# edX Studio Documentation

***Release 0.1***

**EdX Doc Team**

October 27, 2013

# CONTENTS

The following documents are targeted at those who are working with various data formats consumed and produced by the edX platform – primarily course authors and those who are conducting research on data in our system. Developer oriented discussion of architecture and strictly internal APIs should be documented elsewhere.

CHAPTER

ONE

# COURSE DATA FORMATS

These are data formats written by people to specify course structure and content. Some of this is abstracted away if you are using the Studio authoring user interface.

## 1.1 Course XML Tutorial

EdX uses an XML format to describe the structure and contents of its courses. While much of this is abstracted away by the Studio authoring interface, it is still helpful to understand how the edX platform renders a course.

This guide was written with the assumption that you've dived straight into the edX platform without necessarily having any prior programming/CS knowledge. It will be especially valuable to you if your course is being authored with XML files rather than Studio – in which case you're likely using functionality that is not yet fully supported in Studio.

### 1.1.1 Goals

After reading this, you should be able to:

- Organize your course content into the files and folders the edX platform expects.
- Add new content to a course and make sure it shows up in the courseware.

*Prerequisites:* it would be helpful to know a little bit about xml. Here is a simple example if you've never seen it before.

### 1.1.2 Introduction

A course is organized hierarchically. We start by describing course-wide parameters, then break the course into chapters, and then go deeper and deeper until we reach a specific pset, video, etc. You could make an analogy to finding a green shirt in your house -> bedroom -> closet -> drawer -> shirts -> green shirt.

We'll begin with a sample course structure as a case study of how XML and files in a course are organized. More technical details will follow, including discussion of some special cases.

### 1.1.3 Case Study

Let's jump right in by looking at the directory structure of a very simple toy course:

```
toy/
    course/
    course.xml
    problem/
    policies/
    roots/
```

The only top level file is *course.xml*, which should contain one line, looking something like this:

```
<course org="edX" course="toy" url_name="2012_Fall"/>
```

This gives all the information to uniquely identify a particular run of any course – which organization is producing the course, what the course name is, and what "run" this is, specified via the *url_name* attribute.

Obviously, this doesn't actually specify any of the course content, so we need to find that next. To know where to look, you need to know the standard organizational structure of our system: course elements are uniquely identified by the combination *(category, url_name)*. In this case, we are looking for a *course* element with the *url_name* "2012_Fall". The definition of this element will be in *course/2012_Fall.xml*. Let's look there next:

```
toy/
    course/
           2012_Fall.xml # <-- Where we look for category="course", url_name="2012_Fall"
```

```
<!-- Contents of course/2012_Fall.xml -->
<course>
  <chapter url_name="Overview">
    <videosequence url_name="Toy_Videos">
      <problem url_name="warmup"/>
      <video url_name="Video_Resources" youtube="1.0:1bK-WdDi6Qw"/>
    </videosequence>
    <video url_name="Welcome" youtube="1.0:p2Q6BrNhdh8"/>
  </chapter>
</course>
```

Aha. Now we've found some content. We can see that the course is organized hierarchically, in this case with only one chapter, with *url_name* "Overview". The chapter contains a *videosequence* and a *video*, with the sequence containing a problem and another video. When viewed in the courseware, chapters are shown at the top level of the navigation accordion on the left, with any elements directly included in the chapter below.

Looking at this file, we can see the course structure, and the youtube urls for the videos, but what about the "warmup" problem? There is no problem content here! Where should we look? This is a good time to pause and try to answer that question based on our organizational structure above.

As you hopefully guessed, the problem would be in *problem/warmup.xml*. (Note: This tutorial doesn't discuss the xml format for problems – there are chapters of edx4edx that describe it.) This is an instance of a *pointer tag*: any xml tag with only the category and a url_name attribute will point to the file *{category}/{url_name}.xml*. For example, this means that our toy *course.xml* could have also been written as

```
<!-- Contents of course/2012_Fall.xml -->
<course>
  <chapter url_name="Overview"/>
</course>
```

with *chapter/Overview.xml* containing

```
<chapter>
  <videosequence url_name="Toy_Videos">
    <problem url_name="warmup"/>
    <video url_name="Video_Resources" youtube="1.0:1bK-WdDi6Qw"/>
  </videosequence>
  <video url_name="Welcome" youtube="1.0:p2Q6BrNhdh8"/>
</chapter>
```

In fact, this is the recommended structure for real courses – putting each chapter into its own file makes it easy to have different people work on each without conflicting or having to merge. Similarly, as sequences get large, it can be handy to split them out as well (in *sequence/{url_name}.xml*, of course).

Note that the *url_name* is only specified once per element – either the inline definition, or in the pointer tag.

## Policy Files

We still haven't looked at two of the directories in the top-level listing above: *policies* and *roots*. Let's look at policies next. The policy directory contains one file:

```
toy/
    policies/
             2012_Fall.json
```

and that file is named *{course-url_name}.json*. As you might expect, this file contains a policy for the course. In our example, it looks like this:

```
{
    "course/2012_Fall": {
        "graceperiod": "2 days 5 hours 59 minutes 59 seconds",
        "start": "2015-07-17T12:00",
        "display_name": "Toy Course"
    },
    "chapter/Overview": {
        "display_name": "Overview"
    },
    "videosequence/Toy_Videos": {
        "display_name": "Toy Videos",
        "format": "Lecture Sequence"
    },
    "problem/warmup": {
        "display_name": "Getting ready for the semester"
    },
    "video/Video_Resources": {
        "display_name": "Video Resources"
    },
    "video/Welcome": {
        "display_name": "Welcome"
    }
}
```

The policy specifies metadata about the content elements – things which are not inherent to the definition of the content, but which describe how the content is presented to the user and used in the course. See below for a full list of metadata attributes; as the example shows, they include *display_name*, which is what is shown when this piece of content is referenced or shown in the courseware, and various dates and times, like *start*, which specifies when the content becomes visible to students, and various problem-specific parameters like the allowed number of attempts. One important point is that some metadata is inherited: for example, specifying the start date on the course makes it the default for every element in the course. See below for more details.

It is possible to put metadata directly in the XML, as attributes of the appropriate tag, but using a policy file has two benefits: it puts all the policy in one place, making it easier to check that things like due dates are set properly, and it allows the content definitions to be easily used in another run of the same course, with the same or similar content, but different policy.

### Roots

The last directory in the top level listing is *roots*. In our toy course, it contains a single file:

```
roots/
      2012_Fall.xml
```

This file is identical to the top-level *course.xml*, containing

```
<course org="edX" course="toy" url_name="2012_Fall"/>
```

In fact, the top level *course.xml* is a symbolic link to this file. When there is only one run of a course, the roots directory is not really necessary, and the top-level course.xml file can just specify the *url_name* of the course. However, if we wanted to make a second run of our toy course, we could add another file called, e.g., *roots/2013_Spring.xml*, containing

```
<course org="edX" course="toy" url_name="2013_Spring"/>
```

After creating *course/2013_Spring.xml* with the course structure (possibly as a symbolic link or copy of *course/2012_Fall.xml* if no content was changing), and *policies/2013_Spring.json*, we would have two different runs of the toy course in the course repository. Our build system understands this roots structure, and will build a course package for each root.

**Note:** If you're using a local development environment, make the top level *course.xml* point to the desired root, and check out the repo multiple times if you need multiple runs simultaneously).

That's basically all there is to the organizational structure. Read the next section for details on the tags we support, including some special case tags like *customtag* and *html* invariants, and look at the end for some tips that will make the editing process easier.

### 1.1.4 Tags

| | |
|---|---|
| *abtest* | Support for A/B testing. TODO: add details.. |
| *chapter* | Top level organization unit of a course. The courseware display code currently expects the top level *course* element to contain only chapters, though there is no philosophical reason why this is required, so we may change it to properly display non-chapters at the top level. |
| *conditional* | Conditional element, which shows one or more modules only if certain conditions are satisfied. |
| *course* | Top level tag. Contains everything else. |
| *customtag* | Render an html template, filling in some parameters, and return the resulting html. See below for details. |
| *discussion* | Inline discussion forum. |
| *html* | A reference to an html file. |
| *error* | Don't put these in by hand :) The internal representation of content that has an error, such as malformed XML or some broken invariant. |
| *problem* | See elsewhere in edx4edx for documentation on the format. |
| *problemset* | Logically, a series of related problems. Currently displayed vertically. May contain explanatory html, videos, etc. |
| *sequential* | A sequence of content, currently displayed with a horizontal list of tabs. If possible, use a more semantically meaningful tag (currently, we only have *videosequence*). |
| *vertical* | A sequence of content, displayed vertically. Content will be accessed all at once, on the right part of the page. No navigational bar. May have to use browser scroll bars. Content split with separators. If possible, use a more semantically meaningful tag (currently, we only have *problemset*). |
| *video* | A link to a video, currently expected to be hosted on youtube. |
| *videosequence* | A sequence of videos. This can contain various non-video content; it just signals to the system that this is logically part of an explanatory sequence of content, as opposed to say an exam sequence. |

#### Container Tags

Container tags include *chapter*, *sequential*, *videosequence*, *vertical*, and *problemset*. They are all specified in the same way in the xml, as shown in the tutorial above.

#### *course*

*course* is also a container, and is similar, with one extra wrinkle: the top level pointer tag *must* have *org* and *course* attributes specified–the organization name, and course name. Note that *course* is referring to the platonic ideal of this course (e.g. "6.002x"), not to any particular run of this course. The *url_name* should be the particular run of this course.

#### *conditional*

*conditional* is as special kind of container tag as well. Here are two examples:

```
<conditional condition="require_completed" required="problem/choiceprob">
  <video url_name="secret_video" />
</conditional>

<conditional condition="require_attempted" required="problem/choiceprob&problem/sumprob">
```

```
    <html url_name="secret_page" />
</conditional>
```

The condition can be either *require_completed*, in which case the required modules must be completed, or *require_attempted*, in which case the required modules must have been attempted.

The required modules are specified as a set of *tag/url_name*, joined by an ampersand.

### *customtag*

When we see:

```
<customtag impl="special" animal="unicorn" hat="blue"/>
```

We will:

1. Look for a file called *custom_tags/special* in your course dir.
2. Render it as a mako template, passing parameters {'animal':'unicorn', 'hat':'blue'}, generating html. (Google *mako* for template syntax, or look at existing examples).

