# Table of Contents

1 Usage .............................................................................. 3

2 Testing ........................................................................ 5

2.1 Maintenance ................................................................. 5
   2.1.1 New docker shas .................................................. 6
   2.1.2 Chain of Trust settings .......................................... 6
   2.1.3 Ed25519 keys ...................................................... 6

2.2 Scriptworker Releases .................................................... 6
   2.2.1 Code changes ...................................................... 6
   2.2.2 Tests and test coverage ........................................ 6
   2.2.3 Versioning ......................................................... 7
   2.2.4 Changelog ......................................................... 7
   2.2.5 Release files ...................................................... 7
   2.2.6 Tagging ............................................................. 7
   2.2.7 Pypi ................................................................. 8
   2.2.8 Rollout ............................................................. 8

2.3 Adding new scriptworker instance types .............................. 8
   2.3.1 Is scriptworker the right tool? ................................. 8
   2.3.2 Creating a new scriptworker instance type ................. 9
   2.3.3 Deployment considerations .................................... 10

2.4 Adding new scriptworker instances of an existing type .......... 11

2.5 Chain of Trust ................................................................ 11
   2.5.1 Overview ......................................................... 11
   2.5.2 Chain of Trust Key Management ............................ 12
   2.5.3 Chain of Trust Artifact Generation ....................... 12
   2.5.4 Chain of Trust Verification ................................... 13
   2.5.5 Chain of Trust Testing / debugging ....................... 15

2.6 Scriptworker Readme ..................................................... 16
   2.6.1 Usage ............................................................ 16
   2.6.2 Testing ........................................................... 16

2.7 scriptworker package .................................................... 17
   2.7.1 Submodules ...................................................... 17
   2.7.2 scriptworker.artifacts module ............................... 17
   2.7.3 scriptworker.client module .................................. 20
   2.7.4 scriptworker.config module ................................. 22
   2.7.5 scriptworker.constants module ............................. 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7.6</td>
<td>scriptworker.context module</td>
<td>23</td>
</tr>
<tr>
<td>2.7.7</td>
<td>scriptworker.cot.generate module</td>
<td>26</td>
</tr>
<tr>
<td>2.7.8</td>
<td>scriptworker.cot.verify module</td>
<td>27</td>
</tr>
<tr>
<td>2.7.9</td>
<td>scriptworker.ed25519 module</td>
<td>39</td>
</tr>
<tr>
<td>2.7.10</td>
<td>scriptworker.github module</td>
<td>40</td>
</tr>
<tr>
<td>2.7.11</td>
<td>scriptworker.exceptions module</td>
<td>42</td>
</tr>
<tr>
<td>2.7.12</td>
<td>scriptworker.log module</td>
<td>44</td>
</tr>
<tr>
<td>2.7.13</td>
<td>scriptworker.task module</td>
<td>46</td>
</tr>
<tr>
<td>2.7.14</td>
<td>scriptworker.utils module</td>
<td>52</td>
</tr>
<tr>
<td>2.7.15</td>
<td>scriptworker.worker module</td>
<td>61</td>
</tr>
<tr>
<td>2.7.16</td>
<td>Module contents</td>
<td>61</td>
</tr>
<tr>
<td>2.8</td>
<td>Indices and tables</td>
<td>62</td>
</tr>
</tbody>
</table>

**Python Module Index**

**Index**
Scriptworker implements the TaskCluster worker model, then launches a pre-defined script.
This worker was designed for Releng processes that need specific, limited, and pre-defined capabilities.

Free software: MPL2 License
• Create a config file. By default scriptworker will look in 

```
./scriptworker.yaml
```

but this config path can be specified as the first and only commandline argument. There is an example config file, and all config items are specified in 

```
scriptworker.constants.DEFAULT_CONFIG
```

Credentials can live in 

```
./scriptworker.yaml
./secrets.json
```

• **Launch:** `scriptworker [config_path]`
Without integration tests install tox, then

```bash
NO_CREDENTIALS_TESTS=1 tox -e py36
```

Without any tests connecting to the net, then

```bash
NO_TESTS_OVER_WIRE=1 tox -e py36
```

With integration tests, first create a client in the Taskcluster UI with the scopes:

```plaintext
queue:cancel-task:test-dummy-scheduler/*
queue:claim-work:test-dummy-provisioner/dummy-worker-*
queue:create-task:lowest:test-dummy-provisioner/dummy-worker-*
queue:define-task:test-dummy-provisioner/dummy-worker-*
queue:get-artifact:SampleArtifacts/_/X.txt
queue:scheduler-id:test-dummy-scheduler
queue:schedule-task:test-dummy-scheduler/*
queue:task-group-id:test-dummy-scheduler/*
queue:worker-id:test-dummy-workers/dummy-worker-*
```

Then generate a no priviledge personal access token in Github for the scriptworker_github_token (to avoid rate limiting) and create a `.secrets.json` or `~/.scriptworker` that looks like:

```json
{
   "integration_credentials": {
      "clientId": "...",
      "accessToken": "...",
   
   "scriptworker_github_token": "...
   
```

then to run all tests: tox

## 2.1 Maintenance

For sheriffs, release/relops, taskcluster, or related users, this page describes maintenance for scriptworkers.
2.1.1 New docker shas

For chain of trust verification, we verify the docker shas that we run in docker-worker.

For some tasks, we build the docker images in docker-image tasks, and we can verify the image’s sha against docker-image task’s output.

However, for decision and docker-image tasks, we download the docker image from docker hub. We allowlist the shas to make sure we are running valid images.

We specify those here. However, if we only specified them in `scriptworker.constants`, we’d have to push a new scriptworker release every time we update this allowlist. So we override this list here.

For now, we need to keep both locations updated. Puppet governs production instances, and the scriptworker repo is used for scriptworker development, and a full allowlist is required for chain of trust verification.

2.1.2 Chain of Trust settings

As above, other chain of trust settings live in `constants.py`. However, if we only specified them in `scriptworker.constants`, we’d have to push a new scriptworker release every time we update them. So we can override them here.

Ideally we keep the delta small, and remove the overrides in puppet when we release a new scriptworker version that updates these defaults. As currently written, each scriptworker instance type will need its scriptworker version bumped individually.

2.1.3 Ed25519 keys

For ed25519 key maintenance, see the chain of trust docs

2.2 Scriptworker Releases

These are the considerations and steps for a new scriptworker release.

2.2.1 Code changes

Ideally, code changes should follow clean architecture best practices

When adding new functions, classes, or files, or when changing function arguments, please add or modify the doc-strings.

2.2.2 Tests and test coverage

Scriptworker has 100% test coverage, and we’d like to keep it that way.

Run tests locally via tox to make sure the tests pass, and we still have 100% coverage.
2.2.3 Versioning

Scriptworker follows semver. Essentially, increment the
   1. MAJOR version when you make incompatible API changes,
   2. MINOR version when you add API functionality in a backwards-compatible manner, and
   3. PATCH version when you make backwards-compatible bug fixes.

2.2.4 Changelog

Update the changelog before making a new release.

2.2.5 Release files

If you’re changing any dependencies, please update setup.py.
If you add change the list of files that need to be packaged (either adding new files, or removing previous packaged files), modify MANIFEST.in.

Versioning

Modify src/scriptworker/version.py to set the __version__ to the appropriate tuple. This is either a 3- or 4-part tuple, e.g.

```
# 0.10.0a1
__version__ = (0, 10, 0, "alpha1")

# 1.0.0b2
__version__ = (1, 0, 0, "beta2")

# 0.9.3
__version__ = (0, 9, 3)
```

Then run version.py:

```
# Using the local venv python>=3.5, python src/scriptworker/version.py
```

This will update version.json. Verify both files look correct.

2.2.6 Tagging

To enable gpg signing in git,
   1. you need a gpg keypair
   2. you need to set your user.signingkey in your ~/.gitconfig or scriptworker/.git/config
   3. If you want to specify a specific gpg executable, specify your gpg.program in your ~/.gitconfig or scriptworker/.git/config

Tag and sign!
# make sure you've committed your changes first!
VERSION=0.9.0
git tag -s $VERSION -m"$VERSION"

Push!

```bash
# By default this will push the new tag to origin; make sure the tag gets pushed to
# mozilla-releng/scriptworker
git push --tags
```

## 2.2.7 Pypi

Someone with access to the scriptworker package on pypi.python.org needs to do the following:

```bash
# from https://packaging.python.org/tutorials/distributing-packages/#uploading-your-project-to-pypi
# Don't use 'python setup.py register' or 'python setup.py upload'; this may use
# cleartext auth!
# Using a python with 'twine' in the virtualenv:
VERSION=4.1.2
# create the source tarball and wheel
python setup.py sdist bdist_wheel
# upload the source tarball + wheel
twine upload dist/scriptworker-$VERSION.tar.gz dist/scriptworker-$VERSION-py2.-py3-none-any.whl
```

That creates source tarball and wheel, and uploads it.

## 2.2.8 Rollout

To roll out to the scriptworker kubernetes pools, wait for a few minutes after the pypi upload, run pin.sh to bump all the workers, get review on that change, then push to the `production` branch when you’re ready to roll out. (Use the `production-___script` branches if you only want to update scriptworker on a subset of workers.) We’ll see a notification in Slack `#releng-notifications` as each pool rolls out.

To roll out to the mac signers, get review for a patch like this and merge. The mac signer maintainers will roll out to the `production-mac-signing` branch as appropriate. We don’t get notifications anywhere, so we likely just need to give this a half hour or so to roll out.

## 2.3 Adding new scriptworker instance types

This doc describes when and how to add a new scriptworker instance type, e.g. signing, pushapk, beetmover, balrog.

Last updated 2016.11.18

### 2.3.1 Is scriptworker the right tool?

Scriptworker is designed to run security-sensitive tasks with limited capabilities.

- Does this task require elevated privileges or access to sensitive secrets to run?
- Is this task sufficiently important to spin up and maintain a new pool of workers?
• Is the expected load sufficiently contained so we don’t require a dynamically sizable pool of workers?

If you answered yes to the above, scriptworker may be a good option.

Chain of Trust considerations

If this is for a new task type in a product/graph that already has scriptworker tasks and chain of trust verification, then adding a new task should be an incremental change.

If this is for a new graph type or new product, and the graph doesn’t look like the Firefox graph, there may be significant changes required to support the chain of trust. This is an important consideration when choosing your solution.

2.3.2 Creating a new scriptworker instance type

Once you’ve decided to use scriptworker, these are the steps to take to implement.

Write a script

This can be a script in any language that can be called from the commandline, although we prefer async python 3. This is standalone, so it’s possible to develop and test this script without scriptworker.

Single purpose, but generic

The script should aim to support a single purpose, like signing or pushing updates. However, ideally it’s generic, so it can sign a number of different file types or push various products to various accounts, given the right config and creds.

Commandline args

Currently, we call the script from scriptworker with the commandline

```
# e.g., ["python", "/path/to/script.py", "/path/to/script_config.json"]
[interpreter, script, config]
```

Where interpreter could be python3, script is the path to the script, and config is the path to the runtime configuration, which doesn’t change between runs.

Config

The config could be anything you need the script to know, including paths to other config files. These config items must be specified, and must match the eventual scriptworker config:

- **work_dir**: this is an absolute path. This directory is deleted after the task and recreated before the next task. Scriptworker will place files in here for the script’s consumption.

- **artifact_dir**: this is where to put the artifacts that scriptworker will upload to taskcluster. The directory layout will look like the directory layout in taskcluster, e.g. public/build/target.apk or public/logs/foo.log
Task

Scriptworker will place the task definition in $work_dir/task.json. The script can read this task definition and behave accordingly.

When testing locally without scriptworker, you can create your own task.json.

**task.payload.upstreamArtifacts**

If the task defines `payload.upstreamArtifacts`, these are artifacts for scriptworker to download and verify their shas against the chain of trust.

`payload.upstreamArtifacts` currently looks like:

```json
[{
   "taskId": "upstream-task-id1",
   "taskType": "build",
   "paths": ["public/artifact/path1", "public/artifact/path2"],
   "formats": []
}, {
   ...
}]
```

It will download them into $artifact_dir/public/cot/$upstream-task-id/$path.

**Scopes**

Taskcluster scopes are its ACLs: restricted behavior is placed behind scopes, and only those people and processes that need access to that behavior are given those scopes. With the Chain of Trust, we can verify that restricted scopes can only be used in specific repos.

If your script is going to have different levels of access (e.g., CI- signing, nightly- signing, and release- signing), then it’s best to put them each behind a different scope, and use that scope for determining which credentials to use.

**2.3.3 Deployment considerations**

You don’t have to address the below during script development, but it may be helpful to know some of the considerations that will affect deployment.

**Graph**

We need to trace upstream tasks back to the tree. We’re able to find our decision task by the `taskGroupId`, but other dependencies we need to either use `upstreamArtifacts` or `task.extra.chainOfTrust.inputs`, which looks like

```json
"inputs": {
   "docker-image": "docker-image-taskid"
}
```

If there are upstream tasks that depend on the output of other tasks, make sure all of them are connected via at least one of these two data structures.
GCP

For more information on deploying this to GCP, please consult the scriptworker-scripts documentation.

### 2.4 Adding new scriptworker instances of an existing type

As of November 2019, we switched over to auto-scalable Kubernetes and tagged along the underlying infrastructure to GCP. Ramping more workers of an existing scriptworker simply means scaling some configurations. You can find out more information on this here.

### 2.5 Chain of Trust

#### 2.5.1 Overview

Taskcluster is versatile and self-serve, and enables developers to make automation changes without being blocked on other teams. In the case of developer testing and debugging, this is very powerful and enabling. In the case of release automation, the ability to schedule arbitrary tasks with arbitrary configs can present a security concern.

The chain of trust is a second factor that isn’t automatically compromised if scopes are compromised. This chain allows us to trace a task’s request back to the tree.

#### High level view

Scopes are how Taskcluster controls access to certain features. These are granted to roles, which are granted to users or LDAP groups.

Scopes and their associated Taskcluster credentials are not leak-proof. Also, by their nature, more people will have restricted scopes than you want, given any security-sensitive scope. Without the chain of trust, someone with release-signing scopes would be able to schedule any arbitrary task to sign any arbitrary binary with the release keys, for example.

The chain of trust is a second factor. The embedded ed25519 keys on the workers are either the something you have or the something you are, depending on how you view the taskcluster workers.

Each chain-of-trust-enabled taskcluster worker generates and signs chain of trust artifacts, which can be used to verify each task and its artifacts, and trace a given request back to the tree.

The scriptworker nodes are the verification points. Scriptworkers run the release sensitive tasks, like signing and publishing releases. They verify their task definitions, as well as all upstream tasks that generate inputs into their task. Any broken link in the chain results in a task exception.

