# Orcoursetrion Documentation

***Release 0.1.1***

**MIT Office of Digital Learning**

February 20, 2015

Contents

Automatic course provisioning for the edx-platform using github and zendesk.

CHAPTER 1

# Quick Start

To get started, clone the repository, and run `pip install .` or just install directory from github.com with `pip install git+https://github.com/mitodl/orcoursetrion`.

Once installed, create or acquire an OAUTH2 token from github. That at least has the `repo`, `write:repo_hook`, and `write:org` permissions.

Add the environment variable `ORC_GH_OAUTH2_TOKEN=<your token>` to your environment, and run `orcoursetrion --help` for available commands and actions.

If you are adding an XML course, you will also need to define `ORC_STAGING_GITRELOAD` in your environment for where Web hooks should be sent for push events.

CHAPTER 2

# Table of Contents

## 2.1 Command Line

There is an exposed command line interface that is available upon installation of the repository. It can be run with `orcoursetrion`, and `orcoursetrion --help` will provide the most up to date help information.

The command allows you to run commands that correspond to actions, currently the only supported action is `create_export_repo`, and if your configuration is setup correctly (see *Configuration*), and at least minimally have set `ORC_GH_OAUTH2_TOKEN` and you run `orcoursetrion create_export_repo -t Spring_2030 -c DevOps.001 -d 'My awesome class repo'` you should see it respond with the URL of the repo that it just created for you.

## 2.2 Available Actions

**create_export_repo** This will create a new repository with the content deployment team from `ORC_STUDIO_DEPLOY_TEAM` added to the repository.

**create_xml_repo** This will create a new repository with the `ORC_XML_DEPLOY_TEAM` and a command line specified team added to repository. It will also set up a git hook to the URL specified with `ORC_STAGING_GITRELOAD`.

## 2.3 Design

### 2.3.1 Workflow, Design, And Architecture

Quite some time is lost to course provisioning and request management. This document both defines what course provisioning is, and is a design for how to create software to assist in automating this process.

There are three major stories/workflows that occur in our course production/publishing architecture. While all three occur for residential MITx, some of them also apply and could be used for use with MOOC/Open Education.

The three major process flows are:

- Creating a brand new course
- Publishing a course to our student facing LMS
- Rerunning an already existing course