Since *customtag* is already a pointer, there is generally no need to put it into a separate file–just use it in place:

```
<customtag url_name="my_custom_tag" impl="blah" attr1="..."/>
```

### *discussion*

The discussion tag embeds an inline discussion module. The XML format is:

```
<discussion for="Course overview" id="6002x_Fall_2012_Overview" discussion_category="Week 1/Over
```

The meaning of each attribute is as follows:

| | |
|---|---|
| *for* | A string that describes the discussion. Purely for descriptive purposes (to the student). |
| *id* | The identifier that the discussion forum service uses to refer to this inline discussion module. Since the *id* must be unique and lives in a common namespace with all other courses, the preferred convention is to use *<course_name>_<course_run>_<descriptor>* as in the above example. The *id* should be "machine-friendly", e.g. use alphanumeric characters, underscores. Do **not** use a period (e.g. *6.002x_Fall_2012_Overview*). |
| *discussion_category* | The inline module will be indexed in the main "Discussion" tab of the course. The inline discussions are organized into a directory-like hierarchy. Note that the forward slash indicates depth, as in conventional filesytems. In the above example, this discussion module will show up in the following "directory": *Week 1/Overview/Course overview* |

Note that the *for* tag has been appended to the end of the *discussion_category*. This can often lead into deeply nested subforums, which may not be intended. In the above example, if we were to use instead:

```
<discussion for="Course overview" id="6002x_Fall_2012_Overview" discussion_category="Week 1" />
```

This discussion module would show up in the main forums as *Week 1 / Course overview*, which is more succinct.

### *html*

Most of our content is in XML, but some HTML content may not be proper xml (all tags matched, single top-level tag, etc), since browsers are fairly lenient in what they'll display. So, there are two ways to include HTML content:

- If your HTML content is in a proper XML format, just put it in *html/{url_name}.xml*.
- If your HTML content is not in proper XML format, you can put it in *html/{filename}.html*, and put *<html filename={filename} />* in *html/{filename}.xml*. This allows another level of indirection, and makes sure that we can read the XML file and then just return the actual HTML content without trying to parse it.

#### *video*

Videos have an attribute *youtube*, which specifies a series of speeds + youtube video IDs:

```
<video youtube="0.75:1yk1A8-FPbw,1.0:vNMrbPvwhU4,1.25:gBW_wqe7rDc,1.50:7AE_TKgaBwA"
       url_name="S15V14_Response_to_impulse_limit_case"/>
```

This video has been encoded at 4 different speeds: *0.75x*, *1x*, *1.25x*, and *1.5x*.

### More on *url_name*

Every content element (within a course) should have a unique id. This id is formed as *{category}/{url_name}*, or automatically generated from the content if *url_name* is not specified. Categories are the different tag types ('chapter', 'problem', 'html', 'sequential', etc). Url_name is a string containing a-z, A-Z, dot (.), underscore (_), and ':'. This is what appears in urls that point to this object.

Colon (':') is special–when looking for the content definition in an xml, ':' will be replaced with '/'. This allows organizing content into folders. For example, given the pointer tag

```
<problem url_name="conceptual:add_apples_and_oranges"/>
```

we would look for the problem definition in *problem/conceptual/add_apples_and_oranges.xml*. (There is a technical reason why we can't just allow '/' in the url_name directly.)

**Important:** A student's state for a particular content element is tied to the element ID, so automatic ID generation is only ok for elements that do not need to store any student state (e.g. verticals or customtags). For problems, sequentials, and videos, and any other element where we keep track of what the student has done and where they are at, you should specify a unique *url_name*. Of course, any content element that is split out into a file will need a *url_name* to specify where to find the definition.

## 1.1.5 Policy Files

- A policy file is useful when running different versions of a course e.g. internal, external, fall, spring, etc. as you can change due dates, etc, by creating multiple policy files.
- A policy file provides information on the metadata of the course–things that are not inherent to the definitions of the contents, but that may vary from run to run.
- Note: We will be expanding our understanding and format for metadata in the not-too-distant future, but for now it is simply a set of key-value pairs.

### Locations

- The policy for a course run *some_url_name* should live in *policies/some_url_name/policy.json* (NOTE: the old format of putting it in *policies/some_url_name.json* will also work, but we suggest using the subdirectory to have all the per-course policy files in one place)

- Grading policy files go in *policies/some_url_name/grading_policy.json* (if there's only one course run, can also put it directly in the course root: */grading_policy.json*)

### Contents

- The file format is JSON, and is best shown by example, as in the tutorial above.
- The expected contents are a dictionary mapping from keys to values (syntax *{ key : value, key2 : value2, etc}*)
- Keys are in the form *{category}/{url_name}*, which should uniquely identify a content element. Values are dictionaries of the form *{"metadata-key" : "metadata-value"}*.
- The order in which things appear does not matter, though it may be helpful to organize the file in the same order as things appear in the content.
- NOTE: JSON is picky about commas. If you have trailing commas before closing braces, it will complain and refuse to parse the file. This can be irritating at first.

## Supported fields at the course level

| | |
|---|---|
| *start* | specify the start date for the course. Format-by-example: *"2012-09-05T12:00"*. |
| *advertised_start* | specify what you want displayed as the start date of the course in the course listing and course about pages. This can be useful if you want to let people in early before the formal start. Format-by-example: *"2012-09-05T12:"00*. |
| *disable_policy_graph* | set to true (or "Yes"), if the policy graph should be disabled (ie not shown). |
| *enrollment_start*, *enrollment_end* | – when can students enroll? (if not specified, can enroll anytime). Same format as *start*. |
| *end* | specify the end date for the course. Format-by-example: *"2012-11-05T12:00"*. |
| *end_of_course_survey_url* | a url for an end of course survey – shown after course is over, next to certificate download links. |
| *tabs* | have custom tabs in the courseware. See below for details on config. |
| *discussion_blackouts* | An array of time intervals during which you want to disable a student's ability to create or edit posts in the forum. Moderators, Community TAs, and Admins are unaffected. You might use this during exam periods, but please be aware that the forum is often a very good place to catch mistakes and clarify points to students. The better long term solution would be to have better flagging/moderation mechanisms, but this is the hammer we have today. Format by example: *[["2012-10-29T04:00", "2012-11-03T04:00"], ["2012-12-30T04:00", "2013-01-02T04:00"]]* |
| *show_calculator* | (value "Yes" if desired) |
| *days_early_for_beta* | number of days (floating point ok) early that students in the beta-testers group get to see course content. Can also be specified for any other course element, and overrides values set at higher levels. |
| *cohort_config* | • *cohorted* : boolean. Set to true if this course uses student cohorts. If so, all inline discussions are automatically cohorted, and top-level discussion topics are configurable via the cohorted_discussions list. Default is not cohorted).<br>• *cohorted_discussions*: list of discussions that should be cohorted. Any not specified in this list are not cohorted.<br>• *auto_cohort*: Truthy.<br>• *auto_cohort_groups*: *["group name 1", "group name 2", ...]* If *cohorted* and *auto_cohort* is true, automatically put each student into a random group from the *auto_cohort_groups* list, creating the group if needed. |
| *pdf_textbooks* | have pdf-based textbooks on tabs in the courseware. See below for details on config. |
| *html_textbooks* | have html-based textbooks on tabs in the courseware. See below for details on config. |

### Available metadata

#### Not Inherited

***display_name*** Name that will appear when this content is displayed in the courseware. Useful for all tag types.

***format*** Subheading under display name – currently only displayed for chapter sub-sections. Also used by the the grader to know how to process students assessments that the section contains. New formats can be defined as a 'type' in the GRADER variable in course_settings.json. Optional. (TODO: double check this–what's the current behavior?)

***hide_from_toc*** If set to true for a chapter or chapter subsection, will hide that element from the courseware navigation accordion. This is useful if you'd like to link to the content directly instead (e.g. for tutorials)

***ispublic*** Specify whether the course is public. You should be able to use start dates instead (?)

#### Inherited

***start*** When this content should be shown to students. Note that anyone with staff access to the course will always see everything.

***showanswer***

**When to show answer. Values: never, attempted, answered, closed, finished, past_due, always. Default: closed. Optional.**

- *never*: never show answer
- *attempted*: show answer after first attempt
- *answered* : this is slightly different from *attempted* – resetting the problems makes "done" False, but leaves attempts unchanged.
- *closed* : show answer after problem is closed, ie due date is past, or maximum attempts exceeded.
- *finished* : show answer after problem closed, or is correctly answered.
- *past_due* : show answer after problem due date is past.
- *always* : always allow answer to be shown.

***graded*** Whether this section will count towards the students grade. "true" or "false". Defaults to "false".

***rerandomize*** Randomize question on each attempt. Optional. Possible values:

***always* (default)** Students see a different version of the problem after each attempt to solve it.

***onreset*** Randomize question when reset button is pressed by the student.

***never*** All students see the same version of the problem.

***per_student*** Individual students see the same version of the problem each time the look at it, but that version is different from what other students see.

***due*** Due date for assignment. Assignment will be closed after that. Values: valid date. Default: none. Optional.

***attempts*** Number of allowed attempts. Values: integer. Default: infinite. Optional.

***graceperiod*** A default length of time that the problem is still accessible after the due date in the format *"2 days 3 hours"* or *"1 day 15 minutes"*. Note, graceperiods are currently the easiest way to handle time zones. Due dates are all expressed in UTC.

***xqa_key*** For integration with Ike's content QA server. – should typically be specified at the course level.

### Inheritance example

This is a sketch (“tue” is not a valid start date), that should help illustrate how metadata inheritance works.

```
<course start="tue">
  <chap1> -- start tue
    <problem>   --- start tue
  </chap1>
  <chap2 start="wed">  -- start wed
   <problem2 start="thu">  -- start thu
   <problem3>      -- start wed
  </chap2>
</course>
```

### Specifying metadata in the XML file

Metadata can also live in the xml files, but anything defined in the policy file overrides anything in the XML. This is primarily for backwards compatibility, and you should probably not use both. If you do leave some metadata tags in the xml, you should be consistent (e.g. if *display_name* stays in XML, they should all stay in XML. Note *display_name* should be specified in the problem xml definition itself, ie, *<problem display_name=”Title”>Problem Text</problem>*, in file *ProblemFoo.xml*).

**Note:** Some xml attributes are not metadata. e.g. in *<video youtube=”xyz987293487293847”/>*, the *youtube* attribute specifies what video this is, and is logically part of the content, not the policy, so it should stay in the xml.

Another example policy file:

```
{
    "course/2012": {
        "graceperiod": "1 day",
        "start": "2012-10-15T12:00",
        "display_name": "Introduction to Computer Science I",
        "xqa_key": "z1y4vdYcy0izkoPeihtPClDxmbY1ogDK"
    },
    "chapter/Week_0": {
        "display_name": "Week 0"
    },
    "sequential/Pre-Course_Survey": {
        "display_name": "Pre-Course Survey",
        "format": "Survey"
    }
}
```

### Deprecated Formats

If you look at some older xml, you may see some tags or metadata attributes that aren’t listed above. They are deprecated, and should not be used in new content. We include them here so that you can understand how old-format content works.