In conjunction with other best practices, like separation of roles, we can reduce attack vectors and make penetration attempts more visible, with task exceptions on release branches.

#### Chain of Trust Versions

1. Initial Chain of Trust implementation with GPG signatures: Initial 1.0.0b1 on 2016-11-14
2. CoT v2: rebuild task definitions via json-e. 7.0.0 on 2018-01-18
3. Generic action hook support. 12.0.0 on 2018-05-29
4. Release promotion action hook support. 17.1.0 on 2018-12-28
5. ed25519 support; deprecate GPG support. 22.0.0 on 2019-03-07
6. drop support for gpg 23.0.0 on 2019-03-27
7. drop support for non-hook actions 41.0.0 on 2021-09-02

2.5.2 Chain of Trust Key Management

Ed25519 key management is a critical part of the chain of trust. There are valid ed25519 keys per worker implementation (docker-worker, generic-worker, and scriptworker).

Base64-encoded seeds that can be converted to valid level 3 ed25519 pubkeys are recorded in scriptworker.constants, in DEFAULT_CONFIG['ed25519_public_keys']. These are tuples to allow for key rotation.

At some point we may add per-cot-project sets of pubkeys. We may also move the source of truth of these pubkeys to a separate location, to enable cot signature verification elsewhere, outside of scriptworker.

verifying new ed25519 keys

The verify_cot commandline tool supports a --verify-sigs option. This will turn on signature verification, and will break if the cot artifacts are not signed by valid level 3 ed25519 keys.

There is also a verify_ed25519_signature commandline tool. This takes a file path and a signature path, and verifies if the file was validly signed by a known valid level 3 key. It also takes an optional --pubkey PUBKEY argument, which allows you to verify if the file was signed by that pubkey.

Rotating the FirefoxCI CoT keys

See this mana page.

2.5.3 Chain of Trust Artifact Generation

Each chain-of-trust-enabled taskcluster worker generates and uploads a chain of trust artifact after each task. This artifact contains details about the task, worker, and artifacts, and is signed by the embedded ed25519 key.

Embedded ed25519 keys

Each supported taskcluster workerType has an embedded ed25519 keypair. These are the second factor.

docker-worker has the ed25519 privkey embedded in the AMI, inaccessible to tasks run inside the docker container.
generic-worker can embed the ed25519 privkey into the AMI for EC2 instances, or into the system directories for hardware. This are permissioned so the task user doesn’t have access to it.

Chain-of-Trust-enabled scriptworker workers have a valid ed25519 keypair.

The pubkeys for trusted workerTypes are recorded in scriptworker.constants.ed25519_public_keys.
Chain of Trust artifacts

After the task finishes, the worker creates a chain of trust JSON blob, ed25519 signs it, then uploads it as public/chain-of-trust.json and its detached signature, public/chain-of-trust.json.sig. It looks like

```json
{
    "artifacts": {
        "path/to/artifact": {
            "sha256": "abcd1234"
        },
        ...
    },
    "chainOfTrustVersion": 1,
    "environment": {
        # worker-impl specific stuff, like ec2 instance id, ip
    },
    "runId": 0,
    "task": {
        # task defn
    },
    "taskId": "...",
    "workerGroup": "...",
    "workerId": "...
}
```

- The v1 chain-of-trust json artifact schema is viewable here.
- This is a real example artifact.

2.5.4 Chain of Trust Verification

Currently, only chain-of-trust-enabled scriptworker instances verify the chain of trust. These are tasks like signing, publishing, and submitting updates to the update server. If the chain of trust is not valid, scriptworker kills the task before it performs any further actions.

The below is how this happens.

**Decision Task**

The decision task is a special task that generates a taskgraph, then submits it to the Taskcluster queue. This graph contains task definitions and dependencies. The decision task uploads its generated graph JSON as an artifact, which can be inspected during chain of trust verification.

We rebuild the decision task’s task definition via json-e, and verify that it matches the runtime task definition.

**Ed25519 key management**

The chain of trust artifacts are signed. We need to keep track of the ed25519 public keys to verify them.

We keep the level 3 gecko pubkeys in scriptworker.constants.ed25519_public_keys, as base64-encoded ascii strings. Once decoded, these are the seeds for the ed25519 public keys. These are tuples of valid keys, to allow for key rotation.
Building the chain

First, scriptworker inspects the [signing/balrog/pushapk/beetmover/etc] task that it claimed from the Taskcluster queue. It adds itself and its decision-task to the chain.

Any task that generates artifacts for the scriptworker then needs to be inspected. For scriptworker tasks, we have `task.payload.upstreamArtifacts`, which looks like

```
[
  {
    "taskId": "upstream-task-id",
    "taskType": "build", # for cot verification purposes
    "paths": ["path/to/artifact1", "path/to/artifact2"],
    "formats": ["gpg", "jar"] # This is signing-specific for now; we could make
    
  },
  ...
]
```

We add each upstream `taskId` to the chain, with corresponding `taskType` (we use this to know how to verify the task).

For each task added to the chain, we inspect the task definition, and add other upstream tasks:

- if the decision task doesn’t match, add it to the chain.
- docker-worker tasks have `task.extra.chainOfTrust.inputs`, which is a dictionary like `{"docker-image": "docker-image-taskid"}`. Add the docker image `taskId` to the chain (this will likely have a different decision `taskId`, so add that to the chain).

Verifying the chain

Scriptworker:

- downloads the chain of trust artifacts for each upstream task in the chain, and verifies their signatures. This requires detecting which worker implementation each task is run on, to know which ed25519 public key to use. At some point in the future, we may switch to an OpenSSL CA.
- downloads each of the `upstreamArtifacts` and verify their shas against the corresponding task’s chain of trust’s artifact shas. the downloaded files live in `cot/TASKID/PATH`, so the script doesn’t have to re-download and re-verify.
- downloads each decision task’s `task-graph.json`. For every other task in the chain, we make sure that their task definition matches a task in their decision task’s task graph.
- rebuilds decision and action task definitions using `json-e`, and verifies the rebuilt task definition matches the runtime definition.
- verifies each docker-worker task is either part of the `prebuild_docker_image_task_types`, or that it downloads its image from a previous docker-image task.
- verifies each docker-worker task’s docker image sha.
- makes sure the `interactive` flag isn’t on any docker-worker task.
- determines which repo we’re building off of.
- matches its task’s scopes against the tree; restricted scopes require specific branches.

Once all verification passes, it launches the task script. If chain of trust verification fails, it exits before launching the task script.
### 2.5.5 Chain of Trust Testing / debugging

The `verify_cot` entry point allows you to test chain of trust verification without running a scriptworker instance locally.

#### Create the virtualenv

- Install git, `python>=3.6`, and `python3 virtualenv`.
- Clone scriptworker and create virtualenv:

  ```bash
  git clone https://github.com/mozilla-releng/scriptworker
cd scriptworker
virtualenv3 venv
. venv/bin/activate
python setup.py develop
  ```

#### Set up the test env

- Create a `~/.scriptworker` or `./secrets.json` with test client creds.
- Create the client at the client manager. Mine has the `assume:project:taskcluster:worker-test-scopes` scope, but I don’t think that’s required.
- The `~/.scriptworker` or `./secrets.json` file will look like this (fill in your clientId and accessToken):

  ```json
  {
  "credentials": {
    "clientId": "mozilla-ldap/asasaki@mozilla.com/signing-test",
    "accessToken": "********"
  }
  }
  ```

#### Find a task to test

- Find a cot-enabled task on treeherder to test.
- Click it, click ‘inspect task’ in the lower left corner.
- The taskId will be in a field near the top of the page.

#### Run the test

- Now you should be able to test chain of trust verification!

  ```bash
  verify_cot --task-type TASKTYPE TASKID  # e.g., verify_cot --task-type signing _→cbYd3U6dRRCKPUbKsEj1Iw
  ```

- To test with signature verification, use the `--verify-sigs` option. This only works for level 3 trusted workers, since we don’t keep track of the other pubkeys..
2.6 Scriptworker Readme

Scriptworker implements the TaskCluster worker model, then launches a pre-defined script.

This worker was designed for Releng processes that need specific, limited, and pre-defined capabilities.

Free software: MPL2 License

2.6.1 Usage

- Create a config file. By default scriptworker will look in ./scriptworker.yaml, but this config path can be specified as the first and only commandline argument. There is an example config file, and all config items are specified in scriptworker.constants.DEFAULT_CONFIG.

Credentials can live in ./scriptworker.yaml, ./secrets.json, ~/.scriptworker.

- Launch: scriptworker [config_path]

2.6.2 Testing

Without integration tests install tox, then

NO_CREDENTIALS_TESTS=1 tox -e py36

Without any tests connecting to the net, then NO_TESTS_OVER_WIRE=1 tox -e py36

With integration tests, first create a client in the Taskcluster UI with the scopes:

```
queue:cancel-task:test-dummy-scheduler/*
queue:claim-work:test-dummy-provisioner/dummy-worker-*
queue:create-task:lowest:test-dummy-provisioner/dummy-worker-*
queue:define-task:test-dummy-provisioner/dummy-worker-*
queue:get-artifact:SampleArtifacts/_/X.txt
queue:scheduler-id:test-dummy-scheduler
queue:schedule-task:test-dummy-scheduler/*
queue:task-group-id:test-dummy-scheduler/*
queue:worker-id:test-dummy-workers/dummy-worker-*
```

Then generate a no privilege personal access token in Github for the scriptworker_github_token (to avoid rate limiting) and create a ./secrets.json or ~/.scriptworker that looks like:

```json
{
    "integration_credentials": {
        "clientId": "...",
        "accessToken": "..."
    },
    "scriptworker_github_token": "..."
}
```

then to run all tests: tox
2.7 scriptworker package

2.7.1 Submodules

2.7.2 scriptworker.artifacts module

Scriptworker artifact-related operations.

Importing this script updates the mimetypes database. This maps some known extensions to text/plain for a better storage in S3.

```python
scriptworker.artifacts.assert_is_parent(path, parent_dir)
```

Raise ScriptworkerTaskException if path is not under parent_dir.

Parameters

- **path (str)** – the path to inspect.
- **parent_dir (str)** – the path that path should be under.

Raises ScriptworkerTaskException – if path is not under parent_dir.

```python
scriptworker.artifacts.compress_artifact_if_supported(artifact_path)
```

Compress artifacts with GZip if they're known to be supported.

This replaces the artifact given by a gzip binary.

Parameters **artifact_path (str)** – the path to compress

Returns Type and encoding of the file. Encoding equals ‘gzip’ if compressed.

Return type content_type, content_encoding (tuple)

```python
scriptworker.artifacts.create_artifact(context, path, target_path, content_type, content_encoding, storage_type='s3', expires=None)
```

Create an artifact and upload it.

This should support s3 and azure out of the box; we’ll need some tweaking if we want to support redirect/error artifacts.

Parameters

- **context (scriptworker.context.Context)** – the scriptworker context.
- **path (str)** – the path of the file to upload.
- **target_path (str)** –
- **content_type (str)** – Content type (MIME type) of the artifact. Values can be found via scriptworker.artifacts.guess_content_type_and_encoding()
- **content_encoding (str)** – Encoding (per mimetypes’ library) of the artifact. None is for no encoding. Values can be found via scriptworker.artifacts.guess_content_type_and_encoding()
- **storage_type (str, optional)** – the taskcluster storage type to use. Defaults to ‘s3’
- **expires (str, optional)** – datestring of when the artifact expires. Defaults to None.

Raises ScriptWorkerRetryException – on failure.
scriptworker.artifacts.download_artifacts(context, file_urls, parent_dir=None, session=None, download_func=download_file, valid_artifact_task_ids=None)

Download artifacts in parallel after validating their URLs.

Valid taskId's for download include the task's dependencies and the taskGroupId, which by convention is the taskId of the decision task.

Parameters

• `context` (scriptworker.context.Context) – the scriptworker context.

• `file_urls` (list) – the list of artifact urls to download.

• `parent_dir` (str, optional) – the path of the directory to download the artifacts into. If None, defaults to `work_dir`. Default is None.

• `session` (aiohttp.ClientSession, optional) – the session to use to download. If None, defaults to context.session. Default is None.

• `download_func` (function, optional) – the function to call to download the files. Default is `download_file`.

• `valid_artifact_task_ids` (list, optional) – the list of task ids that are valid to download from. If None, defaults to all task dependencies plus the decision taskId. Defaults to None.

Returns the full paths to the files downloaded

Return type list

Raises scriptworker.exceptions.BaseDownloadError – on download failure after any applicable retries.

scriptworker.artifacts.get_and_check_single_upstream_artifact_full_path(context, task_id, path)

Return the full path where an upstream artifact is located on disk.

Parameters

• `context` (scriptworker.context.Context) – the scriptworker context.

• `task_id` (str) – the task id of the task that published the artifact

• `path` (str) – the relative path of the artifact

Returns absolute path to the artifact

Return type str

Raises scriptworker.exceptions.ScriptWorkerTaskException – when an artifact doesn’t exist.

scriptworker.artifacts.get_artifact_url(context, task_id, path)

Get a TaskCluster artifact url.

Parameters

• `context` (scriptworker.context.Context) – the scriptworker context

• `task_id` (str) – the task id of the task that published the artifact

• `path` (str) – the relative path of the artifact

Returns the artifact url
Return type  str

Raises TaskClusterFailure – on failure.

scriptworker.artifacts.get_expiration_arrow (context)
Return an arrow matching context.task['expires'].

Parameters context (scriptworker.context.Context) – the scriptworker context

Returns context.task['expires'].

Return type  arrow

scriptworker.artifacts.get_optional_artifacts_per_task_id (upstream_artifacts)
Return every optional artifact defined in upstream_artifacts, ordered by taskId.

Parameters upstream_artifacts – the list of upstream artifact definitions

Returns list of paths to downloaded artifacts ordered by taskId

Return type  dict

scriptworker.artifacts.get_single_upstream_artifact_full_path (context, task_id, path)
Return the full path where an upstream artifact should be located.

Artifact may not exist. If you want to be sure if does, use get_and_check_single_upstream_artifact_full_path() instead.

This function is mainly used to move artifacts to the expected location.

Parameters

• context (scriptworker.context.Context) – the scriptworker context.

• task_id (str) – the task id of the task that published the artifact

• path (str) – the relative path of the artifact

Returns absolute path to the artifact should be.

Return type  str

scriptworker.artifacts.get_upstream_artifacts_full_paths_per_task_id (context)
List the downloaded upstream artifacts.

Parameters context (scriptworker.context.Context) – the scriptworker context.

Returns

lists of the paths to upstream artifacts, sorted by task_id. First dict represents the existing upstream artifacts. The second one maps the optional artifacts that couldn’t be downloaded

Return type  dict, dict

Raises scriptworker.exceptions.ScriptWorkerTaskException – when an artifact doesn’t exist.

scriptworker.artifacts.guess_content_type_and_encoding (path)
Guess the content type of a path, using mimetypes.

