CliContext*, *less: bool*, *offset: int*, *limit: int*, *localize: bool*, *parent: int*, *parameter_filters: List[str] | None*) → None

List existing Stage's executions in Cryton.

Parameters

- ctx – Click ctx object
- less – Show less like output
- offset – Initial index from which to return the results
- limit – Number of results per page
- localize – If datetime variables should be converted to local timezone
- parent – Plan execution ID used to filter returned Stage executions
- parameter_filters – Filter results using returned parameters (for example *id*, *name*, etc.)

Returns
None

cryton_cli.lib.commands.stage.stage_execution_delete(*ctx: cryton_cli.lib.util.util.CliContext*, *execution_id: int*) → None

Delete Stage's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Stage's execution you want to delete.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Stage's execution

Returns
None

cryton_cli.lib.commands.stage.stage_execution_read(*ctx: cryton_cli.lib.util.util.CliContext*, *execution_id: int*, *less: bool*, *localize: bool*) → None

Show Stage's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Stage's execution you want to see.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Stage's execution
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.stage.stage_execution_report(*ctx: cryton_cli.lib.util.util.CliContext*, *execution_id: int*, *file: str*, *less: bool*, *localize: bool*) → None

Create report for Stage's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Stage's execution you want to create report for.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Stage's execution
- file – File to save the report to (default is /tmp)
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.stage.stage_execution_kill(*ctx: cryton_cli.lib.util.util.CliContext*, *execution_id: int*) → None

Kill Stage's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Stage's execution you want to kill.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Stage's execution

Returns
None

cryton_cli.lib.commands.stage.stage_execution_re_execute(*ctx: cryton_cli.lib.util.util.CliContext, execution_id: int, immediately: bool*) → None

Re-execute Stage's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Stage's execution you want to kill.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Stage's execution
- immediately – True if StageExecution should be executed immediately without starting its Trigger

Returns
None

cryton_cli.lib.commands.step

Module Contents

Functions

step(→ None)	Manage Steps from here.
step_list(→ None)	List existing Steps in Cryton.
step_create(→ None)	Create Step from FILE and add it to Stage with STAGE_ID.
step_read(→ None)	Show Step with STEP_ID saved in Cryton.
step_delete(→ None)	Delete Step with STEP_ID saved in Cryton.
step_validate(→ None)	Validate (syntax check) your FILE with Step.
step_execute(→ None)	Execute Step with STEP_ID under Stage execution with STAGE_EXECUTION_ID.
step_execution(→ None)	Manage Step's executions from here.
step_execution_list(→ None)	List existing Step's executions in Cryton.
step_execution_delete(→ None)	Delete Step's execution with EXECUTION_ID saved in Cryton.
step_execution_read(→ None)	Show Step's execution with EXECUTION_ID saved in Cryton.
step_execution_report(→ None)	Create report for Step's execution with EXECUTION_ID saved in Cryton.
step_execution_kill(→ None)	Kill Step's execution with EXECUTION_ID saved in Cryton.
step_execution_re_execute(→ None)	Re-execute Step's execution with EXECUTION_ID saved in Cryton.

cryton_cli.lib.commands.step.step(_) → None

Manage Steps from here.

Parameters
_ – Click ctx object

Returns
None

cryton_cli.lib.commands.step.step_list(*ctx: cryton_cli.lib.util.util.CliContext*, *less: bool*, *offset: int*, *limit: int*, *localize: bool*, *parent: int*, *parameter_filters: List[str] | None*) → None

List existing Steps in Cryton.

Parameters

- ctx – Click ctx object
- less – Show less like output
- offset – Initial index from which to return the results
- limit – Number of results per page
- localize – If datetime variables should be converted to local timezone
- parent – Stage ID used to filter returned Steps
- parameter_filters – Filter results using returned parameters (for example *id*, *name*, etc.)

Returns
None

cryton_cli.lib.commands.step.step_create(*ctx: cryton_cli.lib.util.util.CliContext*, *stage_id: int*, *file: str*, *inventory_files: list*) → None

Create Step from FILE and add it to Stage with STAGE_ID.

STAGE_ID is an ID of the Stage you want to add the Stage to. FILE is a path to the file containing the Step template.