#### Obsolete Tags

***section*** This used to be necessary within chapters. Now, you can just use any standard tag inside a chapter, so use the container tag that makes the most sense for grouping content–e.g. *problemset*, *videosequence*, and just include content directly if it belongs inside a chapter (e.g. *html*, *video*, *problem*)

***videodev, book, slides, image, discuss*** There used to be special purpose tags that all basically did the same thing, and have been subsumed by *customtag*. The list is *videodev, book, slides, image, discuss*. Use *customtag* in new content. (e.g. instead of <*book page="12"/*>, use <*customtag impl="book" page="12"/*>)

**Obsolete Attributes**

***slug*** Old term for *url_name*. Use *url_name*

***name*** We didn't originally have a distinction between *url_name* and *display_name* – this made content element ids fragile, so please use *url_name* as a stable unique identifier for the content, and *display_name* as the particular string you'd like to display for it.

### 1.1.6 Static links

If your content links (e.g. in an html file) to *"static/blah/ponies.jpg"*, we will look for this...

- If your course dir has a *static/* subdirectory, we will look in *YOUR_COURSE_DIR/static/blah/ponies.jpg*. This is the prefered organization, as it does not expose anything except what's in *static/* to the world.
- If your course dir does not have a *static/* subdirectory, we will look in *YOUR_COURSE_DIR/blah/ponies.jpg*. This is the old organization, and requires that the web server allow access to everything in the couse dir. To switch to the new organization, move all your static content into a new *static/* dir (e.g. if you currently have things in *images/*, *css/*, and *special/*, create a dir called *static/*, and move *images/, css/, and special/* there).

Links that include */course* will be rewritten to the root of your course in the courseware (e.g. *courses/{org}/{course}/{url_name}/* in the current url structure). This is useful for linking to the course wiki, for example.

### 1.1.7 Tabs

If you want to customize the courseware tabs displayed for your course, specify a "tabs" list in the course-level policy, like the following example:

```
"tabs" : [
  {"type": "courseware"},
  {
    "type": "course_info",
    "name": "Course Info"
  },
  {
    "type": "external_link",
    "name": "My Discussion",
    "link": "http://www.mydiscussion.org/blah"
  },
  {"type": "progress", "name": "Progress"},
  {"type": "wiki", "name": "Wonderwiki"},
  {
    "type": "static_tab",
    "url_slug": "news",
    "name": "Exciting news"
  },
  {"type": "textbooks"},
  {"type": "html_textbooks"},
  {"type": "pdf_textbooks"}
]
```

- If you specify any tabs, you must specify all tabs. They will appear in the order given.
- The first two tabs must have types *"courseware"* and *"course_info"*, in that order, or the course will not load.
- The *courseware* tab never has a name attribute – it's always rendered as "Courseware" for consistency between courses.
- The *textbooks* tab will actually generate one tab per textbook, using the textbook titles as names.
- The *html_textbooks* tab will actually generate one tab per html_textbook. The tab name is found in the html textbook definition.
- The *pdf_textbooks* tab will actually generate one tab per pdf_textbook. The tab name is found in the pdf textbook definition.
- For static tabs, the *url_slug* will be the url that points to the tab. It can not be one of the existing courseware url types (even if those aren't used in your course). The static content will come from *tabs/{course_url_name}/{url_slug}.html*, or *tabs/{url_slug}.html* if that doesn't exist.
- An Instructor tab will be automatically added at the end for course staff users.

Table 1.1: Supported Tabs and Parameters

| | |
|---|---|
| *courseware* | No other parameters. |
| *course_info* | Parameter *name*. |
| *wiki* | Parameter *name*. |
| *discussion* | Parameter *name*. |
| *external_link* | Parameters *name*, *link*. |
| *textbooks* | No parameters–generates tab names from book titles. |
| *html_textbooks* | No parameters–generates tab names from html book definition. (See discussion below for configuration.) |
| *pdf_textbooks* | No parameters–generates tab names from pdf book definition. (See discussion below for configuration.) |
| *progress* | Parameter *name*. |
| *static_tab* | Parameters *name*, *url_slug*–will look for tab contents in 'tabs/{course_url_name}/{tab url_slug}.html' |
| *staff_grading* | No parameters. If specified, displays the staff grading tab for instructors. |

## 1.1.8 Textbooks

Support is currently provided for image-based, HTML-based and PDF-based textbooks. In addition to enabling the display of textbooks in tabs (see above), specific information about the location of textbook content must be configured.

### Image-based Textbooks

#### Configuration

Image-based textbooks are configured at the course level in the XML markup. Here is an example:

```
<course>
  <textbook title="Textbook 1" book_url="https://www.example.com/textbook_1/" />
  <textbook title="Textbook 2" book_url="https://www.example.com/textbook_2/" />
  <chapter url_name="Overview">
  <chapter url_name="First week">
</course>
```

Each *textbook* element is displayed on a different tab. The *title* attribute is used as the tab's name, and the *book_url* attribute points to the remote directory that contains the images of the text. Note the trailing slash on the end of the *book_url* attribute.

The images must be stored in the same directory as the *book_url*, with filenames matching *pXXX.png*, where *XXX* is a three-digit number representing the page number (with leading zeroes as necessary). Pages start at *p001.png*.

Each textbook must also have its own table of contents. This is read from the *book_url* location, by appending *toc.xml*. This file contains a *table_of_contents* parent element, with *entry* elements nested below it. Each *entry* has attributes for *name*, *page_label*, and *page*, as well as an optional *chapter* attribute. An arbitrary number of levels of nesting of *entry* elements within other *entry* elements is supported, but you're likely to only want two levels. The *page* represents the actual page to link to, while the *page_label* matches the displayed page number on that page. Here's an example:

```
<table_of_contents>
  <entry page="1" page_label="i" name="Title" />
  <entry page="2" page_label="ii" name="Preamble">
    <entry page="2" page_label="ii" name="Copyright"/>
    <entry page="3" page_label="iii" name="Brief Contents"/>
    <entry page="5" page_label="v" name="Contents"/>
    <entry page="9" page_label="1" name="About the Authors"/>
    <entry page="10" page_label="2" name="Acknowledgments"/>
    <entry page="11" page_label="3" name="Dedication"/>
    <entry page="12" page_label="4" name="Preface"/>
  </entry>
  <entry page="15" page_label="7" name="Introduction to edX" chapter="1">
    <entry page="15" page_label="7" name="edX in the Modern World"/>
    <entry page="18" page_label="10" name="The edX Method"/>
    <entry page="18" page_label="10" name="A Description of edX"/>
    <entry page="29" page_label="21" name="A Brief History of edX"/>
    <entry page="51" page_label="43" name="Introduction to edX"/>
    <entry page="56" page_label="48" name="Endnotes"/>
  </entry>
  <entry page="73" page_label="65" name="Art and Photo Credits" chapter="30">
    <entry page="73" page_label="65" name="Molecular Models"/>
    <entry page="73" page_label="65" name="Photo Credits"/>
  </entry>
  <entry page="77" page_label="69" name="Index" />
</table_of_contents>
```

### Linking from Content

It is possible to add links to specific pages in a textbook by using a URL that encodes the index of the textbook and the page number. The URL is of the form */course/book/${bookindex}/$page}*. If the page is omitted from the URL, the first page is assumed.

You can use a *customtag* to create a template for such links. For example, you can create a *book* template in the *customtag* directory, containing:

```
<img src="/static/images/icons/textbook_icon.png"/> More information given in <a href="/course/book/
```

The course content can then link to page 25 using the *customtag* element:

```
<customtag book="0" page="25" impl="book"/>
```

### HTML-based Textbooks

#### Configuration

HTML-based textbooks are configured at the course level in the policy file. The JSON markup consists of an array of maps, with each map corresponding to a separate textbook. There are two styles to presenting HTML-based material. The first way is as a single HTML on a tab, which requires only a tab title and a URL for configuration. A second way permits the display of multiple HTML files that should be displayed together on a single view. For this view, a side panel of links is available on the left, allowing selection of a particular HTML to view.

```
"html_textbooks": [
  {"tab_title": "Textbook 1",
   "url": "https://www.example.com/thiscourse/book1/book1.html" },
  {"tab_title": "Textbook 2",
   "chapters": [
       { "title": "Chapter 1", "url": "https://www.example.com/thiscourse/book2/Chapter1.html" },
       { "title": "Chapter 2", "url": "https://www.example.com/thiscourse/book2/Chapter2.html" },
       { "title": "Chapter 3", "url": "https://www.example.com/thiscourse/book2/Chapter3.html" },
       { "title": "Chapter 4", "url": "https://www.example.com/thiscourse/book2/Chapter4.html" },
       { "title": "Chapter 5", "url": "https://www.example.com/thiscourse/book2/Chapter5.html" },
       { "title": "Chapter 6", "url": "https://www.example.com/thiscourse/book2/Chapter6.html" },
       { "title": "Chapter 7", "url": "https://www.example.com/thiscourse/book2/Chapter7.html" }
       ]
  }
]
```

Some notes:

- It is not a good idea to include a top-level URL and chapter-level URLs in the same textbook configuration.

#### Linking from Content

It is possible to add links to specific pages in a textbook by using a URL that encodes the index of the textbook, the chapter (if chapters are used), and the page number. For a book with no chapters, the URL is of the form */course/htmlbook/${bookindex}*. For a book with chapters, use */course/htmlbook/${bookindex}/chapter/${chapter}* for a specific chapter, or */course/htmlbook/${bookindex}* will default to the first chapter.

For example, for the book with no chapters configured above, the textbook can be reached using the URL */course/htmlbook/0*. Reaching the third chapter of the second book is accomplished with */course/htmlbook/1/chapter/3*.