### 2.3.2 Creating a Brand New Course

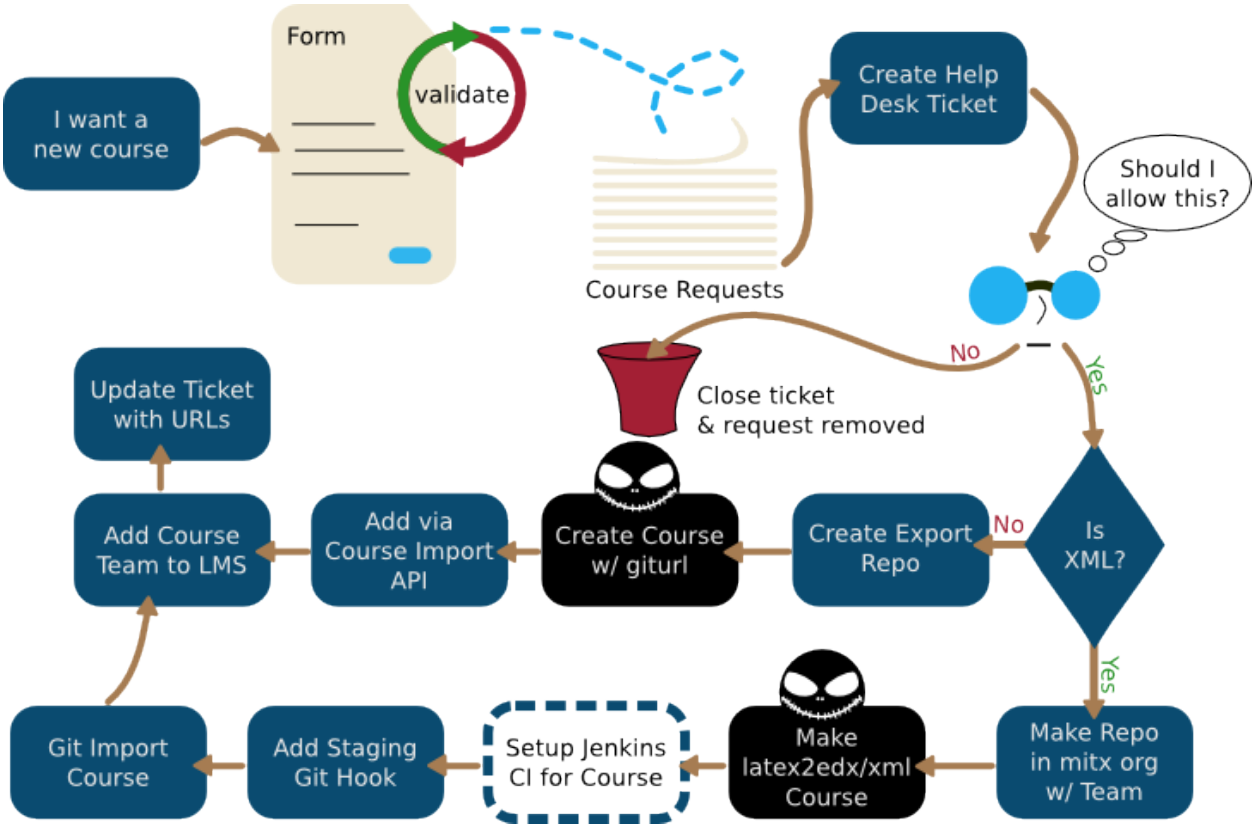


Figure 2.1: New Course

#### Areas of note

- Setting up Jenkins CI for the course is doable, but is left in cutout in the diagram as it is a quite complex and not required for course provisioning.
- The form needs to both programmatically validate course team members, and the `course id` entered (i.e. 8.01r). It is intended that the validation for `course id` will either use the edX-Platform course listing API directly to check for duplicates, or maintain a (mostly) complete list of all current courses internally
- A fully automatic mode could be supported without the help desk ticket approval step.

### 2.3.3 Moving a Course from Staging to Production

#### Areas of note

- The course quality check could potentially be done by a jenkins job, or ACPx could provide that feature itself since it will already have to be somewhat courseware capable.
- Similar to a new course, the approval process can be skipped here