Parameters

- ctx – Click ctx object
- stage_id – ID of the Stage to use
- file – File used as the Step template
- inventory_files – Inventory file(s) used to fill the template

Returns
None

cryton_cli.lib.commands.step.step_read(*ctx: cryton_cli.lib.util.util.CliContext*, *step_id: int*, *less: bool*, *localize: bool*) → None

Show Step with STEP_ID saved in Cryton.

STEP_ID is ID of the Step you want to see.

Parameters

- ctx – Click ctx object
- step_id – ID of the desired Step
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.step.step_delete(*ctx: cryton_cli.lib.util.util.CliContext*, *step_id: int*) → None

Delete Step with STEP_ID saved in Cryton.

STEP_ID is ID of the Step you want to delete.

Parameters

- ctx – Click ctx object
- step_id – ID of the desired Step

Returns
None

cryton_cli.lib.commands.step.step_validate(*ctx: cryton_cli.lib.util.util.CliContext*, *file: str*, *inventory_files: list*) → None

Validate (syntax check) your FILE with Step.

FILE is path/to/your/file that you want to validate.

Parameters

- ctx – Click ctx object
- file – File containing your Step in yaml
- inventory_files – Inventory file(s) used to fill the template

Returns
None

cryton_cli.lib.commands.step.step_execute(*ctx: cryton_cli.lib.util.util.CliContext*, *step_id: int*, *stage_execution_id: int*) → None

Execute Step with STEP_ID under Stage execution with STAGE_EXECUTION_ID.

STEP_ID is ID of the Step you want to execute. STAGE_EXECUTION_ID is an ID of the Stage execution you want to set as a parent of the Step execution.

Parameters

- ctx – Click ctx object
- step_id – ID of the Step that will be used to create execution
- stage_execution_id – ID of the Stage execution that will be set as a parent of the Step execution

Returns
None

cryton_cli.lib.commands.step.step_execution(_) → None

Manage Step's executions from here.

Parameters
_ – Click ctx object

Returns
None

cryton_cli.lib.commands.step.step_execution_list(*ctx: cryton_cli.lib.util.util.CliContext*, *less: bool*, *offset: int*, *limit: int*, *localize: bool*, *parent: int*, *parameter_filters: List[str] | None*) → None

List existing Step's executions in Cryton.

Parameters

- ctx – Click ctx object
- less – Show less like output
- offset – Initial index from which to return the results
- limit – Number of results per page
- localize – If datetime variables should be converted to local timezone
- parent – Stage execution ID used to filter returned Step executions
- parameter_filters – Filter results using returned parameters (for example *id*, *name*, etc.)

Returns
None

cryton_cli.lib.commands.step.step_execution_delete(*ctx: cryton_cli.lib.util.util.CliContext*, *execution_id: int*) → None

Delete Step's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Step's execution you want to delete.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Step's execution

Returns
None

cryton_cli.lib.commands.step.step_execution_read(*ctx: cryton_cli.lib.util.util.CliContext*, *execution_id: int*, *less: bool*, *localize: bool*) → None

Show Step's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Step's execution you want to see.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Step's execution
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.step.step_execution_report(*ctx: cryton_cli.lib.util.util.CliContext*, *execution_id: int*, *file: str*, *less: bool*, *localize: bool*) → None

Create report for Step's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Step's execution you want to create report for.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Step's execution
- file – File to save the report to (default is /tmp)
- less – Show less like output

- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.step.step_execution_kill(*ctx:* cryton_cli.lib.util.util.CliContext, *execution_id: int*) → None

Kill Step's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Step's execution you want to kill.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Step's execution

Returns
None

cryton_cli.lib.commands.step.step_execution_re_execute(*ctx:* cryton_cli.lib.util.util.CliContext, *execution_id: int*) → None

Re-execute Step's execution with EXECUTION_ID saved in Cryton.

EXECUTION_ID is ID of the Step's execution you want to kill.

Parameters

- ctx – Click ctx object
- execution_id – ID of the desired Step's execution

Returns
None

cryton_cli.lib.commands.worker

Module Contents

Functions

worker(→ None)	Manage Workers from here.
worker_list(→ None)	List existing Workers in Cryton.
worker_create(→ None)	Create new Worker with NAME and save it into Cryton.
worker_read(→ None)	Show Worker with WORKER_ID saved in Cryton.
worker_delete(→ None)	Delete Worker with WORKER_ID saved in Cryton.
worker_health_check(→ None)	Check if Worker with WORKER_ID saved in Cryton is online.

cryton_cli.lib.commands.worker.worker(_) → None

Manage Workers from here.