Falls back to “application/binary” if no content type is found.

Parameters path (str) – the path to guess the mimetype of

Returns the content type of the file

Return type  str
scriptworker.artifacts.retry_create_artifact(*args, **kwargs)

Retry create_artifact() calls.

Parameters

• *args – the args to pass on to create_artifact
• **kwargs – the args to pass on to create_artifact

scriptworker.artifacts.upload_artifacts(context, files)

Compress and upload the requested files from artifact_dir, preserving relative paths.

Compression only occurs with files known to be supported.

This function expects the directory structure in artifact_dir to remain the same. So if we want the files in public/..., create an artifact_dir/public and put the files in there.

Parameters

• context (scriptworker.context.Context) – the scriptworker context.
• files (list of str) – files that should be uploaded as artifacts

Raises Exception – any exceptions the tasks raise.

2.7.3 scriptworker.client module

Scripts running in scriptworker will use functions in this file.

This module should be largely standalone. This should only depend on scriptworker.exceptions and scriptworker.constants, or other standalone modules, to avoid circular imports.

scriptworker.client.log

the log object for the module

Type logging.Logger

scriptworker.client.get_task(config: Dict[str, Any]) → Dict[str, Any]

Read the task.json from work_dir.

Parameters config (dict) – the running config, to find work_dir.

Returns the contents of task.json

Return type dict

Raises ScriptWorkerTaskException – on error.

scriptworker.client.sync_main(async_main: Callable[[Any], Awaitable[None]], config_path: Optional[str] = None, default_config: Optional[Dict[str, Any]] = None, should_validate_task: bool = True, loop_function: Callable[[], asyncio.events.AbstractEventLoop] = <built-in function get_event_loop>) → None

Entry point for scripts using scriptworker.

This function sets up the basic needs for a script to run. More specifically:

• it creates the scriptworker context and initializes it with the provided config
• the path to the config file is either taken from config_path or from sys.argv[1].
• it verifies sys.argv doesn’t have more arguments than the config path.
• it creates the asyncio event loop so that async_main can run
Parameters

- **async_main** *(function)* – The function to call once everything is set up
- **config_path** *(str, optional)* – The path to the file to load the config from. Loads from `sys.argv[1]` if None. Defaults to None.
- **default_config** *(dict, optional)* – The default config to use for _init_context_. Defaults to None.
- **should_validate_task** *(bool, optional)* – Whether we should validate the task schema. Defaults to True.
- **loop_function** *(function, optional)* – The function to call to get the event loop; here for testing purposes. Defaults to `asyncio.get_event_loop`.

```python
scriptworker.client.validate_artifact_url(valid_artifact_rules: Tuple[Any], valid_artifact_task_ids: List[str], url: str) → str
```

Ensure a URL fits in given scheme, netloc, and path restrictions.

If we fail any checks, raise a ScriptWorkerTaskException with `malformed-payload`.

Parameters

- **valid_artifact_rules** *(tuple)* – The tests to run, with schemas, netlocs, and path regexes.
- **valid_artifact_task_ids** *(list)* – The list of valid task IDs to download from.
- **url** *(str)* – The url of the artifact.

Returns the filepath of the path regex.

Return type *str*

Raises ScriptWorkerTaskException – on failure to validate.

```python
scriptworker.client.validate_json_schema(data: Dict[str, Any], schema: Dict[str, Any], name: str = 'task') → None
```

Given data and a jsonschema, let’s validate it.

This happens for tasks and chain of trust artifacts.

Parameters

- **data** *(dict)* – The json to validate.
- **schema** *(dict)* – The jsonschema to validate against.
- **name** *(str, optional)* – The name of the json, for exception messages. Defaults to “task”.

Raises ScriptWorkerTaskException – on failure

```python
scriptworker.client.validate_task_schema(context: Any, schema_key: str = 'schema_file') → None
```

Validate the task definition.

Parameters

- **context** *(scriptworker.context.Context)* – The scriptworker context. It must contain a task and the config pointing to the schema file
- **schema_key** – The key in `context.config` where the path to the schema file is. Key can contain dots (e.g.: ‘schema_files.file_a’), in which case
2.7.4 scriptworker.config module

Config for scriptworker.

scriptworker.config.log
the log object for the module.

Type  logging.Logger

scriptworker.config.CREDS_FILES
an ordered list of files to look for taskcluster credentials, if they aren’t in the config file or environment.

Type  tuple

scriptworker.config.apply_product_config(config)
Apply config values that are keyed by cot_product.

This modifies the passed in configuration.

Parameters  dict(config) – the config to apply cot_product keying too

Returns: dict

scriptworker.config.check_config(config, path)
Validate the config against DEFAULT_CONFIG.

Any unknown keys or wrong types will add error messages.

Parameters

• config(dict) – the running config.
• path(str) – the path to the config file, used in error messages.

Returns  the error messages found when validating the config.

Return type  list

scriptworker.config.create_config(config_path='scriptworker.yaml')
Create a config from DEFAULT_CONFIG, arguments, and config file.

Then validate it and freeze it.

Parameters  config_path(str, optional) – the path to the config file. Defaults to “scriptworker.yaml”

Returns  (config immutabledict, credentials dict)

Return type  tuple

Raises  SystemExit – on failure

scriptworker.config.get_context_from_cmdln(args, desc='Run scriptworker')
Create a Context object from args.

Parameters  args(list) – the commandline args. Generally sys.argv

Returns

scriptworker.context.Context with populated config, and credentials immutable-dict

Return type  tuple
scriptworker.config.get_frozen_copy(values)
Convert values's list values into tuples, and dicts into immutabledicts.

   Parameters values (dict/list) – the values/list to be modified in-place.

scriptworker.config.get_unfrozen_copy(values)
Recursively convert value’s tuple values into lists, and immutabledicts into dicts.

   Parameters values (immutabledict/tuple) – the immutabledict/tuple.

   Returns the unfrozen copy.

   Return type values (dict/list)

scriptworker.config.read_worker_creds(key='credentials')
Get credentials from CREDS_FILES or the environment.
This looks at the CREDS_FILES in order, and falls back to the environment.

   Parameters key (str, optional) – each CREDS_FILE is a json dict. This key’s value contains the credentials. Defaults to ‘credentials’.

   Returns the credentials found. None if no credentials found.

   Return type dict

2.7.5 scriptworker.constants module

scriptworker constants.

scriptworker.constants.DEFAULT_CONFIG
the default config for scriptworker. Running configs are validated against this.

   Type immutabledict

scriptworker.constants.STATUSES
maps taskcluster status (string) to exit code (int).

   Type dict

scriptworker.constants.get_reversed_statuses(context: Any) → Dict[int, str]
Return a mapping of exit codes to status strings.

   Parameters context (scriptworker.context.Context) – the scriptworker context

   Returns the mapping of exit codes to status strings.

   Return type dict

2.7.6 scriptworker.context module

scriptworker context.
Most functions need access to a similar set of objects. Rather than having to pass them all around individually or create a monolithic ‘self’ object, let’s point to them from a single context object.

scriptworker.context.log
the log object for the module.

   Type logging.Logger
scriptworker.context.DEFAULT_MAX_CONCURRENT_DOWNLOADS
default max concurrent downloads

    Type  int
class scriptworker.context.Context
    Bases: object
    Basic config holding object.
    Avoids putting everything in single monolithic object, but allows for passing around config and easier overriding
    in tests.
    config
        the running config. In production this will be an immutabledict.
        Type  dict
    credentials_timestamp
        the unix timestamp when we last updated our credentials.
        Type  int
    proc
        when launching the script, this is the process object.
        Type  task_process.TaskProcess
    queue
        the taskcluster Queue object containing the scriptworker credentials.
        Type  taskcluster.aio.Queue
    session
        the default aiohttp session
        Type  aiohttp.ClientSession
    task
        the task definition for the current task.
        Type  dict
    temp_queue
        the taskcluster Queue object containing the task-specific temporary credentials.
        Type  taskcluster.aio.Queue
    claim_task
        The current or most recent claimTask definition json from the queue.
        This contains the task definition, as well as other task-specific info.
        When setting claim_task, we also set self.task and self.temp_credentials, zero out
        self.reclaim_task and self.proc, then write a task.json to disk.
        Type  dict
    config = None
create_queue(credentials: Optional[Dict[str, Any]]) → Optional[taskcluster.generated.aio.queue.Queue]
Create a taskcluster queue.

    Parameters credentials (dict) – taskcluster credentials.
credentials

The current scriptworker credentials.

These come from the config or CREDS_FILES or environment.

When setting credentials, also create a new self.queue and update self.credentials_timestamp.

    Type dict

credentials_timestamp = None
download_semaphore
event_loop

the running event loop.

This fixture mainly exists to allow for overrides during unit tests.

    Type asyncio.BaseEventLoop

populate_projects (force: bool = False) → None

Download the projects.yml file and populate self.projects.

This only sets it once, unless force is set.

    Parameters force (bool, optional) – Re-run the download, even if self.projects is already defined. Defaults to False.

proc = None
projects

The current contents of projects.yml, which defines CI configuration.

I’d love to auto-populate this; currently we need to set this from the config’s project_configuration_url.

    Type dict

queue = None
reclaim_task

The most recent reclaimTask definition.

This contains the newest expiration time and the newest temp credentials.

When setting reclaim_task, we also set self.temp_credentials.

reclaim_task will be None if there hasn’t been a claimed task yet, or if a task has been claimed more recently than the most recent reclaimTask call.

    Type dict

running_tasks = None
session = None
task = None
task_id

The running task’s taskId.

    Type string
temp_credentials

The latest temp credentials, or None if we haven’t claimed a task yet.

When setting, create self.temp_queue from the temp taskcluster creds.
temp_queue = None

verify_task() → None
Run some task sanity checks on self.task.

write_json(path: str, contents: Dict[str, Any], message: str) → None
Write json to disk.

Parameters

- **path** *(str)* – the path to write to
- **contents** *(dict)* – the contents of the json blob
- **message** *(str)* – the message to log

### 2.7.7 `scriptworker.cot.generate` module

Chain of Trust artifact generation.

**scriptworker.cot.generate.log**
the log object for this module.

Type  logging.Logger

**scriptworker.cot.generate.ed25519_private_key_from_file(path)**
Create an ed25519 key from the contents of path.

Path is a filepath containing a base64-encoded ed25519 key seed.

Parameters

- **fn** *(callable)* – the function to call with the contents from path
- **path** *(str)* – the file path to the base64-encoded key seed.

Returns  the appropriate key type from path

Return type  obj

Raises  ScriptWorkerEd25519Error

**scriptworker.cot.generate.generate_cot(context, parent_path=None)**
Format and sign the cot body, and write to disk.

Parameters

- **context** *(scriptworker.context.Context)* – the scriptworker context.
- **parent_path** *(str, optional)* – The directory to write the chain of trust artifacts to. If None, this is artifact_dir/public/. Defaults to None.

Returns  the contents of the chain of trust artifact.

Return type  str

Raises  ScriptWorkerException – on schema error.

**scriptworker.cot.generate.generate_cot_body(context)**
Generate the chain of trust dictionary.

This is the unsigned and unformatted chain of trust artifact contents.

Parameters  **context** *(scriptworker.context.Context)* – the scriptworker context.
Returns the unsign and unformatted chain of trust artifact contents.

Return type dict

Raises ScriptWorkerException – on error.

`scriptworker.cot.generate.get_cot_artifacts(context)`
Generate the artifact relative paths and shas for the chain of trust.

Parameters
- **context** (scriptworker.context.Context) – the scriptworker context.

Returns a dictionary of `{"path/to/artifact": {"hash_alg": "..."}, ...}

Return type dict

`scriptworker.cot.generate.get_cot_environment(context)`
Get environment information for the chain of trust artifact.

Parameters
- **context** (scriptworker.context.Context) – the scriptworker context.

Returns the environment info.

Return type dict

### 2.7.8 scriptworker.cot.verify module

Chain of Trust artifact verification.

`scriptworker.cot.verify.DECISION_TASK_TYPES`
the decision task types.

Type tuple

`scriptworker.cot.verify.PARENT_TASK_TYPES`
the parent task types.

Type tuple

`scriptworker.cot.verify.log`
the log object for this module.

Type logging.Logger

`class scriptworker.cot.verify.AuditLogFormatter (fmt=None, datefmt=None, style='%', validate=True)`

Bases: logging.Formatter

Format the chain of trust log.

`format (record)`
Space debug messages for more legibility.

`class scriptworker.cot.verify.ChainOfTrust (context, name, task_id=None)`

Bases: object

The master Chain of Trust, tracking all the various “LinkOfTrust”s.

**context**
the scriptworker context

Type scriptworker.context.Context

**decision_task_id**
the task_id of self.task’s decision task

Type str

2.7. scriptworker package
parent_task_id
   the task_id of self.task’s parent task
       Type  str

links
   the list of ‘‘LinkOfTrust’’s
       Type  list

name
   the name of the task (e.g., signing)
       Type  str

task_id
   the taskId of the task
       Type  str

task_type
   the task type of the task (e.g., decision, build)
       Type  str

worker_impl
   the taskcluster worker class (e.g., docker-worker) of the task
       Type  str

dependent_task_ids()
   Get all task_id’s for all ‘‘LinkOfTrust tasks.
       Returns  the list of ‘‘task_id’’s
       Return type list

get_all_links_in_chain()
   Return all links in the chain of trust, including the target task.

   By default, we’re checking a task and all its dependencies back to the tree, so the full chain is self.
   links + self. However, we also support checking the decision task itself. In that case, we populate the
   decision task as a link in self.links, and we don’t need to add another check for self.

       Returns  of all ‘‘LinkOfTrust’’s to verify.
       Return type list

get_link(task_id)
   Get a LinkOfTrust by task id.

       Parameters task_id (str) – the task id to find.
       Returns  the link matching the task id.
       Return type LinkOfTrust
       Raises CoTError – if no LinkOfTrust matches.

has_restricted_scopes()
   Determine if this task is requesting any restricted scopes.