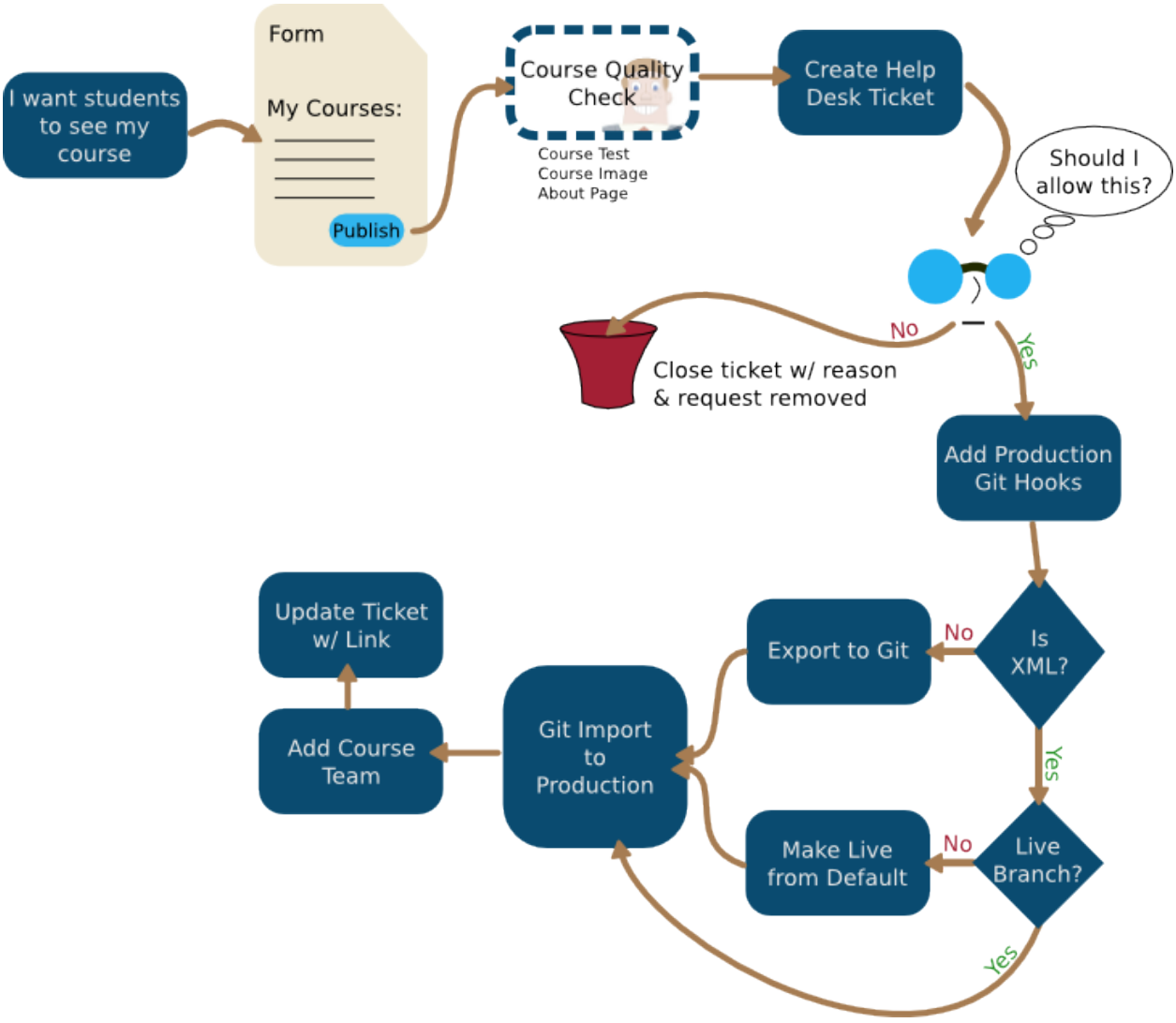


Figure 2.2: Moving a Course

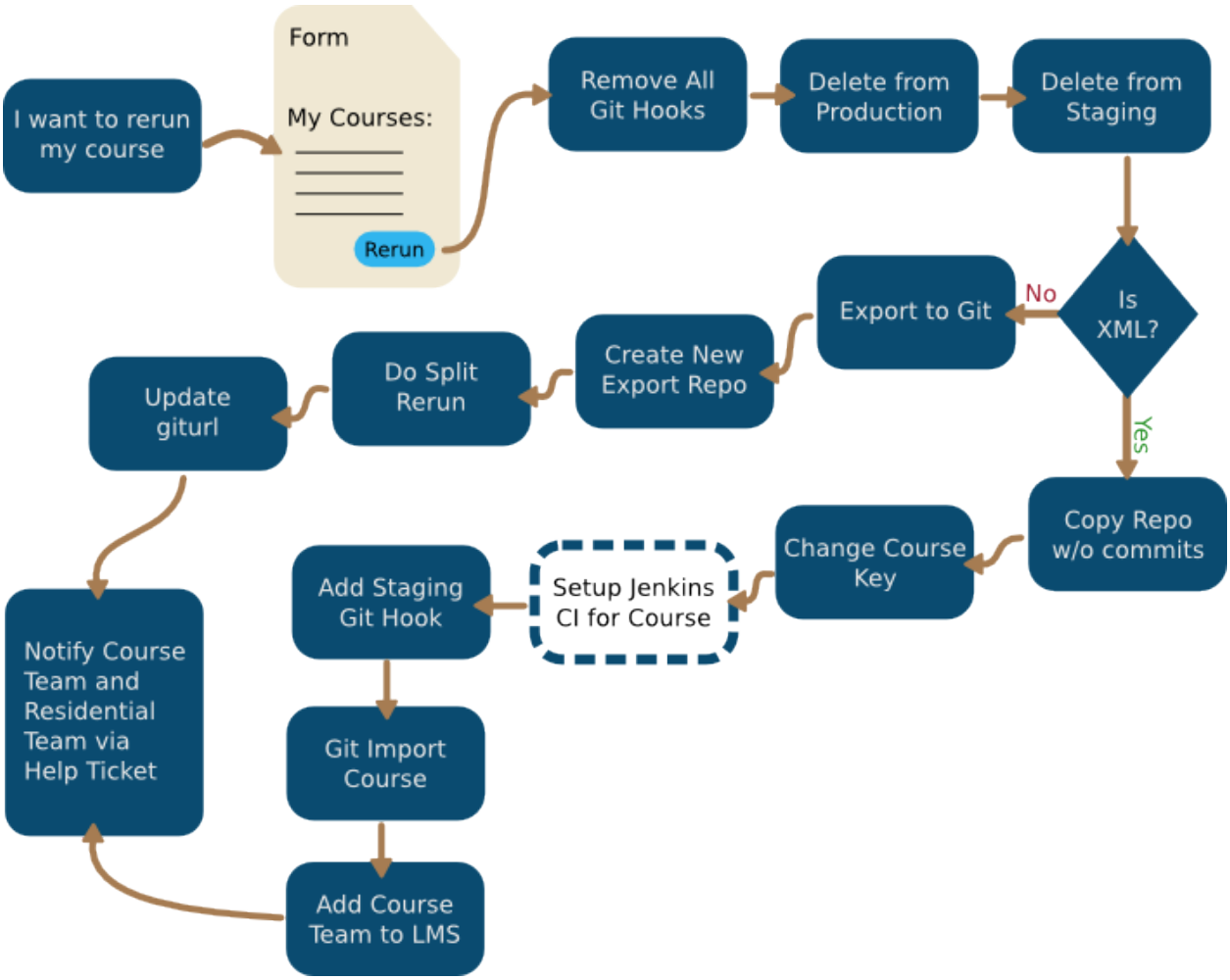


Figure 2.3: Rerunning a Course

### 2.3.4 Rerunning a Course

#### Areas of note

- This is pretty much a blend of the move and rerun flows, and only adds the rerun edX platform call, the deletions, and the giturl update.
- As an option, instead of doing the split rerun function above, we could potentially use the same flow as the git course by deleting it, modifying the course XML (and using an XML cleanup script i.e. Piotr's Cleanup Script, and re-importing.

### 2.3.5 System Architecture

#### Areas of note

- None of the APIs for edx-platform currently exist and will likely have to be replaced with instruction updates to the helpdesk tickets instructing the agent to perform the requested task and provide instructions that make it as quickj and easy as possible for the agent to do so, and mark them complete.
- We have two of the eight needed API calls implemented in a feature branch:
  - Import Tar
  - Course List and we can likely easily add the rest if that feature branch is merged.

### 2.3.6 Data Structure/Entity Relationship Diagram

Below is an Oracle style ERD diagram, but I do not presuppose the database choice, and do not think it is neccessary to use a traditional RDBMS as the persistance layer in the application. The persistence area is a flexible area within the architecture since even a simple document store would likely be adequate.

## 2.4 Orcoursetrion API Docs

For convenient reference in development, here are the Orcoursetrion API docs.