Parameters
_ – Click ctx object

Returns
None

cryton_cli.lib.commands.worker.worker_list(*ctx: cryton_cli.lib.util.util.CliContext*, *less: bool*, *offset: int*, *limit: int*, *localize: bool*, *parameter_filters: List[str] | None*) → None

List existing Workers in Cryton.

Parameters

- ctx – Click ctx object
- less – Show less like output
- offset – Initial index from which to return the results
- limit – Number of results per page
- localize – If datetime variables should be converted to local timezone
- parameter_filters – Filter results using returned parameters (for example *id*, *name*, etc.)

Returns
None

cryton_cli.lib.commands.worker.worker_create(*ctx: cryton_cli.lib.util.util.CliContext*, *name: str*, *description: str*, *force: bool*) → None

Create new Worker with NAME and save it into Cryton.

NAME of your Worker (will be used to match your Worker). For example: "MyCustomName".

Parameters

- ctx – Click ctx object
- name – Custom name for Worker
- description – Worker's description
- force – If Worker with the same name exists, create a new one anyway

Returns
None

cryton_cli.lib.commands.worker.worker_read(*ctx: cryton_cli.lib.util.util.CliContext*, *worker_id: int*, *less: bool*, *localize: bool*) → None

Show Worker with WORKER_ID saved in Cryton.

WORKER_ID is ID of the Worker you want to see.

Parameters

- ctx – Click ctx object
- worker_id – ID of the desired Worker
- less – Show less like output
- localize – If datetime variables should be converted to local timezone

Returns
None

cryton_cli.lib.commands.worker.worker_delete(*ctx: cryton_cli.lib.util.util.CliContext*, *worker_id: int*) → None

Delete Worker with WORKER_ID saved in Cryton.

WORKER_ID is ID of the Worker you want to delete.

Parameters

- ctx – Click ctx object

- worker_id – ID of the desired Worker

Returns
None

cryton_cli.lib.commands.worker.worker_health_check(*ctx:* cryton_cli.lib.util.util.CliContext, *worker_id: int*) → None

Check if Worker with WORKER_ID saved in Cryton is online.

WORKER_ID is ID of the Worker you want to check.

Parameters

- ctx – Click ctx object
- worker_id – ID of the desired Worker

Returns
None

cryton_cli.lib.util

Submodules

cryton_cli.lib.util.api

Module Contents

Functions

create_rest_api_url(→ str)	Create REST API URL
get_request(→ Union[str, requests.Response])	Custom get request.
post_request(→ Union[str, requests.Response])	Custom post request.
delete_request(→ Union[str, requests.Response])	Custom delete request.

Attributes

RUN_LIST
RUN_CREATE
RUN_READ
RUN_DELETE
RUN_EXECUTE
RUN_PAUSE

continues on next page

Table 1 – continued from previous page

RUN_POSTPONE
RUN_REPORT
RUN_RESCHEDULE
RUN_SCHEDULE
RUN_UNPAUSE
RUN_UNSCHEDULE
RUN_KILL
RUN_VALIDATE_MODULES
RUN_GET_PLAN
PLAN_LIST
PLAN_CREATE
PLAN_VALIDATE
PLAN_READ
PLAN_DELETE
PLAN_EXECUTE
PLAN_GET_PLAN
PLAN_EXECUTION_LIST
PLAN_EXECUTION_READ
PLAN_EXECUTION_DELETE
PLAN_EXECUTION_PAUSE
PLAN_EXECUTION_REPORT
PLAN_EXECUTION_UNPAUSE
PLAN_EXECUTION_VALIDATE_MODULES
PLAN_EXECUTION_KILL
STAGE_LIST
STAGE_CREATE