       Returns  whether this task requested restricted scopes.
       Return type  bool
is_decision()
Determine if the chain is a decision task.
Returns whether it is a decision task.
Return type bool

is_scope_in_restricted_scopes(scope, restricted_scopes)
Determine if a scope matches in a list of restricted_scopes. If one of the restricted scopes ends with '*', find a partial match.
Returns string of matching restricted_scope, if no match ""
Return type String

is_try_or_pull_request()
Determine if any task in the chain is a try task.
Returns True if a task is a try task.
Return type bool

class scriptworker.cot.verify.LinkOfTrust (context, name, task_id)
Bases: object
Each LinkOfTrust represents a task in the Chain of Trust and its status.
context
the scriptworker context
Type scriptworker.context.Context
decision_task_id
the task_id of self.task’s decision task
Type str
parent_task_id
the task_id of self.task’s parent task
Type str
is_try_or_pull_request
whether the task is a try or a pull request task
Type bool
name
the name of the task (e.g., signing.decision)
Type str
task_id
the taskId of the task
Type str
task_graph
the task graph of the task, if this is a decision task
Type dict
task_type
the task type of the task (e.g., decision, build)
worker_impl
the taskcluster worker class (e.g., docker-worker) of the task

cot
the chain of trust json body.
cot_dir
the local path containing this link’s artifacts.

get_artifact_full_path(path)
str: the full path where an artifact should be located.

is_try_or_pull_request()
bool: the task is either a try or a pull request one.

status = None
task
the task definition.

When set, we also set self.decision_task_id, self.parent_task_id, and self.
worker_impl based on the task definition.
task_graph
the decision task graph, if this is a decision task.

scriptworker.cot.verify.build_link(chain, task_name, task_id)
Build a LinkOfTrust and add it to the chain.

Parameters
• chain (ChainOfTrust) – the chain of trust to add to.
• task_name (str) – the name of the task to operate on.
• task_id (str) – the taskId of the task to operate on.

Raises CoTError – on failure.

scriptworker.cot.verify.build_task_dependencies(chain, task, name, my_task_id)
Recursively build the task dependencies of a task.

Parameters
• chain (ChainOfTrust) – the chain of trust to add to.
• task (dict) – the task definition to operate on.
• name (str) – the name of the task to operate on.
• my_task_id (str) – the taskId of the task to operate on.

Raises CoTError – on failure.

scriptworker.cot.verify.build_taskcluster_yml_url(link)
Build the url to the repo root .taskcluster.yml.
Parameters **link** (LinkOfTrust) – the parent link to get the source url from.

Returns the .taskcluster.yml url

Return type string

scriptworker.cot.verify.check_and_update_action_task_group_id(parent_link, decision_link, rebuilt_definitions)

Update the ACTION_TASK_GROUP_ID of an action after verifying.

Actions have varying ACTION_TASK_GROUP_ID behavior. Release Promotion action tasks set the ACTION_TASK_GROUP_ID to match the action taskId so the large set of release tasks have their own taskgroup. Non-relpro action tasks set the ACTION_TASK_GROUP_ID to match the decision taskId, so tasks are more discoverable inside the original on-push taskgroup.

This poses a json-e task definition problem, hence this function.

This function first checks to make sure the ACTION_TASK_GROUP_ID is a member of (action_task_id, decision_task_id). Then it makes sure the ACTION_TASK_GROUP_ID in the rebuilt_definition is set to the parent_link.task's ACTION_TASK_GROUP_ID so the json-e comparison doesn't fail out.

Ideally, we want to obsolete and remove this function.

Parameters

- **parent_link** (LinkOfTrust) – the parent link to test.
- **decision_link** (LinkOfTrust) – the decision link to test.
- **rebuilt_definitions** (dict) – the rebuilt definitions to check and update.

Raises CoTError – on failure.

scriptworker.cot.verify.check_interactive_docker_worker(link)

Given a task, make sure the task was not defined as interactive.

- task.payload.features.interactive must be absent or False.
- task.payload.env.TASKCLUSTER_INTERACTIVE must be absent or False.

Parameters **link** (LinkOfTrust) – the task link we’re checking.

 Raises CoTError – on interactive.

scriptworker.cot.verify.check_interactive_generic_worker(link)

Given a task, make sure the task was not defined as interactive.

- task.payload.rdpInfo must be absent or False.
- task.payload.scopes must not contain a scope starting with generic-worker:allow-rdp:

Parameters **link** (LinkOfTrust) – the task link we’re checking.

 Raises CoTError – on interactive.

scriptworker.cot.verify.compare_jsone_task_definition(parent_link, rebuilt_definitions)

Compare the json-e rebuilt task definition vs the runtime definition.

Parameters

- **parent_link** (LinkOfTrust) – the parent link to test.
• **rebuilt_definitions (dict)** – the rebuilt task definitions.

    Raises  CoTError – on failure.

```
scriptworker.cot.verify.create_test_workdir (args=None, event_loop=None)
```

Create a test workdir, for manual testing purposes.

Parameters

• **args (list, optional)** – the commandline args to parse. If None, use sys.
  argv[1:]. Defaults to None.

• **event_loop (asyncio.events.AbstractEventLoop)** – the event loop to use. If
  None, use asyncio.get_event_loop(). Defaults to None.

```
scriptworker.cot.verify.download_cot (chain)
```

Download the signed chain of trust artifacts.

Parameters **chain (ChainOfTrust)** – the chain of trust to add to.

    Raises  BaseDownloadError – on failure.

```
scriptworker.cot.verify.download_cot_artifact (chain, task_id, path)
```

Download an artifact and verify its SHA against the chain of trust.

Parameters

• **chain (ChainOfTrust)** – the chain of trust object

• **task_id (str)** – the task ID to download from

• **path (str)** – the relative path to the artifact to download

Returns  the full path of the downloaded artifact

    Return type  str

    Raises  CoTError – on failure.

```
scriptworker.cot.verify.download_cot_artifacts (chain)
```

Call download_cot_artifact in parallel for each “upstreamArtifacts”.

Optional artifacts are allowed to not be downloaded.

Parameters **chain (ChainOfTrust)** – the chain of trust object

Returns  list of full paths to downloaded artifacts. Failed optional artifacts aren’t returned

    Return type  list

    Raises  

    • CoTError – on chain of trust sha validation error, on a mandatory artifact

    • BaseDownloadError – on download error on a mandatory artifact

```
scriptworker.cot.verify.find_sorted_task_dependencies (task, task_name, task_id)
```

Find the taskId’s of the chain of trust dependencies of a given task.

Parameters

• **task (dict)** – the task definition to inspect.

• **task_name (str)** – the name of the task, for logging and naming children.

• **task_id (str)** – the taskId of the task.

Returns  tuples associating dependent task name to dependent task taskId.
Return type  list

scriptworker.cot.verify.get_action_context_and_template(chain, parent_link, decision_link)

Get the appropriate json-e context and template for an action task.

Parameters

- chain (ChainOfTrust) – the chain of trust.
- parent_link (LinkOfTrust) – the parent link to test.
- decision_link (LinkOfTrust) – the parent link’s decision task link.
- tasks_for (str) – the reason the parent link was created (cron, hg-push, action)

Returns  the json-e context and template.

Return type  (dict, dict)

scriptworker.cot.verify.get_all_artifacts_per_task_id(chain, upstream_artifacts)

Return every artifact to download, including the Chain Of Trust Artifacts.

Parameters

- chain (ChainOfTrust) – the chain of trust object
- upstream_artifacts – the list of upstream artifact definitions

Returns  sorted list of paths to downloaded artifacts ordered by taskId

Return type  dict

scriptworker.cot.verify.get_in_tree_template(link)

Get the in-tree json-e template for a given link.

By convention, this template is SOURCE_REPO/.taskcluster.yml.

Parameters  link (LinkOfTrust) – the parent link to get the source url from.

Raises

- CoTError – on non-yaml source_url
- KeyError – on non-well-formed source template

Returns  the first task in the template.

Return type  dict

scriptworker.cot.verify.get_jsone_context_and_template(chain, parent_link, decision_link, tasks_for)

Get the appropriate json-e context and template for any parent task.

Parameters

- chain (ChainOfTrust) – the chain of trust.
- parent_link (LinkOfTrust) – the parent link to test.
- decision_link (LinkOfTrust) – the parent link’s decision task link.
- tasks_for (str) – the reason the parent link was created (cron, hg-push, action)

Returns  the json-e context and template.

Return type  (dict, dict)

scriptworker.cot.verify.get_pushlog_info(decision_link)

Get pushlog info for a decision LinkOfTrust.
Parameters **decision_link** (*LinkOfTrust*) – the decision link to get pushlog info about.

**Returns** pushlog info.

**Return type** dict

`scriptworker.cot.verify.get_scm_level(context, project)`  
Get the scm level for a project from projects.yml.

We define all known projects in projects.yml. Let’s make sure we have it populated in `context`, then return the scm level of `project`.

SCM levels are an integer, 1-3, matching Mozilla commit levels. [https://www.mozilla.org/en-US/about/governance/policies/commit/access-policy/](https://www.mozilla.org/en-US/about/governance/policies/commit/access-policy/)

**Parameters**

- **context** (*scriptworker.context.Context*) – the scriptworker context
- **project** (*str*) – the project to get the scm level for.

**Returns** the level of the project, as a string.

**Return type** str

`scriptworker.cot.verify.get_source_url(obj)`  
Get the source url for a Trust object.

**Parameters** **obj** (*ChainOfTrust or LinkOfTrust*) – the trust object to inspect

**Raises** CoTError – if repo and source are defined and don’t match

**Returns** the source url.

**Return type** str

`scriptworker.cot.verify.get_valid_task_types()`  
Get the valid task types, e.g. signing.

No longer a constant, due to code ordering issues.

**Returns** maps the valid task types (e.g., signing) to their validation functions.

**Return type** immutabledict

`scriptworker.cot.verify.get_valid_worker_impls()`  
Get the valid worker_impls, e.g. docker-worker.

No longer a constant, due to code ordering issues.

**Returns** maps the valid worker_impls (e.g., docker-worker) to their validation functions.

**Return type** immutabledict

`scriptworker.cot.verify.guess_task_type(name, task_defn)`  
Guess the task type of the task.

**Parameters** **name** (*str*) – the name of the task.

**Returns** the task_type.

**Return type** str

**Raises** CoTError – on invalid task_type.

`scriptworker.cot.verify.guess_worker_impl(link)`  
Given a task, determine which worker implementation (e.g., docker-worker) it was run on.
• check for the *worker-implementation* tag
• docker-worker: *task.payload.image* is not None
• check for scopes beginning with the worker type name.
• generic-worker: *task.payload.osGroups* is not None
• generic-worker: *task.payload.mounts* is not None

**Parameters**

- **link** (*LinkOfTrust or ChainOfTrust*) – the link to check.
- **Returns** the worker type.
- **Return type** *str*
- **Raises** CoTError – on inability to determine the worker implementation

**scriptworker.cot.verify.is_artifact_optional**(chain, task_id, path)
Tells whether an artifact is flagged as optional or not.

**Parameters**

- **chain** (*ChainOfTrust*) – the chain of trust object
- **task_id** (*str*) – the id of the aforementioned task

- **Returns** True if artifact is optional
- **Return type** *bool*

**scriptworker.cot.verify.populate_jsone_context**(chain, parent_link, decision_link, tasks_for)
Populate the json-e context to rebuild parent_link’s task definition.
This defines the context that .taskcluster.yml expects to be rendered with. See comments at the top of that file for details.

**Parameters**

- **chain** (*ChainOfTrust*) – the chain of trust to add to.
- **parent_link** (*LinkOfTrust*) – the parent link to test.
- **decision_link** (*LinkOfTrust*) – the parent link’s decision task link.
- **tasks_for** (*str*) – the reason the parent link was created (cron, hg-push, action)

- **Raises** CoTError, KeyError, ValueError – on failure.
- **Returns** the json-e context.
- **Return type** *dict*

**scriptworker.cot.verify.raise_on_errors**(errors, level=50)
Raise a CoTError if errors.

Helper function because I had this code block everywhere.

**Parameters**

- **errors** (*list*) – the error errors
- **level** (*int, optional*) – the log level to use. Defaults to logging.CRITICAL

- **Raises** CoTError – if errors is non-empty
scriptworker.cot.verify.trace_back_to_tree(chain)
Trace the chain back to the tree.

Parameters chain (ChainOfTrust) – the chain we’re operating on

Raises CoTError – on error.

scriptworker.cot.verify.verify_build_task(chain, link)
Verify the build Link.

The main points of concern are tested elsewhere: The task is the same as the task graph task; the command; the
docker-image for docker-worker builds; the revision and repo.

Parameters

• chain (ChainOfTrust) – the chain we’re operating on.
• link (LinkOfTrust) – the task link we’re checking.

scriptworker.cot.verify.verify_chain_of_trust(chain, *, check_task=False)
Build and verify the chain of trust.

Parameters

• chain (ChainOfTrust) – the chain we’re operating on
• check_task (bool) – Whether to download and verify the task itself. This is useful for
  verifying a task after it has run.

Raises CoTError – on failure.

scriptworker.cot.verify.verify_cot_cmdln(args=None, event_loop=None)
Test the chain of trust from the commandline, for debugging purposes.

Parameters

• args (list, optional) – the commandline args to parse. If None, use sys.
  argv[1:]. Defaults to None.
• event_loop (asyncio.events(AbstractEventLoop)) – the event loop to use. If
  None, use asyncio.get_event_loop(). Defaults to None.

scriptworker.cot.verify.verify_cot_signatures(chain)
Verify the signatures of the chain of trust artifacts populated in download_cot.

Parameters chain (ChainOfTrust) – the chain of trust to add to.

Raises CoTError – on failure.

scriptworker.cot.verify.verify_docker_image_sha(chain, link)
Verify that built docker shas match the artifact.

Parameters

• chain (ChainOfTrust) – the chain we’re operating on.
• link (LinkOfTrust) – the task link we’re checking.

Raises CoTError – on failure.

scriptworker.cot.verify.verify_docker_image_task(chain, link)
Verify the docker image Link.
Parameters

- `chain` (ChainOfTrust) – the chain we’re operating on.
- `link` (LinkOfTrust) – the task link we’re checking.

scriptworker.cot.verify.verify_docker_worker_task (chain, link)

Docker-worker specific checks.

Parameters

- `chain` (ChainOfTrust) – the chain we’re operating on
- `link` (ChainOfTrust or LinkOfTrust) – the trust object for the signing task.

Raises CoTError – on failure.

scriptworker.cot.verify.verify_generic_worker_task (chain, link)

generic-worker specific checks.

Parameters

- `chain` (ChainOfTrust) – the chain we’re operating on
- `link` (ChainOfTrust or LinkOfTrust) – the trust object for the signing task.

Raises CoTError – on failure.

scriptworker.cot.verify.verify_link_ed25519_cot_signature (chain, link, unsigned_path, signature_path)

Verify the ed25519 signatures of the chain of trust artifacts populated in download_cot.

Parameters

- `chain` (ChainOfTrust) – the chain of trust to add to.

Raises (CoTError, ScriptWorkerEd25519Error) – on signature verification failure.

scriptworker.cot.verify.verify_link_in_task_graph (chain, decision_link, task_link)

Compare the runtime task definition against the decision task graph.