### 2.4.1 Actions

The actions that are available to use. Action library access

`orcoursetrion.actions.`**`create_export_repo`**(*course*, *term*, *description=None*)

Creates a course repo at `ORC_GH_API_URL` with key `ORC_GH_OAUTH2_TOKEN`, at organization `ORC_STUDIO_ORG`, and with collabarator `ORC_STUDIO_DEPLOY_TEAM`

**Parameters**

- **course** (*str*) – Course name to be used to name repo (i.e. 6.004r)
- **term** (*str*) – Term the course is expected to run (i.e. 2015_Spring)
- **description** (*str*) – Optional description for repo to show up on github

**Returns** Github dictionary of a repo (https://developer.github.com/v3/repos/#create)

**Return type** dict

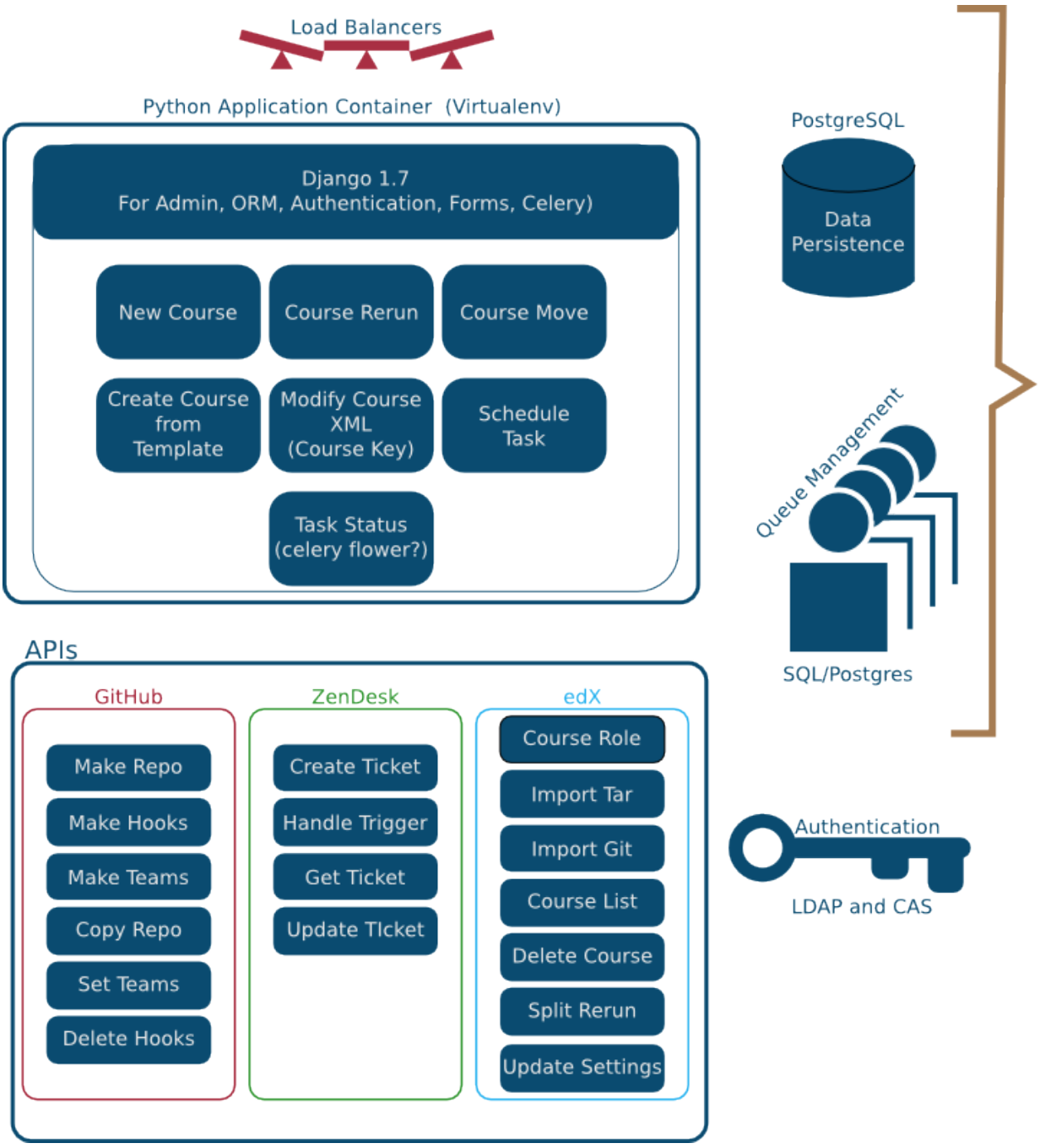


Figure 2.4: System Architecture diagram

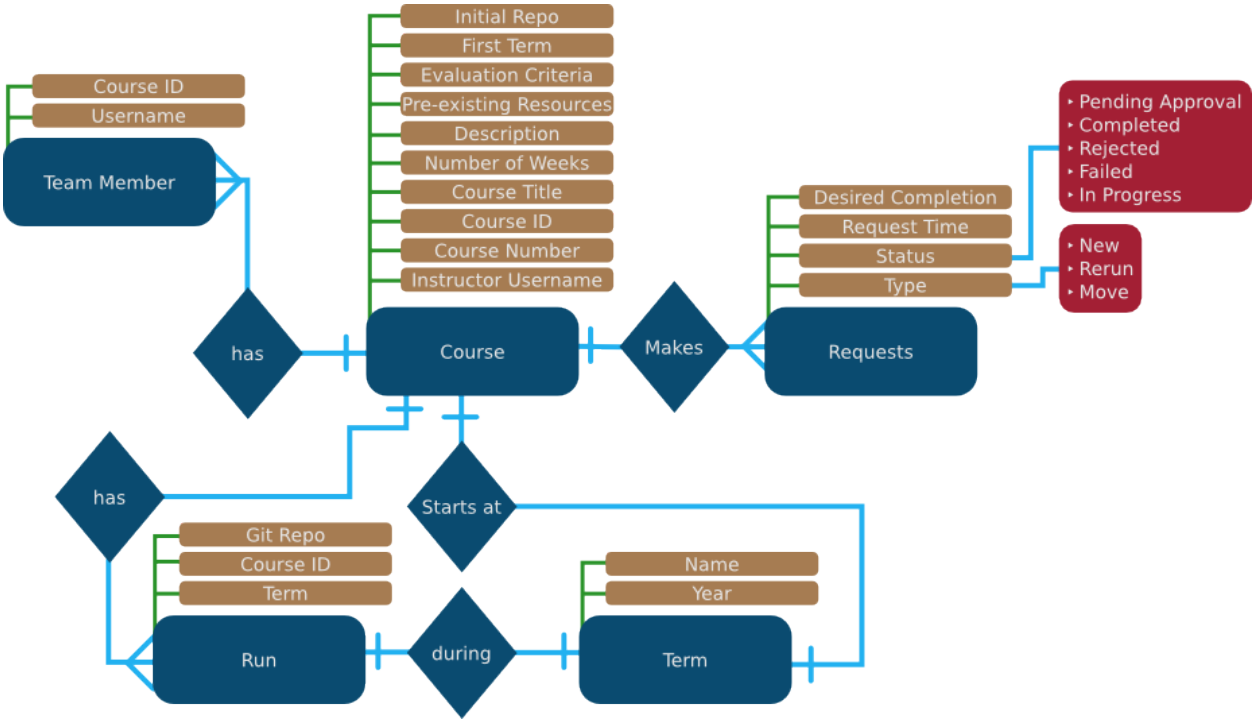


Figure 2.5: Entity Relationship Diagram

orcoursetrion.actions.**create_xml_repo**(*course*, *term*, *team*, *description=None*)

Creates a course repo at ORC_GH_API_URL with key ORC_GH_OAUTH2_TOKEN and at organization ORC_XML_ORG, and with team as a collaborator (Along with ORC_XML_DEPLOY_TEAM).

This also adds a github Web hook to the course development environment gitreload server via ORC_STAGING_GITRELOAD.