continues on next page

Table 1 – continued from previous page

STAGE_VALIDATE
STAGE_READ
STAGE_DELETE
STAGE_START_TRIGGER
STAGE_EXECUTION_LIST
STAGE_EXECUTION_READ
STAGE_EXECUTION_DELETE
STAGE_EXECUTION_REPORT
STAGE_EXECUTION_KILL
STAGE_EXECUTION_RE_EXECUTE
STEP_LIST
STEP_CREATE
STEP_VALIDATE
STEP_READ
STEP_DELETE
STEP_EXECUTE
STEP_EXECUTION_LIST
STEP_EXECUTION_READ
STEP_EXECUTION_DELETE
STEP_EXECUTION_REPORT
STEP_EXECUTION_KILL
STEP_EXECUTION_RE_EXECUTE
WORKER_LIST
WORKER_CREATE
WORKER_READ
WORKER_DELETE

continues on next page

Table 1 – continued from previous page

WORKER_HEALTH_CHECK
TEMPLATE_LIST
TEMPLATE_CREATE
TEMPLATE_READ
TEMPLATE_DELETE
TEMPLATE_GET_TEMPLATE
EXECUTION_VARIABLE_LIST
EXECUTION_VARIABLE_CREATE
EXECUTION_VARIABLE_READ
EXECUTION_VARIABLE_DELETE
LOG_LIST