You can use a *customtag* to create a template for such links. For example, you can create a *htmlbook* template in the *customtag* directory, containing:

```
<img src="/static/images/icons/textbook_icon.png"/> More information given in <a href="/course/htmlbo
```

And a *htmlchapter* template containing:

```
<img src="/static/images/icons/textbook_icon.png"/> More information given in <a href="/course/htmlbo
```

The example pages can then be linked using the *customtag* element:

```
<customtag book="0" impl="htmlbook"/>
<customtag book="1" chapter="3" impl="htmlchapter"/>
```

## PDF-based Textbooks

### Configuration

PDF-based textbooks are configured at the course level in the policy file. The JSON markup consists of an array of maps, with each map corresponding to a separate textbook. There are two styles to presenting PDF-based material. The first way is as a single PDF on a tab, which requires only a tab title and a URL for configuration. A second way permits the display of multiple PDFs that should be displayed together on a single view. For this view, a side panel of links is available on the left, allowing selection of a particular PDF to view.

```
"pdf_textbooks": [
  {"tab_title": "Textbook 1",
   "url": "https://www.example.com/thiscourse/book1/book1.pdf" },
  {"tab_title": "Textbook 2",
   "chapters": [
       { "title": "Chapter 1", "url": "https://www.example.com/thiscourse/book2/Chapter1.pdf" },
       { "title": "Chapter 2", "url": "https://www.example.com/thiscourse/book2/Chapter2.pdf" },
       { "title": "Chapter 3", "url": "https://www.example.com/thiscourse/book2/Chapter3.pdf" },
       { "title": "Chapter 4", "url": "https://www.example.com/thiscourse/book2/Chapter4.pdf" },
       { "title": "Chapter 5", "url": "https://www.example.com/thiscourse/book2/Chapter5.pdf" },
       { "title": "Chapter 6", "url": "https://www.example.com/thiscourse/book2/Chapter6.pdf" },
       { "title": "Chapter 7", "url": "https://www.example.com/thiscourse/book2/Chapter7.pdf" }
       ]
  }
]
```

Some notes:

- It is not a good idea to include a top-level URL and chapter-level URLs in the same textbook configuration.

### Linking from Content

It is possible to add links to specific pages in a textbook by using a URL that encodes the index of the textbook, the chapter (if chapters are used), and the page number. For a book with no chapters, the URL is of the form */course/pdfbook/${bookindex}/$page}*. For a book with chapters, use */course/pdfbook/${bookindex}/chapter/${chapter}/${page}*. If the page is omitted from the URL, the first page is assumed.

For example, for the book with no chapters configured above, page 25 can be reached using the URL */course/pdfbook/0/25*. Reaching page 19 in the third chapter of the second book is accomplished with */course/pdfbook/1/chapter/3/19*.

You can use a *customtag* to create a template for such links. For example, you can create a *pdfbook* template in the *customtag* directory, containing:

```
<img src="/static/images/icons/textbook_icon.png"/> More information given in <a href="/course/pdfboo
```

And a *pdfchapter* template containing:

```
<img src="/static/images/icons/textbook_icon.png"/> More information given in <a href="/course/pdfboo
```

The example pages can then be linked using the *customtag* element:

```
<customtag book="0" page="25" impl="pdfbook"/>
<customtag book="1" chapter="3" page="19" impl="pdfchapter"/>
```

### 1.1.9 Other file locations (info and about)

With different course runs, we may want different course info and about materials. This is now supported by putting files in as follows:

```
/
  about/
       foo.html      -- shared default for all runs
       url_name1/
            foo.html   -- version used for url_name1
            bar.html   -- bar for url_name1
       url_name2/
            bar.html   -- bar for url_name2
                       -- url_name2 will use default foo.html
```

and the same works for the *info* directory.

### 1.1.10 Tips for content developers

1. We will be making better tools for managing policy files soon. In the meantime, you can add dummy definitions to make it easier to search and separate the file visually. For example, you could add *"WEEK 1" : "###################"*, before the week 1 material to make it easy to find in the file.
2. Come up with a consistent pattern for url_names, so that it's easy to know where to look for any piece of content. It will also help to come up with a standard way of splitting your content files. As a point of departure, we suggest splitting chapters, sequences, html, and problems into separate files.
3. Prefer the most "semantic" name for containers: e.g., use problemset rather than sequential for a problem set. That way, if we decide to display problem sets differently, we don't have to change the XML.

## 1.2 Course Grading

This document is written to help professors understand how a final grade for a course is computed.

Course grading is the process of taking all of the problems scores for a student in a course and generating a final score (and corresponding letter grade). This grading process can be split into two phases - totaling sections and section weighting.

### 1.2.1 Totaling sections

The process of totaling sections is to get a percentage score (between 0.0 and 1.0) for every section in the course. A section is any module that is a direct child of a chapter. For example, psets, labs, and sequences are all common sections. Only the *percentage* on the section will be available to compute the final grade, *not* the final number of points earned / possible.

**Important:** For a section to be included in the final grade, the policies file must set *graded = True* for the section.

For each section, the grading function retrieves all problems within the section. The section percentage is computed as (total points earned) / (total points possible).

### 1.2.2 Weighting Problems

In some cases, one might want to give weights to problems within a section. For example, a final exam might contain four questions each worth 1 point by default. This means each question would by default have the same weight. If one wanted the first problem to be worth 50% of the final exam, the policy file could specify weights of 30, 10, 10, and 10 to the four problems, respectively.

Note that the default weight of a problem **is not 1**. The default weight of a problem is the module's *max_grade*.

If weighting is set, each problem is worth the number of points assigned, regardless of the number of responses it contains.

Consider a Homework section that contains two problems.

```
<problem display_name="Problem 1">
  <numericalresponse> ... </numericalreponse>
</problem>

<problem display_name="Problem 2">
  <numericalresponse> ... </numericalreponse>
  <numericalresponse> ... </numericalreponse>
  <numericalresponse> ... </numericalreponse>
</problem>
```

Without weighting, Problem 1 is worth 25% of the assignment, and Problem 2 is worth 75% of the assignment.

Weighting for the problems can be set in the policy.json file.

```
"problem/problem1": {
  "weight": 2
},
"problem/problem2": {
  "weight": 2
},
```

With the above weighting, Problems 1 and 2 are each worth 50% of the assignment.

Please note: When problems have weight, the point value is automatically included in the display name *except* when *"weight": 1*. When the weight is 1, no visual change occurs in the display name, leaving the point value open to interpretation to the student.

### 1.2.3 Weighting Sections

Once each section has a percentage score, we must total those sections into a final grade. Of course, not every section has equal weight in the final grade. The policies for weighting sections into a final grade are specified in the grading_policy.json file.

The *grading_policy.json* file specifies several sub-graders that are each given a weight and factored into the final grade. There are currently two types of sub-graders, section format graders and single section graders.

We will use this simple example of a grader with one section format grader and one single section grader.

```
"GRADER" : [
    {
      "type" : "Homework",
      "min_count" : 12,
      "drop_count" : 2,
      "short_label" : "HW",
      "weight" : 0.4
    },
    {
      "type" : "Final",
      "name" : "Final Exam",
      "short_label" : "Final",
      "weight" : 0.6
    }
]
```

### Section Format Graders

A section format grader grades a set of sections with the same format, as defined in the course policy file. To make a vertical named Homework1 be graded by the Homework section format grader, the following definition would be in the course policy file.

```
"vertical/Homework1": {
    "display_name": "Homework 1",
    "graded": true,
    "format": "Homework"
},
```

In the example above, the section format grader declares that it will expect to find at least 12 sections with the format "Homework". It will drop the lowest 2. All of the homework assignments will have equal weight, relative to each other (except, of course, for the assignments that are dropped).

This format supports forecasting the number of homework assignments. For example, if the course only has 3 homeworks written, but the section format grader has been told to expect 12, the missing 9 will have an assumed 0% and will still show up in the grade breakdown.

A section format grader will also show the average of that section in the grade breakdown (shown on the Progress page, gradebook, etc.).

### Single Section Graders

A single section grader grades exactly that - a single section. If a section is found with a matching format and display name then the score of that section is used. If not, a score of 0% is assumed.

### Combining sub-graders

The final grade is computed by taking the score and weight of each sub grader. In the above example, homework will be 40% of the final grade. The final exam will be 60% of the final grade.

## 1.2.4 Displaying the final grade

The final grade is then rounded up to the nearest percentage point. This is so the system can consistently display a percentage without worrying whether the displayed percentage has been rounded up or down (potentially misleading the student). The formula for the rounding is:

```
rounded_percent = round(computed_percent * 100 + 0.05) / 100
```

The grading policy file also specifies the cutoffs for the grade levels. A grade is either A, B, or C. If the student does not reach the cutoff threshold for a C grade then the student has not earned a grade and will not be eligible for a certificate. Letter grades are only awarded to students who have completed the course. There is no notion of a failing letter grade.

## 1.3 Specific Problem Types

### 1.3.1 XML format of drag and drop input [inputtypes]

#### Format description

The main tag of Drag and Drop (DnD) input is:

```
<drag_and_drop_input> ... </drag_and_drop_input>
```

`drag_and_drop_input` can include any number of the following 2 tags: `draggable` and `target`.

##### drag_and_drop_input tag

The main container for a single instance of DnD. The following attributes can be specified for this tag:

```
img - Relative path to an image that will be the base image. All draggables
      can be dragged onto it.
target_outline - Specify whether an outline (gray dashed line) should be
      drawn around targets (if they are specified). It can be either
      'true' or 'false'. If not specified, the default value is
      'false'.
one_per_target - Specify whether to allow more than one draggable to be
      placed onto a single target. It can be either 'true' or 'false'. If
      not specified, the default value is 'true'.
no_labels - default is false, in default behaviour if label is not set, label
      is obtained from id. If no_labels is true, labels are not automatically
      populated from id, and one can not set labels and obtain only icons.
```

##### draggable tag

Draggable tag specifies a single draggable object which has the following attributes:

```
id - Unique identifier of the draggable object.
label - Human readable label that will be shown to the user.
icon - Relative path to an image that will be shown to the user.
can_reuse - true or false, default is false. If true, same draggable can be
used multiple times.
```

A draggable is what the user must drag out of the slider and place onto the base image. After a drag operation, if the center of the draggable ends up outside the rectangular dimensions of the image, it will be returned back to the slider.

In order for the grader to work, it is essential that a unique ID is provided. Otherwise, there will be no way to tell which draggable is at what coordinate, or over what target. Label and icon attributes are optional. If they are provided they will be used, otherwise, you can have an empty draggable. The path is relative to 'course_folder' folder, for example, /static/images/img1.png.

##### target tag

Target tag specifies a single target object which has the following required attributes:

```
id - Unique identifier of the target object.
x - X-coordinate on the base image where the top left corner of the target
    will be positioned.
y - Y-coordinate on the base image where the top left corner of the target
    will be positioned.
w - Width of the target.
h - Height of the target.
```

A target specifies a place on the base image where a draggable can be positioned. By design, if the center of a draggable lies within the target (i.e. in the rectangle defined by [[x, y], [x + w, y + h]], then it is within the target. Otherwise, it is outside.

If at lest one target is provided, the behavior of the client side logic changes. If a draggable is not dragged on to a target, it is returned back to the slider.

If no targets are provided, then a draggable can be dragged and placed anywhere on the base image.