**Parameters**

- **course** (*str*) – Course name to be used to name repo (i.e. 6.004r)
- **term** (*str*) – Term the course is expected to run (i.e. 2015_Spring)
- **team** (*str*) – Name of an organizational team that already exists to add read/write access to this repo.
- **description** (*str*) – Optional description for repo to show up on github

**Returns** Github dictionary of a repo (https://developer.github.com/v3/repos/#create)

**Return type** dict

### 2.4.2 Library

API libraries. Orchestrion library

**class** orcoursetrion.lib.**GitHub**(*api_url*, *oauth2_token*)

Bases: object

API class for handling calls to github

Initialize a requests session for use with this class by specifying the base API endpoint and key.

**Parameters**

- **api_url** (*str*) – Github API URL such as https://api.github.com/
- **oauth2_token** (*str*) – Github OAUTH2 token for v3

**add_team_repo** (*org*, *repo*, *team*)
Add a repo to an existing team (by name) in the specified org.

We first look up the team to get its ID (https://developer.github.com/v3/orgs/teams/#list-teams), and then add the repo to that team (https://developer.github.com/v3/orgs/teams/#add-team-repo).

**Parameters**

- **org** (*str*) – Organization to create the repo in.
- **repo** (*str*) – Name of the repo to create.
- **team** (*str*) – Name of team to add.

**Raises**

- GitHubNoTeamFound –
- GitHubUnknownError –
- requests.RequestException –

**add_web_hook** (*org*, *repo*, *url*)
Adds an active hook to a github repository.

This utilizes https://developer.github.com/v3/repos/hooks/#create-a-hook to create a form type Web hook that responds to push events (basically all the defaults).

**Parameters**

- **org** (*str*) – Organization to create the repo in.
- **repo** (*str*) – Name of the repo to create.
- **url** (*str*) – URL of the hook to add

**Raises**

- GitHubUnknownError –
- requests.RequestException –

**Returns** Github dictionary of a hook (https://developer.github.com/v3/repos/hooks/#response-2)

**Return type** dict

**create_repo** (*org*, *repo*, *description*)
Creates a new github repository or raises exceptions

**Parameters**

- **org** (*str*) – Organization to create the repo in.
- **repo** (*str*) – Name of the repo to create.
- **description** (*str*) – Description of repo to use.

**Raises**

- GitHubRepoExists –
- GitHubUnknownError –
- requests.RequestException –

**Returns** Github dictionary of a repo (https://developer.github.com/v3/repos/#create)

**Return type** dict

**exception** orcoursetrion.lib.**GitHubException**
Bases: exceptions.Exception

Base exception class others inherit.

**exception** orcoursetrion.lib.**GitHubRepoExists**
Bases: orcoursetrion.lib.github.GitHubException

Repo exists, and thus cannot be created.

**exception** orcoursetrion.lib.**GitHubUnknownError**
Bases: orcoursetrion.lib.github.GitHubException

Unexpected status code exception

**exception** orcoursetrion.lib.**GitHubNoTeamFound**
Bases: orcoursetrion.lib.github.GitHubException

Name team not found in list

### 2.4.3 Configuration

Configuration options Configuration needed for Orchestrion to function (i.e. API keys)

config.**ORC_GH_OAUTH2_TOKEN = GitHub OAUTH2 Token**

config.**ORC_GH_API_URL = GitHub API URL**

config.**ORC_COURSE_PREFIX = Prefix to use in repository name**

config.**ORC_STUDIO_ORG = Organization to use for Studio export repos**

config.**ORC_STUDIO_DEPLOY_TEAM = Deployment team for Studio Export repos**

config.**ORC_XML_ORG = Organization to use for XML/latex2edx courses**

config.**ORC_XML_DEPLOY_TEAM = Deployment team for XML/latex2edx courses**

config.**ORC_STAGING_GITRELOAD = 'gitreload <https://github.com/mitodl/gitreload>'_ server URL (including username a**

CHAPTER 3

# Indices and Search

- *genindex*
- *modindex*
- *search*

## O

## A



## C



## G



## O