cryton_cli.lib.util.api.RUN_LIST

cryton_cli.lib.util.api.RUN_CREATE

cryton_cli.lib.util.api.RUN_READ

cryton_cli.lib.util.api.RUN_DELETE

cryton_cli.lib.util.api.RUN_EXECUTE

cryton_cli.lib.util.api.RUN_PAUSE

cryton_cli.lib.util.api.RUN_POSTPONE

cryton_cli.lib.util.api.RUN_REPORT

cryton_cli.lib.util.api.RUN_RESCHEDULE

cryton_cli.lib.util.api.RUN_SCHEDULE

cryton_cli.lib.util.api.RUN_UNPAUSE

cryton_cli.lib.util.api.RUN_UNSCHEDULE

cryton_cli.lib.util.api.RUN_KILL

cryton_cli.lib.util.api.RUN_VALIDATE_MODULES

cryton_cli.lib.util.api.RUN_GET_PLAN

cryton_cli.lib.util.api.PLAN_LIST

cryton_cli.lib.util.api.PLAN_CREATE

cryton_cli.lib.util.api.PLAN_VALIDATE

cryton_cli.lib.util.api.PLAN_READ

cryton_cli.lib.util.api.PLAN_DELETE

cryton_cli.lib.util.api.PLAN_EXECUTE

cryton_cli.lib.util.api.PLAN_GET_PLAN

cryton_cli.lib.util.api.PLAN_EXECUTION_LIST

cryton_cli.lib.util.api.PLAN_EXECUTION_READ

cryton_cli.lib.util.api.PLAN_EXECUTION_DELETE

cryton_cli.lib.util.api.PLAN_EXECUTION_PAUSE

cryton_cli.lib.util.api.PLAN_EXECUTION_REPORT

cryton_cli.lib.util.api.PLAN_EXECUTION_UNPAUSE

cryton_cli.lib.util.api.PLAN_EXECUTION_VALIDATE_MODULES

cryton_cli.lib.util.api.PLAN_EXECUTION_KILL

cryton_cli.lib.util.api.STAGE_LIST

cryton_cli.lib.util.api.STAGE_CREATE

cryton_cli.lib.util.api.STAGE_VALIDATE

cryton_cli.lib.util.api.STAGE_READ

cryton_cli.lib.util.api.STAGE_DELETE

cryton_cli.lib.util.api.STAGE_START_TRIGGER

cryton_cli.lib.util.api.STAGE_EXECUTION_LIST

cryton_cli.lib.util.api.STAGE_EXECUTION_READ

cryton_cli.lib.util.api.STAGE_EXECUTION_DELETE

cryton_cli.lib.util.api.STAGE_EXECUTION_REPORT

cryton_cli.lib.util.api.STAGE_EXECUTION_KILL

cryton_cli.lib.util.api.STAGE_EXECUTION_RE_EXECUTE

cryton_cli.lib.util.api.STEP_LIST

cryton_cli.lib.util.api.STEP_CREATE

cryton_cli.lib.util.api.STEP_VALIDATE

cryton_cli.lib.util.api.STEP_READ

cryton_cli.lib.util.api.STEP_DELETE

cryton_cli.lib.util.api.STEP_EXECUTE

cryton_cli.lib.util.api.STEP_EXECUTION_LIST

cryton_cli.lib.util.api.STEP_EXECUTION_READ

cryton_cli.lib.util.api.STEP_EXECUTION_DELETE

cryton_cli.lib.util.api.STEP_EXECUTION_REPORT

cryton_cli.lib.util.api.STEP_EXECUTION_KILL

cryton_cli.lib.util.api.STEP_EXECUTION_RE_EXECUTE

cryton_cli.lib.util.api.WORKER_LIST

cryton_cli.lib.util.api.WORKER_CREATE

cryton_cli.lib.util.api.WORKER_READ

cryton_cli.lib.util.api.WORKER_DELETE

cryton_cli.lib.util.api.WORKER_HEALTH_CHECK

cryton_cli.lib.util.api.TEMPLATE_LIST

cryton_cli.lib.util.api.TEMPLATE_CREATE

cryton_cli.lib.util.api.TEMPLATE_READ

cryton_cli.lib.util.api.TEMPLATE_DELETE

cryton_cli.lib.util.api.TEMPLATE_GET_TEMPLATE

cryton_cli.lib.util.api.EXECUTION_VARIABLE_LIST

cryton_cli.lib.util.api.EXECUTION_VARIABLE_CREATE

cryton_cli.lib.util.api.EXECUTION_VARIABLE_READ

cryton_cli.lib.util.api.EXECUTION_VARIABLE_DELETE

cryton_cli.lib.util.api.LOG_LIST

cryton_cli.lib.util.api.create_rest_api_url(*host: str*, *port: int*, *ssl: bool*) → str

Create REST API URL :param host: Address of the host :param port: Port of the host :param ssl: If true, use HTTPS, else use HTTP :return: REST API URL

cryton_cli.lib.util.api.get_request(*api_url: str*, *endpoint_url: str*, *object_id: int = None*, *custom_params: dict = None*) → str | requests.Response

Custom get request. :param api_url: API url :param endpoint_url: API endpoint url :param object_id: ID of the desired object :param custom_params: Optional dictionary containing custom params :return: API response

cryton_cli.lib.util.api.post_request(*api_url: str*, *endpoint_url: str*, *object_id: int = None*, *custom_dict: dict = None*, *files: dict = None*, *data: dict = None*) → str | requests.Response

Custom post request. :param api_url: API url :param endpoint_url: API endpoint url :param object_id: ID of the desired object :param custom_dict: instance yaml :param files: files to be sent :param data: data to be sent :return: API response

cryton_cli.lib.util.api.delete_request(*api_url: str*, *endpoint_url: str*, *object_id: int = None*) → str | requests.Response

Custom delete request. :param api_url: API url :param endpoint_url: API endpoint url :param object_id: ID of the desired object :return: API response

cryton_cli.lib.util.constants

Module Contents

cryton_cli.lib.util.constants.TIME_FORMAT = '%Y-%m-%dT%H:%M:%SZ'

cryton_cli.lib.util.constants.TIME_DETAILED_FORMAT = '%Y-%m-%dT%H:%M:%S.%fZ'

cryton_cli.lib.util.util

Module Contents

Classes

CliContext	Context object for CLI. Contains necessary data for link creation.

Functions

save_yaml_to_file(→ str)	Save content into file.
load_files(→ Dict[str, bytes])	Load files from paths.
convert_from_utc(→ datetime.datetime)	Convert datetime in UTC timezone to specified timezone
parse_response(→ Union[str, dict])	Parse the response to the desired format.
echo_msg(→ None)	Echo message containing information about success or failure of user's request.
format_result_line(→ str)	Filter dictionary values, optionally update timezone, and create user readable line.
format_list_results(→ list)	Format list response to be user readable.
echo_list(→ None)	Remove ignored parameters and echo the rest.
update_report(iterable, localize)	Go through each iterable in report and do the following:
get_yaml(→ None)	Get yaml and save/echo it.
render_documentation(→ str)	Process and create documentation in markdown.
clean_up_documentation(→ str)	Remove forbidden characters from documentation.

cryton_cli.lib.util.util.save_yaml_to_file(*content: dict*, *file_path: str*, *file_prefix: str = 'file'*) → str

Save content into file. :param file_path: Where to save the file (default is /tmp/prefix_timestamp_tail) :param content: What should be saved to the file :param file_prefix: Prefix for the file name (only if path is */tmp*) :return: Path to the file

cryton_cli.lib.util.util.load_files(*file_paths: List[str]*) → Dict[str, bytes]