Parameters

- `chain` (ChainOfTrust) – the chain we’re operating on.
- `decision_link` (LinkOfTrust) – the decision task link
- `task_link` (LinkOfTrust) – the task link we’re testing

Raises CoTError – on failure.

scriptworker.cot.verify.verify_parent_task (chain, link)

Verify the parent task Link.

Action task verification is currently in the same verification function as decision tasks, because sometimes we’ll have an action task masquerading as a decision task, e.g. in templatized actions for release graphs. To make sure our guess of decision or action task isn’t fatal, we call this function; this function uses is_action() to determine how to verify the task.

Parameters

- `chain` (ChainOfTrust) – the chain we’re operating on.
- `link` (LinkOfTrust) – the task link we’re checking.

Raises CoTError – on chain of trust verification error.
Scriptworker Documentation, Release 42.2.0

scriptworker.cot.verify.verify_parent_task_definition(chain, parent_link)
Rebuild the decision/action/cron task definition via json-e.

This is Chain of Trust verification version 2, aka cotv2. Instead of looking at various parts of the parent task’s task definition and making sure they look well-formed, let’s rebuild the task definition from the tree and make sure it matches.

We use the the link with the task_id of parent_link.decision_task_id for a number of the checks here. If parent_link is a decision or cron task, they’re the same task. If parent_link is an action task, this will reference the decision task that the action task is based off of.

Parameters
• chain(ChainOfTrust) – the chain of trust to add to.
• parent_link(LinkOfTrust) – the parent link to test.

Raises CoTError – on failure.

scriptworker.cot.verify.verify_partials_task(chain, obj)
Verify the partials trust object.

The main points of concern are tested elsewhere: Runs as a docker-worker.

Parameters
• chain(ChainOfTrust) – the chain we’re operating on
• obj(ChainOfTrust or LinkOfTrust) – the trust object for the balrog task.

Raises CoTError – on error.

scriptworker.cot.verify.verify_repo_matches_url(repo, url)
Verify url is a part of repo.

We were using startswith() for a while, which isn’t a good comparison. This function allows us to urlparse and compare host and path.

Parameters
• repo(str) – the repo url
• url(str) – the url to verify is part of the repo

Returns True if the repo matches the url.

Return type bool

scriptworker.cot.verify.verify_scriptworker_task(chain, obj)
Verify the signing trust object.

Currently the only check is to make sure it was run on a scriptworker.

Parameters
• chain(ChainOfTrust) – the chain we’re operating on
• obj(ChainOfTrust or LinkOfTrust) – the trust object for the signing task.

scriptworker.cot.verify.verify_task_in_task_graph(task_link, graph_defn, level=50)
Verify a given task_link’s task against a given graph task definition.

This is a helper function for verify_link_in_task_graph; this is split out so we can call it multiple times when we fuzzy match.

Parameters
• task_link(LinkOfTrust) – the link to try to match
• **graph_defn** *(dict)* – the task definition from the task-graph.json to match task_link against

• **level** *(int, optional)* – the logging level to use on errors. Defaults to logging.CRITICAL

Raises CoTError – on failure

**scriptworker.cot.verify.verify_task_types** *(chain)*

Verify the task type (e.g. decision, build) of each link in the chain.

Parameters **chain** *(ChainOfTrust)* – the chain we’re operating on

Returns mapping task type to the number of links.

Return type *dict*

**scriptworker.cot.verify.verify_worker_impls** *(chain)*

Verify the task type (e.g. decision, build) of each link in the chain.

Parameters **chain** *(ChainOfTrust)* – the chain we’re operating on

Raises CoTError – on failure

### 2.7.9 scriptworker.ed25519 module

ed25519 support for scriptworker.

**scriptworker.ed25519.log**

the log object for the module

Type *logging.Logger*

**scriptworker.ed25519.ed25519_private_key_from_file** *(path)*

Create an ed25519 key from the contents of path.

path is a filepath containing a base64-encoded ed25519 key seed.

Parameters

• **fn** *(callable)* – the function to call with the contents from path

• **path** *(str)* – the file path to the base64-encoded key seed.

Returns the appropriate key type from path

Return type *obj*

Raises *ScriptWorkerEd25519Error*

**scriptworker.ed25519.ed25519_private_key_from_string** *(string)*

Create an ed25519 private key from string, which is a seed.

Parameters **string** *(str)* – the string to use as a seed.

Returns the private key

Return type *Ed25519PrivateKey*

**scriptworker.ed25519.ed25519_private_key_to_string** *(key)*

Convert an ed25519 private key to a base64-encoded string.

Parameters **key** *(Ed25519PrivateKey)* – the key to write to the file.

Returns the key representation as a str

Return type *str*
scriptworker.ed25519.ed25519_public_key_from_file(path)
Create an ed25519 key from the contents of path.

path is a filepath containing a base64-encoded ed25519 key seed.

Parameters

• fn (callable) – the function to call with the contents from path
• path (str) – the file path to the base64-encoded key seed.

Returns the appropriate key type from path

Return type obj

Raises ScriptWorkerEd25519Error

scriptworker.ed25519.ed25519_public_key_from_string(string)
Create an ed25519 public key from string, which is a seed.

Parameters string (str) – the string to use as a seed.

Returns the public key

Return type Ed25519PublicKey

scriptworker.ed25519.ed25519_public_key_to_string(key)
Convert an ed25519 public key to a base64-encoded string.

Parameters key (Ed25519PublicKey) – the key to write to the file.

Returns the key representation as a str

Return type str

scriptworker.ed25519.verify_ed25519_signature(public_key, contents, signature, message)
Verify that signature comes from public_key and contents.

Parameters

• public_key (Ed25519PublicKey) – the key to verify the signature
• contents (bytes) – the contents that was signed
• signature (bytes) – the signature to verify
• message (str) – the error message to raise.

Raises ScriptWorkerEd25519Error – on failure

scriptworker.ed25519.verify_ed25519_signature_cmdln(args=None, exception=<class 'SystemExit'>)
Verify an ed25519 signature from the command line.

Parameters

• args (list, optional) – the commandline args to parse. If None, use sys.argv[1:]. Defaults to None.
• exception (Exception, optional) – the exception to raise on failure. Defaults to SystemExit.

2.7.10 scriptworker.github module

GitHub helper functions.
class scriptworker.github.GitHubRepository(owner, repo_name, token="")
    Bases: object

    Wrapper around GitHub API. Used to access public data.

    **definition**
    Fetch the definition of the repository, exposed by the GitHub API.

    **Returns** a representation of the repo definition

    **Return type** dict

    **get_commit**(commit_hash)
    Fetch the definition of the commit, exposed by the GitHub API.

    **Parameters** commit_hash (str) – the hash of the git commit

    **Returns** a representation of the commit

    **Return type** dict

    **get_pull_request**(pull_request_number)
    Fetch the definition of the pull request, exposed by the GitHub API.

    **Parameters** pull_request_number (int) – the ID of the pull request

    **Returns** a representation of the pull request

    **Return type** dict

    **get_release**(tag_name)
    Fetch the definition of the release matching the tag name.

    **Parameters** tag_name (str) – the tag linked to the release

    **Returns** a representation of the tag

    **Return type** dict

    **get_tag_hash**(tag_name)
    Fetch the commit hash that was tagged with tag_name.

    **Parameters** tag_name (str) – the name of the tag

    **Returns** the commit hash linked by the tag

    **Return type** str

    **has_commit_landed_on_repository**(context, revision)
    Tell if a commit was landed on the repository or if it just comes from a pull request.

    **Parameters**

    • context (scriptworker.context.Context) – the scriptworker context.

    • revision (str) – the commit hash or the tag name.

    **Returns** True if the commit is present in one of the branches of the main repository

    **Return type** bool

scriptworker.github.extract_github_repo_and_revision_from_source_url(url)
    Given an URL, return the repo name and who owns it.

    **Parameters** url (str) – The URL to the GitHub repository

    **Raises** ValueError – on url that aren’t from github or when the revision cannot be extracted

    **Returns** the owner of the repository, the repository name
Return type  str, str

scriptworker.github.extract_github_repo_full_name(url)
Given an URL, return the full name of it.
The full name is RepoOwner/RepoName.

Parameters url (str) – The URL to the GitHub repository

Raises ValueError – on url that aren’t from github

Returns the full name.

Return type  str

scriptworker.github.extract_github_repo_owner_and_name(url)
Given an URL, return the repo name and who owns it.

Parameters url (str) – The URL to the GitHub repository

Raises ValueError – on url that aren’t from github

Returns the owner of the repository, the repository name

Return type  str, str

scriptworker.github.extract_github_repo_ssh_url(url)
Given an URL, return the ssh url.

Parameters url (str) – The URL to the GitHub repository

Raises ValueError – on url that aren’t from github

Returns the ssh url

Return type  str

scriptworker.github.is_github_repo_owner_the_official_one(context, repo_owner)
Given a repo_owner, check if it matches the one configured to be the official one.

Parameters

• context (scriptworker.context.Context) – the scriptworker context.

• repo_owner (str) – the repo_owner to verify

Raises scriptworker.exceptions.ConfigError – when no official owner was defined

Returns True when repo_owner matches the one configured to be the official one

Return type  bool

scriptworker.github.is_github_url(url)
Tell if a given URL matches a Github one.

Parameters url (str) – The URL to test. It can be None.

Returns False if the URL is not a string or if it doesn’t match a Github URL

Return type  bool

2.7.11 scriptworker.exceptions module

scriptworker exceptions.
exception scriptworker.exceptions.BaseDownloadError(msg: str)
Bases: scriptworker.exceptions.ScriptWorkerTaskException

Base class for DownloadError and Download404.

exit_code
this is set to 4 (resource-unavailable).

Type int

exception scriptworker.exceptions.CoTError(msg: str)
Bases: scriptworker.exceptions.ScriptWorkerTaskException, KeyError

Failure in Chain of Trust verification.

exit_code
this is set to 3 (malformed-payload).

Type int

exception scriptworker.exceptions.ConfigError
Bases: scriptworker.exceptions.ScriptWorkerException

Invalid configuration provided to scriptworker.

exit_code
this is set to 5 (internal-error).

Type int

exception scriptworker.exceptions.Download404(msg: str)
Bases: scriptworker.exceptions.BaseDownloadError

404 in scriptworker.utils.download_file.

exit_code
this is set to 4 (resource-unavailable).

Type int

exception scriptworker.exceptions.DownloadError(msg: str)
Bases: scriptworker.exceptions.BaseDownloadError

Failure in scriptworker.utils.download_file.

exit_code
this is set to 4 (resource-unavailable).

Type int

exception scriptworker.exceptions.ScriptWorkerEd25519Error(msg: str)
Bases: scriptworker.exceptions.CoTError

Scriptworker ed25519 error.

exit_code
this is set to 5 (internal-error).

Type int

exception scriptworker.exceptions.ScriptWorkerException
Bases: Exception

The base exception in scriptworker.

When raised inside of the run_loop loop, set the taskcluster task status to at least self.exit_code.
exit_code
this is set to 5 (internal-error).
    Type  int
    exit_code = 5

exception  scriptworker.exceptions.ScriptWorkerRetryException
    Bases: scriptworker.exceptions.ScriptWorkerException
    Scriptworker retry error.
    exit_code
    this is set to 4 (resource-unavailable)
    Type  int
    exit_code = 4

exception  scriptworker.exceptions.ScriptWorkerTaskException(*args, exit_code: int = 1)
    Bases: scriptworker.exceptions.ScriptWorkerException
    Scriptworker task error.
    To use:

```python
import sys
try:
    ...
except ScriptWorkerTaskException as exc:
    log.exception("log message")
    sys.exit(exc.exit_code)
```

exit_code
this is 1 by default (failure)
    Type  int

exception  scriptworker.exceptions.TaskVerificationError(msg: str)
    Bases: scriptworker.exceptions.ScriptWorkerTaskException
    Verification error on a Taskcluster task.
    Use it when your script fails to validate any input from the task definition

exception  scriptworker.exceptions.WorkerShutdownDuringTask
    Bases: BaseException
    Task cancelled because worker is shutting down.

2.7.12 scriptworker.log module

Scriptworker logging.

scriptworker.log.log
the log object for this module.
    Type  logging.Logger

Add a short-lived log with a contextmanager for cleanup.

Parameters

- **context** (scriptworker.context.Context) – the scriptworker context
- **path** (str) – the path to the log file to create
- **log_obj** (logging.Logger) – the log object to modify. If None, use scriptworker.log.log. Defaults to None.
- **level** (int, optional) – the logging level. Defaults to logging.DEBUG.
- **formatter** (logging.Formatter, optional) – the logging formatter. If None, defaults to logging.Formatter(fmt=fmt). Default is None.

Yields None – but cleans up the handler afterwards.

scriptworker.log.get_log_filehandle(context: Any) → Iterator[IO[str]]

Open the log and error filehandles.

Parameters **context** (scriptworker.context.Context) – the scriptworker context.

Yields log filehandle

scriptworker.log.get_log_filename(context: Any) → str

Get the task log/error file paths.

Parameters **context** (scriptworker.context.Context) – the scriptworker context.

Returns log file path

Return type string

scriptworker.log.pipe_to_log(pipe: asyncio.streams.StreamReader, filehandles: Sequence[IO[str]] = (), level: int = 20) → None

Log from a subprocess PIPE.

Parameters

- **pipe** (filehandle) – subprocess process STDOUT or STDERR
- **filehandles** (list of filehandles, optional) – the filehandle(s) to write to. If empty, don’t write to a separate file. Defaults to ()
- **level** (int, optional) – the level to log to. Defaults to logging.INFO.

scriptworker.log.update_logging_config(context: Any, log_name: Optional[str] = None, file_name: str = 'worker.log') → None

Update python logging settings from config.

By default, this sets the scriptworker log settings, but this will change if some other package calls this function or specifies the log_name.

- Use formatting from config settings.
- Log to screen if verbose
- Add a rotating logfile from config settings.

Parameters

- **context** (scriptworker.context.Context) – the scriptworker context.
• **log_name** *(str, optional)* – the name of the Logger to modify. If None, use the top level module (‘scriptworker’). Defaults to None.

### 2.7.13 scriptworker.task module

Scriptworker task execution.

- **scriptworker.task.REPO_SCOPE_REGEX**
  - the regex for the repo_scope of a task
    - **Type** regex

- **scriptworker.task.log**
  - the log object for the module
    - **Type** logging.Logger

- **scriptworker.task.claim_work**(context)**
  - Find and claim the next pending task in the queue, if any.
    - **Parameters** context *(scriptworker.context.Context)* – the scriptworker context.
    - **Returns** a dict containing a list of the task definitions of the tasks claimed.
    - **Return type** dict

- **scriptworker.task.complete_task**(context, result)**
  - Mark the task as completed in the queue.
    - Decide whether to call reportCompleted, reportFailed, or reportException based on the exit status of the script.
    - If the task has expired or been cancelled, we’ll get a 409 status.
    - **Parameters** context *(scriptworker.context.Context)* – the scriptworker context.
    - **Raises** taskcluster.exceptions.TaskclusterRestFailure – on non-409 error.

