Codex User Guide

Pedro Vasconcelos, pbv@dcc.fc.up.pt.

Version 0.8, January 2017.

Codex is a web system for setting up exercises with automatic assessment for programming classes. Unlike other systems for this same purpose, it aims on providing good automatic feedback by using state-of-art automatic testing tools. It is also directed more towards a learning environment rather than for programming contests; Mooshak is a better tool for the later purpose.

Althought *Codex* is stil in early development stage, it is already being used in the Faculty of Science of the University of Porto for teaching introductory courses on programming in Python.

This guide explains how to write *Codex* exercise pages.

Pages

Codex repositories are organized into *pages*. A page can contain formated text, links, tables, images, mathematics, etc.

Codex pages are simple text files with the `.md` extension; Markdown annotations are used for formating. Here is an example page:

```
# This is a header

## This is a sub-header

This is the first paragraph. This sentence shows *emphasis*,
**strong emphasis** and `inline-code style`.

This is another paragraph with an itemized list:

1. first item;
2. second item;
3. last item.

~~~{.python}
# a verbatim code block (with Python highlighting)
def dist(x, y):
   return sqrt(x*x + y*y)
~~~

This is a link to [Google's home page](http://www.google.com).
```

```
You can also include LaTeX mathematics either
inline $h = \sqrt{x^2+y^2}$ or as displayed equations:
$$ erf(x) = \frac{1}{\sqrt{x}}\int_{-x}^x e^{-t^2} dt\,. $$
```

The above could be rendered in HTML as follows:

This is a header

This is a sub-header

This is the first paragraph. This sentence shows *emphasis*, **strong emphasis** and `inline-code style`.

This is another paragraph with an itemized list:

1. first item;
2. second item;
3. last item.

```
# a verbatim code block (with Python highlighting)
def dist(x, y):
  return sqrt(x*x + y*y)
```

This is a link to Google's home page.

You can also include LaTeX mathematics either inline $h = \sqrt{x^2+y^2}$ or as displayed equations:

$$erf(x) = \frac{1}{\sqrt{x}}\int_{-x}^{x} e^{-t^2} dt\,.$$

Codex uses the Pandoc library for reading and rendering Markdown and MathJaX for displaying mathematics. For details on the Markdown syntax accepted, check the Pandoc user manual. Note, however, that raw HTML markup is *not* accepted (it will simply be escaped and rendered as ordinary text).

Metadata blocks

Markdown text can also include YAML metadata blocks delimited between 3 dashes (`---`) and 3 full stops (`...`); here is an example:

```
---
author: Pedro Vasconcelos
title: A sample exercise
exercise: true
language: python
...
```

Metadata blocks can occur anywhere, but the convention is to put them at the beginning of the document. Several metatata blocks are also allowed and equivalent to a single one with all collected fields.

Some fields (like `author` and `title`) are generic, while others (like `exercise` and `language`) are specific to exercise testing in Codex. These are described in detail in a later section.

Exercise pages

A page marked with metadata `exercise: true` is an *exercise page*; this means that users will be able to:

- *submit* solutions for automatic assessment;
- *view feedback* on their submissions;
- *view* past submissions and feedback;
- *edit and re-submit* past submissions.

Any user can submit to any exercise. Note that exercises are identified in the submissions database by the page's *request path* relative to the `/pub` handle; e.g. an exercise with URL `https://server.domain/pub/foo/bar.md` is identified as `foo/bar.md`. This means that if the file name or path are modified, any previous submissions will no longer be visible (but will still be recorded in the database).

Linking exercise pages

After a successful login, Codex shows an `index.md` page; authors should edit this page to link other pages and exercises.

For example, supose you have created exercises `work1.md`, `work2.md` and `work3.md`; here is a suitable `index.md` page:

```
# Welcome!

Here is a list of available exercises:

1. [](work1.md){.ex}
2. [](work2.md){.ex}
3. [](work3.md){.ex}
```

Links for exercises are marked with a special class `.ex`. Codex will then automatically fill-in the exercise title for link anchor text with and insert a short summary of previous submissions by the logged-in user.

Note also that authors can freely modify exercise order, or group exercises by using sections or sub-pages. It is also possible to include explanatory pages,

images or links to external resources.

Metadata fields

`title` Specify a title for the exercise; if this is field missing, the first header is used instead.

`language` Specify the programming language for this exercise, e.g. `python`, `haskell`, `c`

`valid` Specify valid submission time interval; the default is `always` which means submissions are always valid. Some alternatives:

- `after 08:00 15/02/2017`
- `between 08:00 15/02/2017 and 12:00 15/02/2017`
- `after 16/02/2017`

Note that dates and times are interpreted relative to the server local timezone.

`feedback` Specify the level of feedback to report (0-100); 0 means no feedback, 50 shows classifications only, 100 shows classifications and failed test cases (default).

`code` Specify an initial "skeleton" for solutions; use an indented block for multiple lines, e.g.:

```
code: |
  ~~~
  # distance between two points in the plane
  def distance(x1, y1, x2, y2):
      # complete this definition
  ~~~
```

Note the vertical bar (`|`) and indentation in the above example.

The following fields are specific to programming languages.

Python fields

`doctest` Specifies the file path for a doctest script for submission testing; if omitted, this defaults to the file name for exercise page with extension replaced by `.tst`, e.g. the doctest for `foo/bar.md` would be `foo/bar.tst`.

Haskell and C fields

`quickcheck` Specifies the file name for QuickCheck properties for submission testing.

`maxSuccess` Number of QuickCheck tests run.

`maxSize` Maximum size for QuickCheck generated test cases.

`maxDiscardRatio` Maximum number of discarded tests per successful test before giving up.

`randSeed` Integer seed value for pseudo-random test cases generation; use to ensure the reproducibility of QuickCheck test cases.

Assement and Feedback

Codex assesses submissions by testing them against a large number of test cases (either provided by the author or randomly-generated). The result of testing is a *classification label*, a *timing label* and a *detailed text message*. The classification labels are similar to the ones used for ICPC programming contests (e.g. *Accepted*, *WrongAnswer*, etc.).

When submission are rejected because of a wrong answer, the text message is a human-readable description of a failed test case; this can be used by the student as a starting point for aiding debugging.

Note that Codex will *always* process submissions regardless of the timing interval specified an exercise; however:

- the system will not show any feedback for early submissions until the start of submission interval;
- late submissions will be marked as *Overdue*; it is up to the exercise author to decide how handle these cases.

Classification labels

Accepted The submission passed all tests.

WrongAnswer The submission was rejected because it failed at least one test case.

CompileError The submission was rejected because it caused a compile-time error.

RuntimeError The submission was rejected because it caused a runtime error (e.g. runtime exception, segmentation fault)

RuntimeLimitExceeded*, *MemoryLimitExceeded The submission was reject because it tried to use too much computing resources (e.g. non-terminating program).

MiscError Some other error (e.g. incorrect metadata fields, test files, etc.)

Evaluating Temporary result while waiting for evaluation; end-users should never see this.

Timing labels

Early Received before the start of submission interval.

Valid Received within the valid submission interval.

Overdue Received after the end of the submission interval.

Pedro Vasconcelos, 2017.