Load files from paths. :param file_paths: File paths to load :return: Loaded files

cryton_cli.lib.util.util.convert_from_utc(*utc_str: str*) → datetime.datetime

Convert datetime in UTC timezone to specified timezone :param utc_str: datetime in UTC timezone to convert :return: datetime with the specified timezone

class cryton_cli.lib.util.util.CliContext(*host: str | None*, *port: int | None*, *ssl: bool*, *debug: bool*)

Bases: object

Context object for CLI. Contains necessary data for link creation.

cryton_cli.lib.util.util.parse_response(*response: requests.Response*) → str | dict

Parse the response to the desired format. :param response: Response containing data from REST API :return: Parsed response

cryton_cli.lib.util.util.echo_msg(*response: str | requests.Response*, *ok_message: str = 'Success!'*, *debug: bool = False*) → None

Echo message containing information about success or failure of user's request. :param response: Response containing data from REST API :param ok_message: Custom message for user in case of success :param debug: Show non formatted raw output :return: None

cryton_cli.lib.util.util.format_result_line(*line: dict*, *to_print: List[str]*, *localize: bool*) → str

Filter dictionary values, optionally update timezone, and create user readable line. :param line: dictionary that should be parsed :param to_print: Keys to be shown (printed out) :param localize: If datetime variables should be converted to local timezone :return: Filtered and localized string

cryton_cli.lib.util.util.format_list_results(*results: list*, *to_print: List[str]*, *localize: bool*) → list

Format list response to be user readable. :param results: Results from API :param to_print: Parameters to be shown (printed out) :param localize: If datetime variables should be converted to local timezone :return: Formatted list

cryton_cli.lib.util.util.echo_list(*response: str | requests.Response*, *to_print: List[str]*, *less: bool = False*, *localize: bool = False*, *debug: bool = False*) → None

Remove ignored parameters and echo the rest. :param response: Response containing data from REST API :param to_print: Parameters to be shown (printed out) :param less: Show less like output :param localize: If datetime variables should be converted to local timezone :param debug: Show non formatted raw output :return: None

cryton_cli.lib.util.util.update_report(*iterable: dict | list | str*, *localize: bool*)

Go through each iterable in report and do the following:

- localize it, if it is a datetime variable

Parameters

- iterable – Element in which we want to change values
- localize – If datetime variables should be converted to local timezone

Returns

None

cryton_cli.lib.util.util.get_yaml(*response: str | requests.Response*, *file_path: str*, *file_prefix: str*, *echo_only: bool = False*, *less: bool = False*, *localize: bool = False*, *debug: bool = False*) → None

Get yaml and save/echo it. :param response: Response containing data from REST API :param file_path: Where to save the file :param file_prefix: Prefix for the file name (only if path is */tmp*) :param echo_only: If the report should be only printed out and not saved :param less: Show less like output :param debug: Show non formatted raw output :param localize: If datetime variables should be converted to local timezone :return: None

cryton_cli.lib.util.util.render_documentation(*raw_documentation: dict*, *layer: int*) → str

Process and create documentation in markdown. :param raw_documentation: Unprocessed documentation :param layer: Header level :return: Documentation in Markdown

cryton_cli.lib.util.util.clean_up_documentation(*documentation: str*) → str

Remove forbidden characters from documentation. :param documentation: Human readable documentation :return: Clean documentation

Submodules

cryton_cli.lib.cli

Module Contents

Functions

cli(→ None)	A CLI wrapper for Cryton API.
generate_docs(file, layer)	Generate Markdown documentation for CLI.

cryton_cli.lib.cli.cli(*ctx: click.Context*, *host: str*, *port: int*, *secure: bool*, *debug: bool*) → None

A CLI wrapper for Cryton API.

Parameters

- ctx – Click context
- secure – True if use HTTPS for requests, else HTTP (default is False)
- debug – Show non formatted raw output
- host – Cryton's REST API url (default is None)
- port – Cryton's REST API port (default is None)

Returns
None

cryton_cli.lib.cli.generate_docs(*file: str*, *layer: int*)

Generate Markdown documentation for CLI.

FILE is path/to/your/file where you want to save the generated documentation.

Parameters

- file – Where to save the generated documentation
- layer – Highest header level

Returns
None

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

S

T

U

W