- **scriptworker.task.get_action_callback_name**(task)**
  - Get the callback name of an action task.
    - **Parameters** obj *(ChainOfTrust or LinkOfTrust)* – the trust object to inspect
    - **Returns** the name. None: if not found.
    - **Return type** str

- **scriptworker.task.get_and_check_tasks_for**(context, task, msg_prefix=“”)**
  - Given a parent task, return the reason the parent task was spawned.
    - .taskcluster.yml uses this to know whether to spawn an action, cron, or decision task definition.
    - tasks_for must be a valid one defined in the context.
    - **Parameters**
      - **task** *(dict)* – the task definition.
      - **msg_prefix** *(str)* – the string prefix to use for an exception.
    - **Raises** (TypeError, ValueError) – on failure to find a valid tasks_for.
    - **Returns** the tasks_for
    - **Return type** str

- **scriptworker.task.get_branch**(task, source_env_prefix)**
  - Get the branch on top of which the graph was made.
Parameters

- **obj** *(ChainOfTrust or LinkOfTrust)* – the trust object to inspect
- **source_env_prefix** *(str)* – The environment variable prefix that is used to get repository information.

Returns the username of the entity who triggered the graph. None: if not defined for this task.

Return type  *str*

```
scriptworker.task.get_commit_message(task)
```

Get the commit message for a task.

Parameters **obj** *(ChainOfTrust or LinkOfTrust)* – the trust object to inspect

Returns the commit message.

Return type  *str*

```
scriptworker.task.get_decision_task_id(task)
```

Given a task dict, return the decision taskId.

By convention, the decision task of the taskId is the task’s taskGroupId.

Parameters **task** *(dict)* – the task dict.

Returns the taskId of the decision task.

Return type  *str*

```
scriptworker.task.get_parent_task_id(task)
```

Given a task dict, return the parent taskId.

The parent taskId could be a decision taskId, or an action taskId. The parent is the task that created this task; it should have a task-graph.json containing this task’s definition as an artifact.

Parameters **task** *(dict)* – the task dict.

Returns the taskId of the parent.

Return type  *str*

```
scriptworker.task.get_project(context, source_url)
```

Given a source_url, return the project.

The project is in the path, but is the repo name. releases/mozilla-beta is the path; mozilla-beta is the project.

Parameters **source_url** *(str)* – the source url to find the project for.

Raises  *RuntimeError* – on failure to find the project.

Returns the project.

Return type  *str*

```
scriptworker.task.get_provisioner_id(task)
```

Given a task dict, return the provisionerId.

Parameters **task** *(dict)* – the task dict.

Returns the provisionerId.

Return type  *str*

```
scriptworker.task.get_pull_request_number(task, source_env_prefix)
```

Get what Github pull request created the graph.

Parameters
• `obj`(*ChainOfTrust* or *LinkOfTrust*) – the trust object to inspect

• `source_env_prefix`(*str*) – The environment variable prefix that is used to get repository information.

**Returns**  the pull request number. None: if not defined for this task.

**Return type**  int

`scriptworker.task.get_push_date_time(task, source_env_prefix)`

Get when a Github commit was pushed.

We usually need to extract this piece of data from the task itself because Github doesn’t expose reliable push data in the 3rd version of their API. This may happen in their future v4 API: https://developer.github.com/v4/object/push/.

**Parameters**

• `obj`(*ChainOfTrust* or *LinkOfTrust*) – the trust object to inspect

• `source_env_prefix`(*str*) – The environment variable prefix that is used to get repository information.

**Returns**  the string when the event was pushed. It’s usually formatted as ISO 8601. However, it may be an epoch timestamp, (known case: github-push events).

None: if not defined for this task.

**Return type**  str

`scriptworker.task.get_repo(task, source_env_prefix)`

Get the repo for a task.

**Parameters**

• `task`(*ChainOfTrust* or *LinkOfTrust*) – the trust object to inspect

• `source_env_prefix`(*str*) – The environment variable prefix that is used to get repository information.

**Returns**  the source url. None: if not defined for this task.

**Return type**  str

`scriptworker.task.get_repo_scope(task, name)`

Given a parent task, return the repo scope for the task.

Background in https://bugzilla.mozilla.org/show_bug.cgi?id=1459705#c3

**Parameters** `task`(*dict*) – the task definition.

**Raises**  `ValueError` – on too many ‘repo_scope’s (we allow for 1 or 0).

**Returns**  the repo_scope None: if no repo_scope is found

**Return type**  str

`scriptworker.task.get_revision(task, source_env_prefix)`

Get the revision for a task.

**Parameters**

• `obj`(*ChainOfTrust* or *LinkOfTrust*) – the trust object to inspect

• `source_env_prefix`(*str*) – The environment variable prefix that is used to get repository information.
Returns the revision. None: if not defined for this task.

Return type str

scriptworker.task.get_run_id(claim_task)
Given a claim_task json dict, return the runId.

Parameters claim_task (dict) – the claim_task dict.

Returns the runId.

Return type int

scriptworker.task.get_task_definition(queue, task_id, exception=<class 'taskcluster.exceptions.TaskclusterFailure'>)
Get the task definition from the queue.

Detect whether the task definition is empty, per bug 1618731.

Parameters

• queue (taskcluster.aio.Queue) – the taskcluster Queue object
• task_id (str) – the taskId of the task
• exception (Exception, optional) – the exception to raise if unsuccessful. Defaults to TaskclusterFailure.

scriptworker.task.get_task_id(claim_task)
Given a claim_task json dict, return the taskId.

Parameters claim_task (dict) – the claim_task dict.

Returns the taskId.

Return type str

scriptworker.task.get_triggered_by(task, source_env_prefix)
Get who triggered the graph.

Parameters

• obj (ChainOfTrust or LinkOfTrust) – the trust object to inspect
• source_env_prefix (str) – The environment variable prefix that is used to get repository information.

Returns the username of the entity who triggered the graph. None: if not defined for this task.

Return type str

scriptworker.task.get_worker_pool_id(task)
Given a task dict, return the worker pool id.

This corresponds to {provisioner_id}/{workerType}.

Parameters task (dict) – the task dict.

Returns the workerPoolId.

Return type str

scriptworker.task.get_worker_type(task)
Given a task dict, return the workerType.

Parameters task (dict) – the task dict.

Returns the workerType.
Return type str

scriptworker.task.is_action(task)
Determine if a task is an action task.

Trusted decision and action tasks are important in that they can generate other valid tasks. The verification of decision and action tasks is slightly different, so we need to be able to tell them apart.

This checks for the following things:

* ``task.payload.env.ACTION_CALLBACK`` exists
* ``task.extra.action`` exists

Parameters task (dict) – the task definition to check

Returns True if it's an action

Return type bool

scriptworker.task.is_github_task(task)
Determine if a task is related to GitHub.

This function currently looks into the schedulerId, extra.tasks_for, and metadata.source.

Parameters task (dict) – the task definition to check.

Returns True if a piece of data refers to GitHub

Return type bool

scriptworker.task.is_pull_request(context, task)
Determine if a task is a pull-request-like task (restricted privs).

This goes further than checking tasks_for. We may or may not want to keep this.

This checks for the following things:

* ``task.extra.env.tasks_for`` == "github-pull-request"
* ``task.payload.env.MOBILE_HEAD_REPOSITORY`` doesn't come from an official repo
* ``task.metadata.source`` doesn't come from an official repo, either
* The last 2 items are landed on the official repo

Parameters

• context (scriptworker.context.Context) – the scriptworker context.
• task (dict) – the task definition to check.

Returns True if it's a pull-request. False if it either comes from the official repos or if the origin can’t be determined. In fact, valid scriptworker tasks don’t expose task.extra.env.tasks_for or task.payload.env.MOBILE_HEAD_REPOSITORY, for instance.

Return type bool

scriptworker.task.is_try(task, source_env_prefix)
Determine if a task is a 'try' task (restricted privs).

This goes further than get_repo. We may or may not want to keep this.

This checks for the following things:
* `task.payload.env.GECKO_HEAD_REPOSITORY` == "https://hg.mozilla.org/try/
* `task.payload.env.MH_BRANCH` == "try"
* `task.metadata.source` == "https://hg.mozilla.org/try/...
* `task.schedulerId` in ("gecko-level-1", )

Parameters

- **task** *(dict)* – the task definition to check
- **source_env_prefix** *(str)* – The environment variable prefix that is used to get repository information.

Returns True if it’s try

Return type *bool*

scriptworker.task.is_try_or_pull_request(*context, task*)

Determine if a task is a try or a pull-request-like task (restricted privs).

Checks are the ones done in `is_try` and `is_pull_request`

Parameters

- **context** *(scriptworker.context.Context)* – the scriptworker context.
- **task** *(dict)* – the task definition to check.

Returns True if it’s a pull-request or a try task

Return type *bool*

scriptworker.task.prepare_to_run_task(*context, claim_task*)

Given a claim_task json dict, prepare the context and work_dir.

Set `context.claim_task` and write a `work_dir/current_task_info.json`

Parameters

- **context** *(scriptworker.context.Context)* – the scriptworker context.
- **claim_task** *(dict)* – the claim_task dict.

Returns the contents of `current_task_info.json`

Return type *dict*

scriptworker.task.reclaim_task(*context, task*)

Try to reclaim a task from the queue.

This is a keepalive / heartbeat. Without it the job will expire and potentially be re-queued. Since this is run async from the task, the task may complete before we run, in which case we’ll get a 409 the next time we reclaim.

Parameters **context** *(scriptworker.context.Context)* – the scriptworker context

Raises taskcluster.exceptions.TaskclusterRestFailure – on non-409 status_code from taskcluster.aio.Queue.reclaimTask()

scriptworker.task.retry_get_task_definition(*queue, task_id, exception=<class 'taskcluster.exceptions.TaskclusterFailure'>, **kwargs*)

Retry get_task_definition.

Parameters

- **queue** *(taskcluster.aio.Queue)* – the taskcluster Queue object
• **task_id**(str) – the taskId of the task

• **exception** (*Exception, optional*) – the exception to raise if unsuccessful. Defaults to TaskclusterFailure.

**scriptworker.task.run_task**(context, to_cancellable_process)
Run the task, sending stdout+stderr to files.

https://github.com/python/asyncio/blob/master/examples/subprocess_shell.py

**Parameters**

• **context** (*scriptworker.context.Context*) – the scriptworker context.

• **to_cancellable_process** (*types.Callable*) – tracks the process so that it can be stopped if the worker is shut down

**Returns** 1 on failure, 0 on success

**Return type** int

**scriptworker.task.worst_level**(level1, level2)
Given two int levels, return the larger.

**Parameters**

• **level1** (*int*) – exit code 1.

• **level2** (*int*) – exit code 2.

**Returns** the larger of the two levels.

**Return type** int

### 2.7.14 scriptworker.utils module

Generic utils for scriptworker.

**scriptworker.utils.log**
the log object for the module

**Type** logging.Logger

**scriptworker.utils.add_enumerable_item_to_dict**(dict, key, item)
Add an item to a list contained in a dict.

For example: If the dict is `{'some_key': ['an_item']}`, then calling this function will alter the dict to `{'some_key': ['an_item', 'another_item']}`.

If the key doesn’t exist yet, the function initializes it with a list containing the item.

List-like items are allowed. In this case, the existing list will be extended.

**Parameters**

• **dict** (*dict*) – the dict to modify

• **key** (*str*) – the key to add the item to

• **item** (*whatever*) – The item to add to the list associated to the key

**scriptworker.utils.add_projectid**(task_def)
Add a projectId property to a task, if none already exists, using the Taskcluster default value of ‘none’.

**Parameters** **task_def** (*dict*) – the task definition

**Returns** the task definition, with projectId
Return type  task_def (dict)

**scriptworker.utils.add_taskqueueid(task_def)**
Add a taskQueueId property to a task, if none already exists, based on the provisionerId and workerType. Then remove those two properties.

**Parameters**
- **task_def (dict)** – the task definition

**Returns**
the task definition, with taskQueueId

**Return type**  task_def (dict)

**scriptworker.utils.calculate_sleep_time(attempt, delay_factor=5.0, randomization_factor=0.5, max_delay=120)**
Calculate the sleep time between retries, in seconds.

Based off of taskcluster.utils.calculateSleepTime, but with kwargs instead of constant delay_factor/randomization_factor/max_delay. The taskcluster function generally slept for less than a second, which didn’t always get past server issues.

**Parameters**
- **attempt (int)** – the retry attempt number
- **delay_factor (float, optional)** – a multiplier for the delay time. Defaults to 5.
- **randomization_factor (float, optional)** – a randomization multiplier for the delay time. Defaults to .5.
- **max_delay (float, optional)** – the max delay to sleep. Defaults to 120 (seconds).

**Returns**
the time to sleep, in seconds.

**Return type**  float

**scriptworker.utils.cleanup(context)**
Clean up the work_dir and artifact_dir between task runs, then recreate.

**Parameters**
- **context (scriptworker.context.Context)** – the scriptworker context.

**scriptworker.utils.create_temp_creds(client_id, access_token, start=None, expires=None, scopes=None, name=None)**
Request temp TC creds with our permanent creds.

**Parameters**
- **client_id (str)** – the taskcluster client_id to use
- **access_token (str)** – the taskcluster access_token to use
- **start (str, optional)** – the datetime string when the credentials will start to be valid. Defaults to 10 minutes ago, for clock skew.
- **expires (str, optional)** – the datetime string when the credentials will expire. Defaults to 31 days after 10 minutes ago.
- **scopes (list, optional)** – The list of scopes to request for the temp creds. Defaults to ['assume:project:taskcluster:worker-test-scopes',]
- **name (str, optional)** – the name to associate with the creds.

**Returns**
the temporary taskcluster credentials.

**Return type**  dict

**scriptworker.utils.datestring_to_timestamp(datestring)**
Create a timetamp from a taskcluster datestring.
Parameters **datestring** *(str)* – the datestring to convert. isoformat, like “2016-04-16T03:46:24.958Z”

**Returns** the corresponding timestamp.

**Return type** int

scriptworker.utils.download_file*(context, url, abs_filename, session=None, chunk_size=128, auth=None)*

Download a file, async.

**Parameters**

- **context** *(scriptworker.context.Context)* – the scriptworker context.
- **url** *(str)* – the url to download
- **abs_filename** *(str)* – the path to download to
- **session** *(aiohttp.ClientSession, optional)* – the session to use. If None, use context.session. Defaults to None.
- **chunk_size** *(int, optional)* – the chunk size to read from the response at a time. Default is 128.

scriptworker.utils.filepaths_in_dir*(path)*

Find all files in a directory, and return the relative paths to those files.