#### Targets on draggables

Sometimes it is not enough to have targets only on the base image, and all of the draggables on these targets. If a complex problem exists where a draggable must become itself a target (or many targets), then the following extended syntax can be used:

```
<draggable {attribute list}>
    <target {attribute list} />
    <target {attribute list} />
    <target {attribute list} />
    ...
</draggable>
```

The attribute list in the tags above ('draggable' and 'target') is the same as for normal 'draggable' and 'target' tags. The only difference is when you will be specifying inner target position coordinates. Using the 'x' and 'y' attributes you are setting the offset of the inner target from the upper-left corner of the parent draggable (that contains the inner target).

#### Limitations of targets on draggables

1.) Currently there is a limitation to the level of nesting of targets.

Even though you can pile up a large number of draggables on targets that themselves are on draggables, the Drag and Drop instance will be graded only in the case if there is a maximum of two levels of targets. The first level are the "base" targets. They are attached to the base image. The second level are the targets defined on draggables.

2.) Another limitation is that the target bounds are not checked against other targets.

For now, it is the responsibility of the person who is constructing the course material to make sure that there is no overlapping of targets. It is also preferable that targets on draggables are smaller than the actual parent draggable. Technically this is not necessary, but from the usability perspective it is desirable.

3.) You can have targets on draggables only in the case when there are base targets defined (base targets are attached to the base image).

If you do not have base targets, then you can only have a single level of nesting (draggables on the base image). In this case the client side will be reporting (x,y) positions of each draggables on the base image.

### Correct answer format

(NOTE: For specifying answers for targets on draggables please see next section.)

There are two correct answer formats: short and long If short from correct answer is mapping of 'draggable_id' to 'target_id':

```
correct_answer = {'grass':     [[300, 200], 200], 'ant': [[500, 0], 200]}
correct_answer = {'name4': 't1', '7': 't2'}
```

In long form correct answer is list of dicts. Every dict has 3 keys: draggables, targets and rule. For example:

```
correct_answer = [
{
'draggables':   ['7', '8'],
'targets':  ['t5_c', 't6_c'],
'rule': 'anyof'
},
{
'draggables': ['1', '2'],
'targets': ['t2_h', 't3_h', 't4_h', 't7_h', 't8_h', 't10_h'],
'rule': 'anyof'
}]
```

Draggables is list of draggables id. Target is list of targets id, draggables must be dragged to with considering rule. Rule is string.

Draggables in dicts inside correct_answer list must not intersect!!!

Wrong (for draggable id 7):

```
correct_answer = [
{
'draggables':   ['7', '8'],
'targets':  ['t5_c', 't6_c'],
'rule': 'anyof'
},
{
'draggables': ['7', '2'],
'targets': ['t2_h', 't3_h', 't4_h', 't7_h', 't8_h', 't10_h'],
'rule': 'anyof'
}]
```

Rules are: exact, anyof, unordered_equal, anyof+number, unordered_equal+number

- Exact rule means that targets for draggable id's in user_answer are the same that targets from correct answer. For example, for draggables 7 and 8 user must drag 7 to target1 and 8 to target2 if correct_answer is:

  ```
  correct_answer = [
  {
  'draggables':   ['7', '8'],
  'targets':  ['tartget1', 'target2'],
  'rule': 'exact'
  }]
  ```

- unordered_equal rule allows draggables be dragged to targets unordered. If one want to allow for student to drag 7 to target1 or target2 and 8 to target2 or target 1 and 7 and 8 must be in different targets, then correct answer must be:

```
correct_answer = [
{
'draggables':   ['7', '8'],
'targets':  ['tartget1', 'target2'],
'rule': 'unordered_equal'
}]
```

- Anyof rule allows draggables to be dragged to any of targets. If one want to allow for student to drag 7 and 8 to target1 or target2, which means that if 7 is on target1 and 8 is on target1 or 7 on target2 and 8 on target2 or 7 on target1 and 8 on target2. Any of theese are correct which anyof rule:

```
correct_answer = [
{
'draggables':   ['7', '8'],
'targets':  ['tartget1', 'target2'],
'rule': 'anyof'
}]
```

- If you have can_reuse true, then you, for example, have draggables a,b,c and 10 targets. These will allow you to drag 4 'a' draggables to ['target1', 'target4', 'target7', 'target10'] , you do not need to write 'a' four times. Also this will allow you to drag 'b' draggable to target2 or target5 for target5 and target2 etc..:

```
correct_answer = [
    {
        'draggables': ['a'],
        'targets': ['target1',  'target4', 'target7', 'target10'],
        'rule': 'unordered_equal'
    },
    {
        'draggables': ['b'],
        'targets': ['target2', 'target5', 'target8'],
        'rule': 'anyof'
    },
    {
        'draggables': ['c'],
        'targets': ['target3', 'target6', 'target9'],
        'rule': 'unordered_equal'
    }]
```

- And sometimes you want to allow drag only two 'b' draggables, in these case you should use 'anyof+number' of 'unordered_equal+number' rule:

```
correct_answer = [
    {
        'draggables': ['a', 'a', 'a'],
        'targets': ['target1',  'target4', 'target7'],
        'rule': 'unordered_equal+numbers'
    },
    {
        'draggables': ['b', 'b'],
        'targets': ['target2', 'target5', 'target8'],
        'rule': 'anyof+numbers'
    },
    {
        'draggables': ['c'],
        'targets': ['target3', 'target6', 'target9'],
        'rule': 'unordered_equal'
    }]
```

In case if we have no multiple draggables per targets (one_per_target="true"), for same number of draggables, anyof is equal to unordered_equal

If we have can_reuse=true, than one must use only long form of correct answer.

### Answer format for targets on draggables

As with the cases described above, an answer must provide precise positioning for each draggable (on which targets it must reside). In the case when a draggable must be placed on a target that itself is on a draggable, then the answer must contain the chain of target-draggable-target. It is best to understand this on an example.

Suppose we have three draggables - 'up', 's', and 'p'. Draggables 's', and 'p' have targets on themselves. More specifically, 'p' has three targets - '1', '2', and '3'. The first requirement is that 's', and 'p' are positioned on specific targets on the base image. The second requirement is that draggable 'up' is positioned on specific targets of draggable 'p'. Below is an excerpt from a problem.:

```
<draggable id="up" icon="/static/images/images_list/lcao-mo/up.png" can_reuse="true" />

<draggable id="s" icon="/static/images/images_list/lcao-mo/orbital_single.png" label="s orbital" can_
    <target id="1" x="0" y="0" w="32" h="32"/>
</draggable>

<draggable id="p" icon="/static/images/images_list/lcao-mo/orbital_triple.png" can_reuse="true" label
    <target id="1" x="0" y="0" w="32" h="32"/>
    <target id="2" x="34" y="0" w="32" h="32"/>
    <target id="3" x="68" y="0" w="32" h="32"/>
</draggable>

...

correct_answer = [
    {
        'draggables': ['p'],
        'targets': ['p-left-target', 'p-right-target'],
        'rule': 'unordered_equal'
    },
    {
        'draggables': ['s'],
        'targets': ['s-left-target', 's-right-target'],
        'rule': 'unordered_equal'
    },
    {
        'draggables': ['up'],
        'targets': ['p-left-target[p][1]', 'p-left-target[p][2]', 'p-right-target[p][2]', 'p-right-ta
        'rule': 'unordered_equal'
    }
]
```

Note that it is a requirement to specify rules for all draggables, even if some draggable gets included in more than one chain.

### Grading logic

1. User answer (that comes from browser) and correct answer (from xml) are parsed to the same format:

```
    group_id: group_draggables, group_targets, group_rule
```

Group_id is ordinal number, for every dict in correct answer incremental group_id is assigned: 0, 1, 2, ...

Draggables from user answer are added to same group_id where identical draggables from correct answer are, for example:

```
If correct_draggables[group_0] = [t1, t2] then
user_draggables[group_0] are all draggables t1 and t2 from user answer:
[t1] or [t1, t2] or [t1, t2, t2] etc..
```

2. For every group from user answer, for that group draggables, if 'number' is in group rule, set() is applied, if 'number' is not in rule, set is not applied:

```
set() : [t1, t2, t3, t3] -> [t1, t2, ,t3]
```

For every group, at this step, draggables lists are equal.

3. For every group, lists of targets are compared using rule for that group.

**Set and '+number' cases** Set() and '+number' are needed only for case of reusable draggables, for other cases there are no equal draggables in list, so set() does nothing.

- Usage of set() operation allows easily create rule for case of "any number of same draggable can be dragged to some targets":

  ```
  {
          'draggables': ['draggable_1'],
          'targets': ['target3', 'target6', 'target9'],
          'rule': 'anyof'
  }
  ```

- 'number' rule is used for the case of reusable draggables, when one want to fix number of draggable to drag. In this example only two instances of draggables_1 are allowed to be dragged:

  ```
  {
          'draggables': ['draggable_1', 'draggable_1'],
          'targets': ['target3', 'target6', 'target9'],
          'rule': 'anyof+number'
  }
  ```

- Note, that in using rule 'exact', one does not need 'number', because you can't recognize from user interface which reusable draggable is on which target. Absurd example:

  ```
  {
          'draggables': ['draggable_1', 'draggable_1', 'draggable_2'],
          'targets': ['target3', 'target6', 'target9'],
          'rule': 'exact'
  }


  Correct handling of this example is to create different rules for draggable_1 and
  draggable_2
  ```

- For 'unordered_equal' (or 'exact' too) we don't need 'number' if you have only same draggable in group, as targets length will provide constraint for the number of draggables:

  ```
  {
          'draggables': ['draggable_1'],
          'targets': ['target3', 'target6', 'target9'],
  ```

```
        'rule': 'unordered_equal'
}
```