**Parameters** **path** *(str)* – the directory path to walk

**Returns**

the list of relative paths to all files inside of path or its subdirectories.

**Return type** list

scriptworker.utils.format_json*(data)*

Format json as a sorted string (indents of 2).

**Parameters** **data** *(dict)* – the json to format.

**Returns** the formatted json.

**Return type** str

scriptworker.utils.get_hash*(path, hash_alg='sha256')*

Get the hash of the file at path.

I’d love to make this async, but evidently file i/o is always ready

**Parameters**

- **path** *(str)* – the path to the file to hash.
- **hash_alg** *(str, optional)* – the algorithm to use. Defaults to ‘sha256’.

**Returns** the hexdigest of the hash.

**Return type** str

scriptworker.utils.get_loggable_url*(url)*

Strip out secrets from taskcluster urls.

**Parameters** **url** *(str)* – the url to strip

**Returns** the loggable url

**Return type** str
scriptworker.utils.get_parts_of_url_path(url)
Given a url, take out the path part and split it by ‘/’.

Parameters  url (str) – the url slice

returns  list: parts after the domain name of the URL

scriptworker.utils.get_results_and_future_exceptions(tasks)
Given a list of futures, await them, then return results and exceptions.

This is similar to raise_future_exceptions, except that it doesn’t raise any exception. They are returned instead. This allows some tasks to optionally fail. Please consider that no exception will be raised when calling this function. You must verify the content of the second item of the tuple. It contains all exceptions raised by the futures.

Parameters  tasks (list) – the list of futures to await and check for exceptions.

Returns  the list of results from the futures, then the list of exceptions.

Return type  tuple

scriptworker.utils.get_single_item_from_sequence(sequence, condition, ErrorClass=<class 'ValueError'>, no_item_error_message='No item matched condition', too_many_item_error_message='Too many items matched condition', append_sequence_to_error_message=True)

Return an item from a python sequence based on the given condition.

Parameters

• sequence (sequence) – The sequence to filter
• condition – A function that serves to filter items from sequence. Function must have one argument (a single item from the sequence) and return a boolean.
• ErrorClass (Exception) – The error type raised in case the item isn’t unique
• no_item_error_message (str) – The message raised when no item matched the condition
• too_many_item_error_message (str) – The message raised when more than one item matched the condition
• append_sequence_to_error_message (bool) – Show or hide what was the tested sequence in the error message. Hiding it may prevent sensitive data (such as password) to be exposed to public logs

Returns  The only item in the sequence which matched the condition

scriptworker.utils.load_json_or_yaml(string: str, is_path: Optional[bool] = False, file_type: Optional[str] = 'json', exception: Optional[Type[BaseException]] = <class 'scriptworker.exceptions.ScriptWorkerTaskException'>, message: str = 'Failed to load %(file_type)s: %(exc)s') → Optional[Dict[str, Any]]

Load json or yaml from a filehandle or string, and raise a custom exception on failure.

Parameters

• string (str) – json/yaml body or a path to open
• **is_path**(bool, *optional*) – if string is a path. Defaults to False.
• **file_type**(str, *optional*) – either “json” or “yaml”. Defaults to “json”.
• **exception**(exception, *optional*) – the exception to raise on failure. If None, don’t raise an exception. Defaults to ScriptWorkerTaskException.
• **message**(str, *optional*) – the message to use for the exception. Defaults to “Failed to load %(file_type)s: %(exc)s”

**Returns** the data from the string.

**Return type** `dict`

**Raises** `Exception` – as specified, on failure


Retry a json/yaml file download, load it, then return its data.

**Parameters**
• **context**(scriptworker.context.Context) – the scriptworker context.
• **url**(str) – the url to download
• **path**(str) – the path to download to
• **overwrite**(bool, *optional*) – if False and path exists, don’t download. Defaults to True.

**Returns** the url data.

**Return type** `dict`

**Raises** `Exception` – as specified, on failure

`scriptworker.utils.makedirs(path: str) → None`

Equivalent to `mkdir -p`.

**Parameters** **path**(str) – the path to `mkdir -p`

**Raises** `ScriptWorkerException` – if path exists already and the realpath is not a dir.

`scriptworker.utils.match_url_path_callback(match: Match[str]) → str`

Return the path, as a `match_url_regex` callback.

**Parameters** **match**(re.match) – the regex match object from `match_url_regex`

**Returns** the path matched in the regex.

**Return type** `string`

`scriptworker.utils.match_url_regex(rules: Tuple[Any], url: str, callback: Callable[[Match[str]], Any]) → Any`

Given rules and a callback, find the rule that matches the url.

**Rules look like:**

```python
{
    'schemes': ['https', 'ssh'],
    'netlocs': ['hg.mozilla.org'],
    'path_regexes': [
        "^(?P<path>/mozilla-(central|unified))(/|$)",
    ],
}
```

(continues on next page)
Parameters

- **rules** *(list)* – a list of dictionaries specifying lists of schemes, netlocs, and path_regexes.
- **url** *(str)* – the url to test
- **callback** *(function)* – a callback that takes a `re.MatchObject`. If it returns `None`, continue searching. Otherwise, return the value from the callback.

**Returns** the value from the callback, or `None` if no match.

**Return type** `value`

`scriptworker.utils.raise_future_exceptions(tasks)`

Given a list of futures, await them, then raise their exceptions if any.

Without something like this, a bare:

```python
await asyncio.wait(tasks)
```

will swallow exceptions.

**Parameters**

- **tasks** *(list)* – the list of futures to await and check for exceptions.

**Returns** the list of results from the futures.

**Return type** `list`

**Raises** `Exception` – any exceptions in `task.exception()`

`scriptworker.utils.read_from_file(path, file_type='text', exception=<class 'scriptworker.exceptions.ScriptWorkerException'>)`

Read from `path`.

Small helper function to read from `file`.

**Parameters**

- **path** *(str)* – the path to read from.
- **file_type** *(str, optional)* – the type of file. Currently accepts `text` or `binary`. Defaults to `text`.
- **exception** *(Exception, optional)* – the exception to raise if unable to read from the file. Defaults to `ScriptWorkerException`

**Returns** if unable to read from `path` and `exception` is `None` `str` or `bytes`: the contents of `path`

**Return type** `None`

**Raises** `Exception` – if `exception` is set.

`scriptworker.utils.remove_empty_keys(values, remove={}[, None][, 'null'])`

Recursively remove key/value pairs where the value is in `remove`.

This is targeted at comparing json-e rebuilt task definitions, since json-e drops key/value pairs with empty values.

**Parameters**

- **values** *(dict/list)* – the dict or list to remove empty keys from.

2.7. scriptworker package
Returns a dict or list copy, with empty keys removed.

Return type values (dict/list)

scriptworker.utils.request(context, url, timeout=60, method='get', good=(200, ), retry=(500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511), return_type='text', **kwargs)

Async aiohttp request wrapper.

Parameters

- **context** (scriptworker.context.Context) – the scriptworker context.
- **url** (str) – the url to request
- **timeout** (int, optional) – timeout after this many seconds. Default is 60.
- **method** (str, optional) – The request method to use. Default is ‘get’.
- **good** (list, optional) – the set of good status codes. Default is (200,)
- **retry** (list, optional) – the set of status codes that result in a retry. Default is tuple(range(500, 512)).
- **return_type** (str, optional) – The type of value to return. Takes ‘json’ or ‘text’; other values will return the response object. Default is text.
- ****kwars** – the kwars to send to the aiohttp request function.

Returns

the response text() if return_type is ‘text’; the response json() if return_type is ‘json’; the aiohttp request response object otherwise.

Return type object

Raises

- **ScriptWorkerRetryException** – if the status code is in the retry list.
- **ScriptWorkerException** – if the status code is not in the retry list or good list.

scriptworker.utils.retry_async(func: Callable[..., Awaitable[Any]], attempts: int = 5, sleeptime_callback: Callable[..., Any] = <function calculate_sleep_time>, retry_exceptions: Union[Type[BaseException], Tuple[Type[BaseException], ...]] = <class 'Exception'>, args: Sequence[Any] = (), kwars: Optional[Dict[str, Any]] = None, sleeptime_kwars: Optional[Dict[str, Any]] = None, log_exceptions: Optional[bool] = False) → Any

Retry func, where func is an awaitable function.

Parameters

- **func** (function) – an awaitable function.
- **attempts** (int, optional) – the number of attempts to make. Default is 5.
- **sleeptime_callback** (function, optional) – the function to use to determine how long to sleep after each attempt. Defaults to calculateSleepTime.
- **retry_exceptions** (list or exception, optional) – the exception(s) to retry on. Defaults to Exception.
- **args** (list, optional) – the args to pass to func. Defaults to ()
- **kwars** (dict, optional) – the kwars to pass to func. Defaults to { }.  

Chapter 2. Testing
• **sleeptime_kwargs** *(dict, optional)* – the kwargs to pass to `sleeptime_callback`. If None, use `{}`. Defaults to None.

Returns the value from a successful function call

Return type `object`

Raises Exception – the exception from a failed function call, either outside of the `retry_exceptions`, or one of those if we pass the max attempts.

```python
scriptworker.utils.retry_async_decorator(retry_exceptions: Union[Type[BaseException], Tuple[Type[BaseException], ...]] = <class 'Exception'>, sleeptime_kwargs: Optional[Dict[str, Any]] = None) → Callable[[...], Callable[[...], Awaitable[Any]]]
```

Decorate a function by wrapping `retry_async` around.

Parameters

• **retry_exceptions** *(list or exception, optional)* – the exception(s) to retry on. Defaults to Exception.

• **sleeptime_kwargs** *(dict, optional)* – the kwargs to pass to `sleeptime_callback`. If None, use `{}`. Defaults to None.

Returns the decorated function

Return type `function`

```python
scriptworker.utils.retry_request(*args, retry_exceptions=(<class 'asyncio.exceptions.TimeoutError'>, <class 'scriptworker.exceptions.ScriptWorkerRetryException'>), retry_async_kwargs=None, **kwargs)
```

Retry the `request` function.

Parameters

• *args – the args to send to `request()` through `retry_async()`.

• **retry_exceptions** *(list, optional)* – the exceptions to retry on. Defaults to (ScriptWorkerRetryException, ).

• **retry_async_kwargs** *(dict, optional)* – the kwargs for `retry_async`. If None, use `{}`. Defaults to None.

• **kwargs** – the kwargs to send to `request()` through `retry_async()`.

Returns the value from `request()`.

Return type `object`

```python
scriptworker.utils.retry_sync(func, attempts=5, sleeptime_callback=<function calculate_sleep_time>, retry_exceptions=<class 'Exception'>, args=(), kwargs=None, sleeptime_kwargs=None)
```

Retry `func`, where `func` is a regular function.

Please favor `retry_async` whenever possible.

Parameters

• **func** *(function)* – a function.

• **attempts** *(int, optional)* – the number of attempts to make. Default is 5.

• **sleeptime_callback** *(function, optional)* – the function to use to determine how long to sleep after each attempt. Defaults to `calculateSleepTime`. 

---

2.7. scriptworker package
• **retry_exceptions** (*list or exception, optional*) – the exception(s) to retry on. Defaults to `Exception`.

• **args** (*list, optional*) – the args to pass to `func`. Defaults to `()`. 

• **kwargs** (*dict, optional*) – the kwargs to pass to `func`. Defaults to `{}`.

• **sleeptime_kwargs** (*dict, optional*) – the kwargs to pass to `sleeptime_callback`. If None, use `{}`. Defaults to None.

**Returns** the value from a successful `func` call

**Return type** object

**Raises** `Exception` – the exception from a failed `func` call, either outside of the `retry_exceptions`, or one of those if we pass the max attempts.

**scriptworker.utils.rm(path)**
Equivalent to `rm -rf`.

Make sure `path` doesn’t exist after this call. If it’s a dir, `shutil.rmtree()`; if it’s a file, `os.remove()`; if it doesn’t exist, ignore.

**Parameters** `path` (*str*) – the path to nuke.

**scriptworker.utils.semaphore_wrapper(semaphore, coro)**
Wrap an async function with semaphores.

**Usage:**

```python
semaphore = asyncio.Semaphore(10)  # max 10 concurrent
futures = []
futures.append(asyncio.ensure_future(semaphore_wrapper(
    semaphore, do_something(arg1, arg2, kwarg1='foo'))))
await raise_future_exceptions(futures)
```

**Parameters**

• **semaphore** (*asyncio.Semaphore*) – the semaphore to wrap the action with

• **coro** (*coroutine*) – an asyncio coroutine

**Returns** the result of action.

**scriptworker.utils.to_unicode(line: Union[str, bytes]) → str**
Avoid `b'line'` type messages in the logs.

**Parameters** `line` (*str*) – The bytecode or unicode string.

**Returns**

the unicode-decoded string, if `line` was a bytecode string. Otherwise return `line` unmodified.

**Return type** str

**scriptworker.utils.write_to_file(path, contents, file_type='text')**
Write `contents` to `path` with optional formatting.

Small helper function to write `contents` to `file` with optional formatting.

**Parameters**

• **path** (*str*) – the path to write to
• **contents** (*str, object, or bytes*) – the contents to write to the file

• **file_type** (*str, optional*) – the type of file. Currently accepts text or binary (contents are unchanged) or json (contents are formatted). Defaults to text.

**Raises**

• `ScriptWorkerException` – with an unknown file_type

• `TypeError` – if file_type is json and contents isn’t JSON serializable

### 2.7.15 `scriptworker.worker` module

Scriptworker worker functions.

`scriptworker.worker.log`

the log object for the module.

`Type`  `logging.Logger`

class `scriptworker.worker.RunTasks`

Bases: `object`

Manages processing of Taskcluster tasks.

`cancel()`

Cancel current work.

`invoke(context)`

Claims and processes Taskcluster work.

**Parameters**

context (`scriptworker.context.Context`) – context of worker

Returns: status code of build

`scriptworker.worker.async_main(context, credentials)`

Set up and run tasks for this iteration.

https://firefox-ci-tc.services.mozilla.com/docs/reference/platform/queue/worker-interaction

**Parameters**

context (`scriptworker.context.Context`) – the scriptworker context.

`scriptworker.worker.do_run_task(context, run_cancellable, to_cancellable_process)`

Run the task logic.

Returns the integer status of the task.

**Parameters**

• context (`scriptworker.context.Context`) – the scriptworker context.

• run_cancellable (`typing.Callable`) – wraps future such that it’ll cancel upon worker shutdown

• to_cancellable_process (`typing.Callable`) – wraps TaskProcess such that it will stop if the worker is shutting down

**Raises**  `Exception` – on unexpected exception.

**Returns**  `exit status`

**Return type**  `int`
scriptworker.worker.do_upload(context, files)
Upload artifacts and return status.

Returns the integer status of the upload.

Parameters

• context (scriptworker.context.Context) – the scriptworker context.

• files (list of str) – list of files to be uploaded as artifacts

Raises Exception – on unexpected exception.

Returns exit status

Return type int

scriptworker.worker.main(event_loop=None)
Scriptworker entry point: get everything set up, then enter the main loop.

Parameters event_loop (asyncio.BaseEventLoop, optional) – the event loop to use.
If None, use asyncio.get_event_loop(). Defaults to None.

scriptworker.worker.run_tasks(context)
Run any tasks returned by claimWork.

Returns the integer status of the task that was run, or None if no task was run.

Parameters context (scriptworker.context.Context) – the scriptworker context.

Raises Exception – on unexpected exception.

Returns exit status None: if no task run.

Return type int

2.7.16 Module contents

Scriptworker.

2.8 Indices and tables

• genindex
• modindex
• search
s

scriptworker, 62
scriptworker.artifacts, 17
scriptworker.client, 20
scriptworker.config, 22
scriptworker.constants, 23
scriptworker.context, 23
scriptworker.cot.generate, 26
scriptworker.cot.verify, 27
scriptworker.ed25519, 39
scriptworker.exceptions, 42
scriptworker.github, 40
scriptworker.log, 44
scriptworker.task, 46
scriptworker.utils, 52
scriptworker.worker, 61
add_enumerable_item_to_dict() (in module scriptworker.utils), 52
add_projectid() (in module scriptworker.utils), 52
add_taskqueueid() (in module scriptworker.utils), 53
apply_product_config() (in module scriptworker.config), 22
assert_is_parent() (in module scriptworker.artifacts), 17
async_main() (in module scriptworker.worker), 61
AuditLogFormatter (class in scriptworker.cot.verify), 27
BaseDownloadError, 42
build_link() (in module scriptworker.cot.verify), 30
build_task_dependencies() (in module scriptworker.cot.verify), 30
build_taskcluster_yml_url() (in module scriptworker.cot.verify), 30
calculate_sleep_time() (in module scriptworker.utils), 53
cancel() (scriptworker.worker.RunTasks method), 61
ChainOfTrust (class in scriptworker.cot.verify), 27
check_and_update_action_task_group_id() (in module scriptworker.cot.verify), 31
check_config() (in module scriptworker.config), 22
check_interactive_docker_worker() (in module scriptworker.cot.verify), 31
check_interactive_generic_worker() (in module scriptworker.cot.verify), 31
claim_task (scriptworker.context.Context attribute), 24
claim_work() (in module scriptworker.task), 46
cleanup() (in module scriptworker.utils), 53
compare_jsone_task_definition() (in module scriptworker.cot.verify), 31
complete_task() (in module scriptworker.task), 46
compress_artifact_if_supported() (in module scriptworker.artifacts), 17
config (scriptworker.context.Context attribute), 24
ConfigError, 43
Context (class in scriptworker.context), 24
cot (scriptworker.cot.verify.ChainOfTrust attribute), 27
context (scriptworker.cot.verify.LinkOfTrust attribute), 29
cot_dir (scriptworker.cot.verify.LinkOfTrust attribute), 30
CoTError, 43
create_artifact() (in module scriptworker.artifacts), 17
create_config() (in module scriptworker.config), 22
create_queue() (scriptworker.context.Context method), 24
create_temp_creds() (in module scriptworker.utils), 53
create_test_workdir() (in module scriptworker.cot.verify), 32
credentials (scriptworker.context.Context attribute), 24
credentials_timestamp (scriptworker.context.Context attribute), 24, 25
CREDS_FILES (in module scriptworker.config), 22
datestring_to_timestamp() (in module scriptworker.utils), 53
decision_task_id (scriptworker.cot.verify.ChainOfTrust attribute), 27
datestring_to_timestamp() (in module scriptworker.utils), 53
decision_task_id (scriptworker.cot.verify.ChainOfTrust attribute), 27
decision_task_id (scriptworker.cot.verify.LinkOfTrust attribute), 29
DECISION_TASK_TYPES (in module scriptworker.cot.verify), 27
DEFAULT_CONFIG (in module scriptworker.constants), 23
DEFAULT_MAX_CONCURRENT_DOWNLOADS (in module scriptworker.context), 23
definition (scriptworker.github.GitHubRepository attribute), 41
dependent_task_ids() (scriptworker.cot.verify.ChainOfTrust method), 28
do_run_task() (in module scriptworker.worker), 61
do_upload() (in module scriptworker.worker), 61
download404, 43
download_artifacts() (in module scriptworker.artifacts), 17
download_cot() (in module scriptworker.cot.verify), 32
download_cot_artifact() (in module scriptworker.cot.verify), 32
download_cot_artifacts() (in module scriptworker.cot.verify), 32
download_file() (in module scriptworker.utils), 54
download_semaphore (scriptworker.context.Context attribute), 25
DownloadError, 43

E
ed25519_private_key_from_file() (in module scriptworker.cot.generate), 26
ed25519_private_key_from_file() (in module scriptworker.ed25519), 39
ed25519_private_key_from_string() (in module scriptworker.ed25519), 39
ed25519_private_key_to_string() (in module scriptworker.ed25519), 39
ed25519_public_key_from_file() (in module scriptworker.ed25519), 40
ed25519_public_key_from_string() (in module scriptworker.ed25519), 40
ed25519_public_key_to_string() (in module scriptworker.ed25519), 40
event_loop (scriptworker.context.Context attribute), 25
exit_code (scriptworker.exceptions.BaseDownloadError attribute), 43
exit_code (scriptworker.exceptions.ConfigError attribute), 43
exit_code (scriptworker.exceptions.CotError attribute), 43
exit_code (scriptworker.exceptions.Download404 attribute), 43
exit_code (scriptworker.exceptions.DownloadError attribute), 43
exit_code (scriptworker.exceptions.ScriptWorkerEd25519Error attribute), 43
exit_code (scriptworker.exceptions.ScriptWorkerException attribute), 43, 44
exit_code (scriptworker.exceptions.ScriptWorkerRetryException attribute), 44
exit_code (scriptworker.exceptions.ScriptWorkerTaskException attribute), 44
extract_github_repo_and_revision_from_source_url() (in module scriptworker.github), 41
extract_github_repo_full_name() (in module scriptworker.github), 42
extract_github_repo_owner_and_name() (in module scriptworker.github), 42
extract_github_repo_ssh_url() (in module scriptworker.github), 42

F
filepaths_in_dir() (in module scriptworker.utils), 54
find_sorted_task_dependencies() (in module scriptworker.cot.verify), 32
format() (scriptworker.cot.verify.AuditLogFormatter method), 27
format_json() (in module scriptworker.utils), 54

G
generate_cot() (in module scriptworker.cot.generate), 26
generate_cot_body() (in module scriptworker.cot.generate), 26
get_action_callback_name() (in module scriptworker.task), 46
get_action_context_and_template() (in module scriptworker.cot.verify), 33
get_all_artifacts_per_task_id() (in module scriptworker.cot.verify), 33
get_all_links_in_chain() (scriptworker.cot.verify.ChainOfTrust method), 28
get_and_check_single_upstream_artifact_full_path() (in module scriptworker.artifacts), 18
get_and_check_tasks_for() (in module scriptworker.task), 46
get_artifact_full_path() (scriptworker.cot.verify.LinkOfTrust method), 30
get_artifact_url() (in module scriptworker.artifacts), 18
get_branch() (in module scriptworker.task), 46
is_decision() (scriptworker.cot.verify.ChainOfTrust method), 28
is_github_repo_owner_the_official_one() (in module scriptworker.github), 42
is_github_task() (in module scriptworker.task), 50
is_github_url() (in module scriptworker.github), 42
is_pull_request() (in module scriptworker.task), 50
is_scope_in_restricted_scopes() (scriptworker.cot.verify.ChainOfTrust method), 29
is_try() (in module scriptworker.task), 50
is_try_or_pull_request(scriptworker.cot.verify.LinkOfTrust attribute), 29
is_try_or_pull_request() (in module scriptworker.task), 51
is_try_or_pull_request() (scriptworker.cot.verify.ChainOfTrust method), 29
is_try_or_pull_request() (scriptworker.cot.verify.LinkOfTrust method), 30

L
LinkOfTrust (class in scriptworker.cot.verify), 29
links (scriptworker.cot.verify.ChainOfTrust attribute), 29
load_json_or_yaml() (in module scriptworker.utils), 55
load_json_or_yaml_from_url() (in module scriptworker.utils), 56
log (in module scriptworker.client), 20
log (in module scriptworker.config), 22
log (in module scriptworker.context), 23
log (in module scriptworker.cot.generate), 26
log (in module scriptworker.cot.verify), 27
log (in module scriptworker.ed25519), 39
log (in module scriptworker.log), 44
log (in module scriptworker.task), 46
log (in module scriptworker.utils), 52
log (in module scriptworker.worker), 61

M
main() (in module scriptworker.worker), 62
makedirs() (in module scriptworker.utils), 56
match_url_path_callback() (in module scriptworker.utils), 56
match_url_regex() (in module scriptworker.utils), 56

N
name (scriptworker.cot.verify.ChainOfTrust attribute), 28
name (scriptworker.cot.verify.LinkOfTrust attribute), 29
parent_task_id (scriptworker.cot.verify.ChainOfTrust attribute), 27
parent_task_id (scriptworker.cot.verify.LinkOfTrust attribute), 29
PARENT_TASKTYPES (in module scriptworker.cot.verify), 27
pipe_to_log() (in module scriptworker.log), 45
populate_jsone_context() (in module scriptworker.cot.verify), 35
populate_projects() (scriptworker.context.Context method), 25
prepare_to_run_task() (in module scriptworker.task), 51
proc(scriptworker.context.Context attribute), 24, 25
projects (scriptworker.context.Context attribute), 25
Q
queue (scriptworker.context.Context attribute), 24, 25
R
raise_future_exceptions() (in module scriptworker.utils), 57
raise_on_errors() (in module scriptworker.cot.verify), 35
read_from_file() (in module scriptworker.utils), 57
read_worker_creds() (in module scriptworker.config), 23
reclaim_task (scriptworker.context.Context attribute), 25
reclaim_task() (in module scriptworker.task), 51
remove_empty_keys() (in module scriptworker.utils), 57
REPO_SCOPE_REGEX (in module scriptworker.task), 46
request() (in module scriptworker.utils), 58
retry_async() (in module scriptworker.utils), 58
retry_async_decorator() (in module scriptworker.utils), 59
retry_create_artifact() (in module scriptworker.artifacts), 19
retry_get_task_definition() (in module scriptworker.task), 51
retry_request() (in module scriptworker.utils), 59
retry_sync() (in module scriptworker.utils), 59
rm() (in module scriptworker.utils), 60
run_task() (in module scriptworker.task), 52
run_tasks() (in module scriptworkerworker), 62
running_tasks (scriptworker.context.Context attribute), 25
RunTasks (class in scriptworker.worker), 61
S
scriptworker (module), 62
scriptworker.artifacts (module), 17
scriptworker.client (module), 20
scriptworker.config (module), 22
scriptworker.constants (module), 23
scriptworker.context (module), 23
scriptworker.cot.generate (module), 26
scriptworker.cot.verify (module), 27
scriptworker.ed25519 (module), 39
scriptworker.exceptions (module), 42
scriptworker.github (module), 40
scriptworker.log (module), 44
scriptworker.task (module), 46
scriptworker.utils (module), 52
scriptworker.worker (module), 61
ScriptWorkerEd25519Error, 43
ScriptWorkerException, 43
ScriptWorkerRetryException, 44
ScriptWorkerTaskException, 44
semaphore_wrapper() (in module scriptworker.utils), 60
session (scriptworker.context.Context attribute), 24, 25
status (scriptworker.cot.verify.LinkOfTrust attribute), 30
STATUSES (in module scriptworker.constants), 23
sync_main() (in module scriptworker.client), 20
T
task (scriptworker.context.Context attribute), 24, 25
task (scriptworker.cot.verify.LinkOfTrust attribute), 30
task_graph (scriptworker.cot.verify.LinkOfTrust attribute), 29, 30
task_id (scriptworker.context.Context attribute), 25
task_id (scriptworker.cot.verify.ChainOfTrust attribute), 28
task_id (scriptworker.cot.verify.LinkOfTrust attribute), 29
task_type (scriptworker.cot.verify.ChainOfTrust attribute), 28
task_type (scriptworker.cot.verify.LinkOfTrust attribute), 29
TaskVerificationError, 44
temp_credentials (scriptworker.context.Context attribute), 25
temp_queue (scriptworker.context.Context attribute), 24, 26
to_unicode() (in module scriptworker.utils), 60
trace_back_to_tree() (in module scriptworker.cot.verify), 35
U
update_logging_config() (in module scriptworker.log), 45
upload_artifacts() (in module scriptworker.artifacts), 20
V
validate_artifact_url() (in module scriptworker.client), 21
validate_json_schema() (in module scriptworker.client), 21
validate_task_schema() (in module scriptworker.client), 21
verify_build_task() (in module scriptworker.cot.verify), 36
verify_chain_of_trust() (in module scriptworker.cot.verify), 36
verify_cot_cmdln() (in module scriptworker.cot.verify), 36
verify_cot_signatures() (in module scriptworker.cot.verify), 36
verify_docker_image_sha() (in module scriptworker.cot.verify), 36
verify_docker_image_task() (in module scriptworker.cot.verify), 36
verify_docker_worker_task() (in module scriptworker.cot.verify), 37
verify_headers() (in module scriptworker.cot.verify), 37
verify_link_ed25519() (in module scriptworker.cot.verify), 37
verify_link_ed25519_cot_signature() (in module scriptworker.cot.verify), 37
verify_link_in_task_graph() (in module scriptworker.cot.verify), 37
verify_parent_task() (in module scriptworker.cot.verify), 37
verify_parent_task_definition() (in module scriptworker.cot.verify), 37
verify_partials_task() (in module scriptworker.cot.verify), 37
verify_repo_matches_url() (in module scriptworker.cot.verify), 38
verify_scriptworker_task() (in module scriptworker.cot.verify), 38
verify_task() (scriptworker.context.Context method), 26
verify_task_in_task_graph() (in module scriptworker.cot.verify), 38
verify_task_types() (in module scriptworker.cot.verify), 39
verify_worker_impls() (in module scriptworker.cot.verify), 39

Index
W

worker_impl  (scriptworker.cot.verify.ChainOfTrust attribute), 28
worker_impl  (scriptworker.cot.verify.LinkOfTrust attribute), 30
WorkerShutdownDuringTask, 44
worst_level()  (in module scriptworker.task), 52
write_json()  (scriptworker.context.Context method), 26
write_to_file()  (in module scriptworker.utils), 60