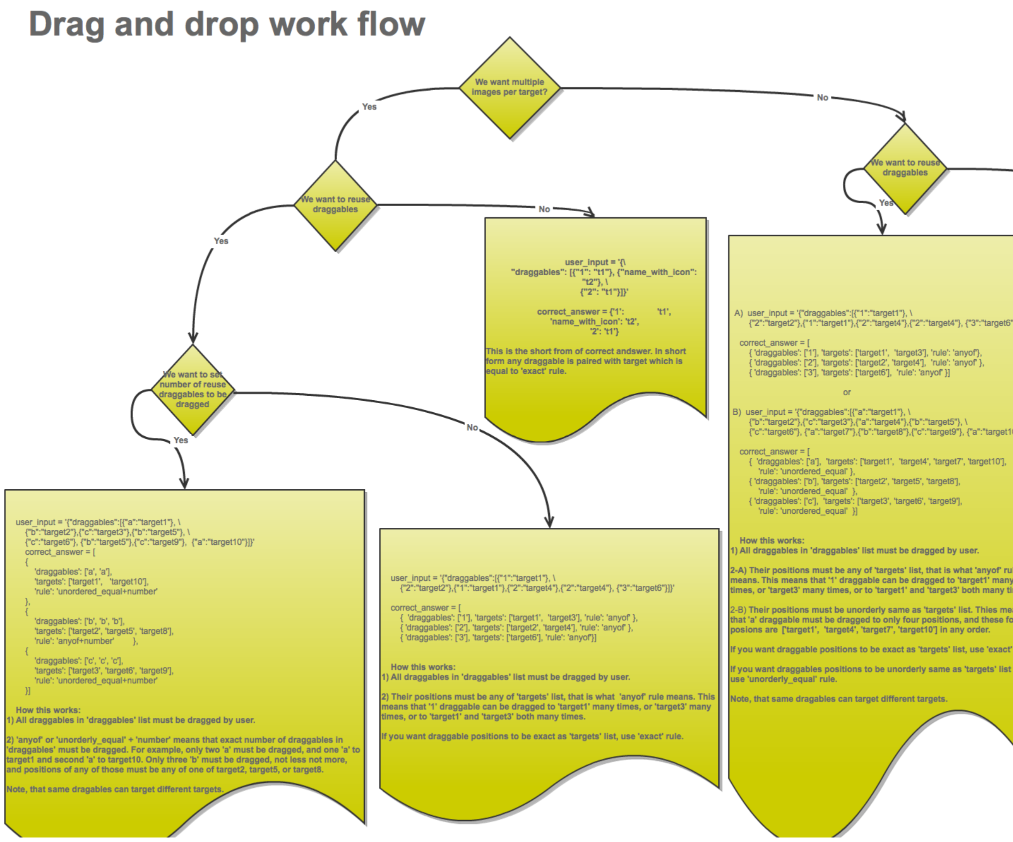
```
This means that only three draggaggables 'draggable_1' can be dragged.
```

- But if you have more that one different reusable draggable in list, you may use 'number' rule:

```
{
        'draggables': ['draggable_1', 'draggable_1', 'draggable_2'],
        'targets': ['target3', 'target6', 'target9'],
        'rule': 'unordered_equal+number'
}
```

```
If not use number, draggables list will be setted to  ['draggable_1', 'draggable_2']
```

### Logic flow


Drag and drop work flow
We want multiple images per target?
Yes
No
We want to reuse draggables
Yes
No
We want to reuse draggables
Yes
No
We want to set number of reuse draggables to be dragged
Yes
No

```
user_input = '{\
"draggables": [{"1": "t1"}, {"name_with_icon":
"t2"}, \
{"2": "t1"}]}'

correct_answer = {'1':            't1',
'name_with_icon': 't2',
'2': 't1'}
```

This is the short from of correct andswer. In short form any draggable is paired with target which is equal to 'exact' rule.

```
A)  user_input = '{"draggables":[{"1":"target1"}, \
     {"2":"target2"},{"1":"target1"},{"2":"target4"},{"2":"target4"}, {"3":"target6"

   correct_answer = [
     { 'draggables': ['1'], 'targets': ['target1',  'target3'], 'rule': 'anyof'},
     { 'draggables': ['2'], 'targets': ['target2', 'target4'],  'rule': 'anyof' },
     { 'draggables': ['3'], 'targets': ['target6'],  'rule': 'anyof' }]

                                  or

B)  user_input = '{"draggables":[{"a":"target1"}, \
     {"b":"target2"},{"c":"target3"},{"a":"target4"},{"b":"target5"}, \
     {"c":"target6"}, {"a":"target7"},{"b":"target8"},{"c":"target9"}, {"a":"target1

   correct_answer = [
     {  'draggables': ['a'],  'targets': ['target1',  'target4', 'target7', 'target10'],
        'rule': 'unordered_equal' },
     { 'draggables': ['b'], 'targets': ['target2', 'target5', 'target8'],
        'rule': 'unordered_equal'  },
     { 'draggables': ['c'],  'targets': ['target3', 'target6', 'target9'],
        'rule': 'unordered_equal'  }]
```

How this works:
1) All draggables in 'draggables' list must be dragged by user.

2-A) Their positions must be any of 'targets' list, that is what 'anyof' rul
means. This means that '1' draggable can be dragged to 'target1' many
times, or 'target3' many times, or to 'target1' and 'target3' both many ti

2-B) Their positions must be unorderly same as 'targets' list. Thies mea
that 'a' draggable must be dragged to only four positions, and these fo
posions are ['target1', 'target4', 'target7', 'target10'] in any order.

If you want draggable positions to be exact as 'targets' list, use 'exact'

If you want draggables positions to be unorderly same as 'targets' list
use 'unorderly_equal' rule.

Note, that same dragables can target different targets.

```
user_input = '{"draggables":[{"a":"target1"}, \
   {"b":"target2"},{"c":"target3"},{"b":"target5"}, \
   {"c":"target6"}, {"b":"target5"},{"c":"target9"},  {"a":"target10"}]}'
   correct_answer = [
   {
      'draggables': ['a', 'a'],
      'targets': ['target1',   'target10'],
      'rule': 'unordered_equal+number'
   },
   {
      'draggables': ['b', 'b', 'b'],
      'targets': ['target2', 'target5', 'target8'],
      'rule': 'anyof+number'        },
   {
      'draggables': ['c', 'c', 'c'],
      'targets': ['target3', 'target6', 'target9'],
      'rule': 'unordered_equal+number'
   }]
```

How this works:
1) All draggables in 'draggables' list must be dragged by user.

2) 'anyof' or 'unorderly_equal' + 'number' means that exact number of draggables in 'draggables' must be dragged. For example, only two 'a' must be dragged, and one 'a' to target1 and second 'a' to target10. Only three 'b' must be dragged, not less not more, and positions of any of those must be any of one of target2, target5, or target8.

Note, that same dragables can target different targets.

```
user_input = '{"draggables":[{"1":"target1"}, \
   {"2":"target2"},{"1":"target1"},{"2":"target4"},{"2":"target4"}, {"3":"target6"}]}'

correct_answer = [
   {  'draggables': ['1'], 'targets': ['target1',  'target3'], 'rule': 'anyof' },
   { 'draggables': ['2'], 'targets': ['target2', 'target4'], 'rule': 'anyof' },
   { 'draggables': ['3'], 'targets': ['target6'], 'rule': 'anyof'}]
```

How this works:
1) All draggables in 'draggables' list must be dragged by user.

2) Their positions must be any of 'targets' list, that is what  'anyof' rule means. This means that '1' draggable can be dragged to 'target1' many times, or 'target3' many times, or to 'target1' and 'target3' both many times.

If you want draggable positions to be exact as 'targets' list, use 'exact' rule.

(Click on image to see full size version.)

## Example

### Examples of draggables that can't be reused

```
<problem display_name="Drag and drop demos: drag and drop icons or labels
    to proper positions." >
```

```
<customresponse>
    <text>
        <h4>[Anyof rule example]</h4><br/>
        <h4>Please label hydrogen  atoms connected with left carbon atom.</h4>
        <br/>
    </text>

    <drag_and_drop_input img="/static/images/images_list/ethglycol.jpg" target_outline="true"
        one_per_target="true" no_labels="true" label_bg_color="rgb(222, 139, 238)">
        <draggable id="1" label="Hydrogen" />
        <draggable id="2" label="Hydrogen" />

        <target id="t1_o" x="10" y="67" w="100" h="100"/>
        <target id="t2" x="133" y="3" w="70" h="70"/>
        <target id="t3" x="2" y="384" w="70" h="70"/>
        <target id="t4" x="95" y="386" w="70" h="70"/>
        <target id="t5_c" x="94" y="293" w="91" h="91"/>
        <target id="t6_c" x="328" y="294" w="91" h="91"/>
        <target id="t7" x="393" y="463" w="70" h="70"/>
        <target id="t8" x="344" y="214" w="70" h="70"/>
        <target id="t9_o" x="445" y="162" w="100" h="100"/>
        <target id="t10" x="591" y="132" w="70" h="70"/>

    </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = [
{'draggables': ['1', '2'],
'targets': ['t2', 't3', 't4' ],
'rule':'anyof'
}]
if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

<customresponse>
    <text>
        <h4>[Complex grading example]</h4><br/>
        <h4>Describe carbon molecule in LCAO-MO.</h4>
        <br/>
    </text>

    <drag_and_drop_input img="/static/images/images_list/lcao-mo/lcao-mo.jpg" target_outline="true"

        <!-- filled bond -->
        <draggable id="1" icon="/static/images/images_list/lcao-mo/u_d.png" />
        <draggable id="2" icon="/static/images/images_list/lcao-mo/u_d.png" />
        <draggable id="3" icon="/static/images/images_list/lcao-mo/u_d.png" />
        <draggable id="4" icon="/static/images/images_list/lcao-mo/u_d.png" />
        <draggable id="5" icon="/static/images/images_list/lcao-mo/u_d.png" />
        <draggable id="6" icon="/static/images/images_list/lcao-mo/u_d.png" />

        <!-- up bond -->
        <draggable id="7"  icon="/static/images/images_list/lcao-mo/up.png"/>
        <draggable id="8"  icon="/static/images/images_list/lcao-mo/up.png"/>
        <draggable id="9"  icon="/static/images/images_list/lcao-mo/up.png"/>
        <draggable id="10" icon="/static/images/images_list/lcao-mo/up.png"/>

        <!-- sigma -->
        <draggable id="11" icon="/static/images/images_list/lcao-mo/sigma.png"/>
        <draggable id="12" icon="/static/images/images_list/lcao-mo/sigma.png"/>

        <!-- sigma* -->
        <draggable id="13" icon="/static/images/images_list/lcao-mo/sigma_s.png"/>
        <draggable id="14" icon="/static/images/images_list/lcao-mo/sigma_s.png"/>

        <!-- pi -->
        <draggable id="15" icon="/static/images/images_list/lcao-mo/pi.png" />

        <!-- pi* -->
        <draggable id="16" icon="/static/images/images_list/lcao-mo/pi_s.png" />

        <!-- images that should not be dragged -->
        <draggable id="17" icon="/static/images/images_list/lcao-mo/d.png" />
        <draggable id="18" icon="/static/images/images_list/lcao-mo/d.png" />

        <!-- positions of electrons and electron pairs -->
        <target id="s_left"       x="130" y="360"    w="32" h="32"/>
        <target id="s_right"      x="505" y="360"    w="32" h="32"/>
        <target id="s_sigma"      x="320" y="425"    w="32" h="32"/>
        <target id="s_sigma_star" x="320" y="290"    w="32" h="32"/>
        <target id="p_left_1"     x="80"  y="100"    w="32" h="32"/>
        <target id="p_left_2"     x="125" y="100"    w="32" h="32"/>
        <target id="p_left_3"     x="175" y="100"    w="32" h="32"/>
        <target id="p_right_1"    x="465" y="100"    w="32" h="32"/>
        <target id="p_right_2"    x="515" y="100"    w="32" h="32"/>
        <target id="p_right_3"    x="560" y="100"    w="32" h="32"/>
        <target id="p_pi_1"       x="290" y="220"    w="32" h="32"/>
        <target id="p_pi_2"       x="335" y="220"    w="32" h="32"/>
        <target id="p_sigma"      x="315" y="170"    w="32" h="32"/>
        <target id="p_pi_star_1"  x="290" y="40"     w="32" h="32"/>
        <target id="p_pi_star_2"  x="340" y="40"     w="32" h="32"/>
        <target id="p_sigma_star" x="315" y="0"      w="32" h="32"/>

        <!-- positions of names of energy levels -->
        <target id="s_sigma_name"      x="400" y="425"  w="32" h="32"/>
        <target id="s_sigma_star_name" x="400" y="290"  w="32" h="32"/>
        <target id="p_pi_name"         x="400" y="220"  w="32" h="32"/>
        <target id="p_sigma_name"      x="400" y="170"  w="32" h="32"/>
        <target id="p_pi_star_name"    x="400" y="40"   w="32" h="32"/>
        <target id="p_sigma_star_name" x="400" y="0"    w="32" h="32"/>

  </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = [
{
  'draggables': ['1', '2', '3', '4', '5', '6'],
  'targets': [
    's_left', 's_right', 's_sigma', 's_sigma_star', 'p_pi_1', 'p_pi_2'
  ],
  'rule': 'unordered_equal'
}, {
  'draggables': ['7','8', '9', '10'],
  'targets': ['p_left_1', 'p_left_2', 'p_right_1','p_right_2'],
  'rule': 'unordered_equal'
}, {
  'draggables': ['11', '12'],
  'targets': ['s_sigma_name', 'p_sigma_name'],
  'rule': 'unordered_equal'
}, {
  'draggables': ['13', '14'],
  'targets': ['s_sigma_star_name', 'p_sigma_star_name'],
  'rule': 'unordered_equal'
}, {
  'draggables': ['15'],
  'targets': ['p_pi_name'],
  'rule': 'unordered_equal'
}, {
  'draggables': ['16'],
  'targets': ['p_pi_star_name'],
  'rule': 'unordered_equal'
}]

if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

<customresponse>
    <text>
        <h4>[Another complex grading example]</h4><br/>
        <h4>Describe oxygen molecule in LCAO-MO</h4>
        <br/>
    </text>

    <drag_and_drop_input img="/static/images/images_list/lcao-mo/lcao-mo.jpg" target_outline="true" 
        <!-- filled bond -->
        <draggable id="1" icon="/static/images/images_list/lcao-mo/u_d.png" />
        <draggable id="2" icon="/static/images/images_list/lcao-mo/u_d.png" />
        <draggable id="3" icon="/static/images/images_list/lcao-mo/u_d.png" />
        <draggable id="4" icon="/static/images/images_list/lcao-mo/u_d.png" />
        <draggable id="5" icon="/static/images/images_list/lcao-mo/u_d.png" />
        <draggable id="6" icon="/static/images/images_list/lcao-mo/u_d.png" />
        <draggable id="v_fb_1" icon="/static/images/images_list/lcao-mo/u_d.png" />
        <draggable id="v_fb_2" icon="/static/images/images_list/lcao-mo/u_d.png" />
        <draggable id="v_fb_3" icon="/static/images/images_list/lcao-mo/u_d.png" />

        <!-- up bond -->
        <draggable id="7"  icon="/static/images/images_list/lcao-mo/up.png"/>
        <draggable id="8"  icon="/static/images/images_list/lcao-mo/up.png"/>
        <draggable id="9"  icon="/static/images/images_list/lcao-mo/up.png"/>
        <draggable id="10" icon="/static/images/images_list/lcao-mo/up.png"/>
        <draggable id="v_ub_1" icon="/static/images/images_list/lcao-mo/up.png"/>
        <draggable id="v_ub_2" icon="/static/images/images_list/lcao-mo/up.png"/>

        <!-- sigma -->
        <draggable id="11" icon="/static/images/images_list/lcao-mo/sigma.png"/>
        <draggable id="12" icon="/static/images/images_list/lcao-mo/sigma.png"/>
        <!-- sigma* -->
        <draggable id="13" icon="/static/images/images_list/lcao-mo/sigma_s.png"/>
        <draggable id="14" icon="/static/images/images_list/lcao-mo/sigma_s.png"/>

        <!-- pi -->
        <draggable id="15" icon="/static/images/images_list/lcao-mo/pi.png" />

        <!-- pi* -->
        <draggable id="16" icon="/static/images/images_list/lcao-mo/pi_s.png" />

        <!-- images that should not be dragged -->
        <draggable id="17" icon="/static/images/images_list/lcao-mo/d.png" />
        <draggable id="18" icon="/static/images/images_list/lcao-mo/d.png" />

        <!-- positions of electrons and electron pairs -->
        <target id="s_left"       x="130" y="360"    w="32" h="32"/>
        <target id="s_right"      x="505" y="360"    w="32" h="32"/>
        <target id="s_sigma"      x="320" y="425"    w="32" h="32"/>
        <target id="s_sigma_star" x="320" y="290"   w="32" h="32"/>
        <target id="p_left_1"     x="80"  y="100"   w="32" h="32"/>
        <target id="p_left_2"     x="125" y="100"   w="32" h="32"/>
        <target id="p_left_3"     x="175" y="100"   w="32" h="32"/>
        <target id="p_right_1"    x="465" y="100"   w="32" h="32"/>
        <target id="p_right_2"    x="515" y="100"   w="32" h="32"/>
        <target id="p_right_3"    x="560" y="100"   w="32" h="32"/>
        <target id="p_pi_1"       x="290" y="220"  w="32" h="32"/>
        <target id="p_pi_2"       x="335" y="220"  w="32" h="32"/>
        <target id="p_sigma"      x="315" y="170"  w="32" h="32"/>
        <target id="p_pi_star_1"  x="290" y="40"  w="32" h="32"/>
        <target id="p_pi_star_2"  x="340" y="40"  w="32" h="32"/>
        <target id="p_sigma_star" x="315" y="0"  w="32" h="32"/>

        <!-- positions of names of energy levels -->
        <target id="s_sigma_name" x="400" y="425"  w="32" h="32"/>
        <target id="s_sigma_star_name" x="400" y="290"  w="32" h="32"/>
        <target id="p_pi_name" x="400" y="220"  w="32" h="32"/>
        <target id="p_pi_star_name" x="400" y="40"  w="32" h="32"/>
        <target id="p_sigma_name" x="400" y="170"  w="32" h="32"/>
        <target id="p_sigma_star_name" x="400" y="0"  w="32" h="32"/>

    </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = [{
  'draggables': ['1', '2', '3', '4', '5', '6', 'v_fb_1', 'v_fb_2', 'v_fb_3'],
  'targets': [
    's_left', 's_right', 's_sigma', 's_sigma_star', 'p_pi_1', 'p_pi_2',
    'p_sigma', 'p_left_1', 'p_right_3'
  ],
  'rule': 'anyof'
}, {
  'draggables': ['7', '8', '9', '10', 'v_ub_1', 'v_ub_2'],
  'targets': [
    'p_left_2', 'p_left_3', 'p_right_1', 'p_right_2', 'p_pi_star_1',
    'p_pi_star_2'
  ],
  'rule': 'anyof'
}, {
  'draggables': ['11', '12'],
  'targets': ['s_sigma_name', 'p_sigma_name'],
  'rule': 'anyof'
}, {
  'draggables': ['13', '14'],
  'targets': ['s_sigma_star_name', 'p_sigma_star_name'],
  'rule': 'anyof'
}, {
  'draggables': ['15'],
  'targets': ['p_pi_name'],
  'rule': 'anyof'
}, {
  'draggables': ['16'],
  'targets': ['p_pi_star_name'],
  'rule': 'anyof'
}]

if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']

]]></answer>
</customresponse>

<customresponse>
    <text>
        <h4>[Individual targets with outlines, One draggable per target]</h4><br/>
        <h4>
        Drag -Ant- to first position and -Star- to third position </h4><br/>
    </text>

    <drag_and_drop_input img="/static/images/cow.png" target_outline="true">
        <draggable id="1" label="Label 1"/>
        <draggable id="name_with_icon" label="Ant" icon="/static/images/images_list/ant.jpg"/>
        <draggable id="with_icon" label="Cloud" icon="/static/images/images_list/cloud.jpg" />
        <draggable id="5" label="Label2" />
        <draggable id="2" label="Drop" icon="/static/images/images_list/drop.jpg" />
        <draggable id="name_label_icon3" label="Grass" icon="/static/images/images_list/grass.jpg" /
        <draggable id="name4" label="Star" icon="/static/images/images_list/star.png" />
        <draggable id="7" label="Label3" />

        <target id="t1" x="20" y="20" w="90" h="90"/>
        <target id="t2" x="300" y="100" w="90" h="90"/>
        <target id="t3" x="150" y="40" w="50" h="50"/>

    </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = {'name_with_icon': 't1', 'name4': 't2'}
if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

<customresponse>
```

```
    <text>
        <h4>[SMALL IMAGE,  Individual targets WITHOUT outlines, One draggable
            per target]</h4><br/>
        <h4>
            Move -Star- to the volcano opening, and -Label3- on to
            the right ear of the cow.
        </h4><br/>
    </text>

    <drag_and_drop_input img="/static/images/cow3.png" target_outline="false">
        <draggable id="1" label="Label 1"/>
        <draggable id="name_with_icon" label="Ant" icon="/static/images/images_list/ant.jpg"/>
        <draggable id="with_icon" label="Cloud" icon="/static/images/images_list/cloud.jpg" />
        <draggable id="5" label="Label2" />
        <draggable id="2" label="Drop" icon="/static/images/images_list/drop.jpg" />
        <draggable id="name_label_icon3" label="Grass" icon="/static/images/images_list/grass.jpg" /
        <draggable id="name4" label="Star" icon="/static/images/images_list/star.png" />
        <draggable id="7" label="Label3" />

        <target id="t1" x="111" y="58" w="90" h="90"/>
        <target id="t2" x="212" y="90" w="90" h="90"/>

    </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = {'name4': 't1',
                  '7': 't2'}
if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

<customresponse>
    <text>
        <h4>[Many draggables per target]</h4><br/>
        <h4>Move -Star- and -Ant- to most left target
            and -Label3- and -Label2- to most right target.</h4><br/>
    </text>

    <drag_and_drop_input img="/static/images/cow.png" target_outline="true" one_per_target="false">
        <draggable id="1" label="Label 1"/>
        <draggable id="name_with_icon" label="Ant" icon="/static/images/images_list/ant.jpg"/>
        <draggable id="with_icon" label="Cloud" icon="/static/images/images_list/cloud.jpg" />
        <draggable id="5" label="Label2" />
        <draggable id="2" label="Drop" icon="/static/images/images_list/drop.jpg" />
        <draggable id="name_label_icon3" label="Grass" icon="/static/images/images_list/grass.jpg" /
        <draggable id="name4" label="Star" icon="/static/images/images_list/star.png" />
        <draggable id="7" label="Label3" />

        <target id="t1" x="20" y="20" w="90" h="90"/>
        <target id="t2" x="300" y="100" w="90" h="90"/>
        <target id="t3" x="150" y="40" w="50" h="50"/>

    </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = {'name4': 't1',
                'name_with_icon': 't1',
                '5': 't2',
                '7':'t2'}
if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

<customresponse>
    <text>
        <h4>[Draggables can be placed anywhere on base image]</h4><br/>
        <h4>
            Place -Grass- in the middle of the image and -Ant- in the
            right upper corner.</h4><br/>
    </text>

    <drag_and_drop_input img="/static/images/cow.png" >
        <draggable id="1" label="Label 1"/>
        <draggable id="ant" label="Ant" icon="/static/images/images_list/ant.jpg"/>
        <draggable id="with_icon" label="Cloud" icon="/static/images/images_list/cloud.jpg" />
        <draggable id="5" label="Label2" />
        <draggable id="2" label="Drop" icon="/static/images/images_list/drop.jpg" />
        <draggable id="grass" label="Grass" icon="/static/images/images_list/grass.jpg" />
        <draggable id="name4" label="Star" icon="/static/images/images_list/star.png" />
        <draggable id="7" label="Label3" />

    </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = {'grass':     [[300, 200], 200],
                  'ant': [[500, 0], 200]}
if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

<customresponse>
    <text>
        <h4>[Another anyof example]</h4><br/>
        <h4>Please identify the Carbon and Oxygen atoms in the molecule.</h4><br/>
    </text>

    <drag_and_drop_input img="/static/images/images_list/ethglycol.jpg" target_outline="true" one_pe
        <draggable id="l1_c" label="Carbon" />
        <draggable id="l2" label="Methane"/>
        <draggable id="l3_o" label="Oxygen" />
        <draggable id="l4" label="Calcium" />
        <draggable id="l5" label="Methane"/>
        <draggable id="l6" label="Calcium" />
        <draggable id="l7" label="Hydrogen" />
        <draggable id="l8_c" label="Carbon" />
        <draggable id="l9" label="Hydrogen" />
        <draggable id="l10_o" label="Oxygen" />
        <target id="t1_o" x="10" y="67" w="100" h="100"/>
        <target id="t2" x="133" y="3" w="70" h="70"/>
        <target id="t3" x="2" y="384" w="70" h="70"/>
        <target id="t4" x="95" y="386" w="70" h="70"/>
        <target id="t5_c" x="94" y="293" w="91" h="91"/>
        <target id="t6_c" x="328" y="294" w="91" h="91"/>
        <target id="t7" x="393" y="463" w="70" h="70"/>
        <target id="t8" x="344" y="214" w="70" h="70"/>
        <target id="t9_o" x="445" y="162" w="100" h="100"/>
        <target id="t10" x="591" y="132" w="70" h="70"/>

    </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = [
{
    'draggables':  ['l3_o', 'l10_o'],
    'targets':  ['t1_o', 't9_o'],
    'rule': 'anyof'
},
{
    'draggables': ['l1_c','l8_c'],
    'targets': ['t5_c','t6_c'],
    'rule': 'anyof'
}
]
if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

<customresponse>
    <text>
        <h4>[Again another anyof example]</h4><br/>
        <h4>If the element appears in this molecule, drag the label onto it</h4>
        <br/>
    </text>

    <drag_and_drop_input img="/static/images/images_list/ethglycol.jpg" target_outline="true"
        one_per_target="true" no_labels="true" label_bg_color="rgb(222, 139, 238)">
        <draggable id="1" label="Hydrogen" />
        <draggable id="2" label="Hydrogen" />
        <draggable id="3" label="Nytrogen" />
        <draggable id="4" label="Nytrogen" />
        <draggable id="5" label="Boron" />
        <draggable id="6" label="Boron" />
        <draggable id="7" label="Carbon" />
        <draggable id="8" label="Carbon" />

        <target id="t1_o" x="10" y="67" w="100" h="100"/>
        <target id="t2_h" x="133" y="3" w="70" h="70"/>
        <target id="t3_h" x="2" y="384" w="70" h="70"/>
        <target id="t4_h" x="95" y="386" w="70" h="70"/>
        <target id="t5_c" x="94" y="293" w="91" h="91"/>
        <target id="t6_c" x="328" y="294" w="91" h="91"/>
        <target id="t7_h" x="393" y="463" w="70" h="70"/>
        <target id="t8_h" x="344" y="214" w="70" h="70"/>
        <target id="t9_o" x="445" y="162" w="100" h="100"/>
        <target id="t10_h" x="591" y="132" w="70" h="70"/>

    </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = [
{
    'draggables':   ['7', '8'],
    'targets':  ['t5_c', 't6_c'],
    'rule': 'anyof'
},
{
    'draggables': ['1', '2'],
    'targets': ['t2_h', 't3_h', 't4_h', 't7_h', 't8_h', 't10_h'],
    'rule': 'anyof'
}]
if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

<customresponse>
    <text>
        <h4>[Wrong base image url example]
        </h4><br/>
    </text>

    <drag_and_drop_input img="/static/images/cow3_bad.png" target_outline="false">
        <draggable id="1" label="Label 1"/>
        <draggable id="name_with_icon" label="Ant" icon="/static/images/images_list/ant.jpg"/>
        <draggable id="with_icon" label="Cloud" icon="/static/images/images_list/cloud.jpg" />
        <draggable id="5" label="Label2" />
        <draggable id="2" label="Drop" icon="/static/images/images_list/drop.jpg" />
        <draggable id="name_label_icon3" label="Grass" icon="/static/images/images_list/grass.jpg" /
        <draggable id="name4" label="Star" icon="/static/images/images_list/star.png" />
        <draggable id="7" label="Label3" />

        <target id="t1" x="111" y="58" w="90" h="90"/>
        <target id="t2" x="212" y="90" w="90" h="90"/>

    </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = {'name4': 't1',
                '7': 't2'}
if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

</problem>
```

### Draggables can be reused

```
<problem display_name="Drag and drop demos: drag and drop icons or labels
    to proper positions." >

<customresponse>
    <text>
        <h4>[Draggable is reusable example]</h4>
        <br/>
        <h4>Please label all hydrogen atoms.</h4>
        <br/>
    </text>

    <drag_and_drop_input
        img="/static/images/images_list/ethglycol.jpg"
        target_outline="true"
        one_per_target="true"
        no_labels="true"
        label_bg_color="rgb(222, 139, 238)"
    >
        <draggable id="1" label="Hydrogen" can_reuse='true' />

        <target id="t1_o" x="10" y="67" w="100" h="100" />
        <target id="t2" x="133" y="3" w="70" h="70" />
        <target id="t3" x="2" y="384" w="70" h="70" />
        <target id="t4" x="95" y="386" w="70" h="70" />
        <target id="t5_c" x="94" y="293" w="91" h="91" />
        <target id="t6_c" x="328" y="294" w="91" h="91" />
        <target id="t7" x="393" y="463" w="70" h="70" />
        <target id="t8" x="344" y="214" w="70" h="70" />
        <target id="t9_o" x="445" y="162" w="100" h="100" />
        <target id="t10" x="591" y="132" w="70" h="70" />
    </drag_and_drop_input>

    <answer type="loncapa/python">
    <![CDATA[
correct_answer = [{
    'draggables': ['1'],
    'targets': ['t2', 't3', 't4', 't7', 't8', 't10'],
    'rule': 'exact'
}]
if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
    ]]>
    </answer>
</customresponse>

<customresponse>
    <text>
        <h4>[Complex grading example]</h4><br/>
        <h4>Describe carbon molecule in LCAO-MO.</h4>
        <br/>
    </text>

    <drag_and_drop_input img="/static/images/images_list/lcao-mo/lcao-mo.jpg" target_outline="true" 
```

```
        <!-- filled bond -->
        <draggable id="1" icon="/static/images/images_list/lcao-mo/u_d.png" can_reuse="true" />

        <!-- up bond -->
        <draggable id="7"  icon="/static/images/images_list/lcao-mo/up.png" can_reuse="true" />

        <!-- sigma -->
        <draggable id="11" icon="/static/images/images_list/lcao-mo/sigma.png" can_reuse="true" />

        <!-- sigma* -->
        <draggable id="13" icon="/static/images/images_list/lcao-mo/sigma_s.png" can_reuse="true" />

        <!-- pi -->
        <draggable id="15" icon="/static/images/images_list/lcao-mo/pi.png" can_reuse="true" />

        <!-- pi* -->
        <draggable id="16" icon="/static/images/images_list/lcao-mo/pi_s.png" can_reuse="true" />

        <!-- images that should not be dragged -->
        <draggable id="17" icon="/static/images/images_list/lcao-mo/d.png" can_reuse="true" />

        <!-- positions of electrons and electron pairs -->
        <target id="s_left"       x="130" y="360"    w="32" h="32"/>
        <target id="s_right"      x="505" y="360"    w="32" h="32"/>
        <target id="s_sigma"      x="320" y="425"    w="32" h="32"/>
        <target id="s_sigma_star" x="320" y="290"    w="32" h="32"/>
        <target id="p_left_1"     x="80"  y="100"    w="32" h="32"/>
        <target id="p_left_2"     x="125" y="100"    w="32" h="32"/>
        <target id="p_left_3"     x="175" y="100"    w="32" h="32"/>
        <target id="p_right_1"    x="465" y="100"    w="32" h="32"/>
        <target id="p_right_2"    x="515" y="100"    w="32" h="32"/>
        <target id="p_right_3"    x="560" y="100"    w="32" h="32"/>
        <target id="p_pi_1"       x="290" y="220"    w="32" h="32"/>
        <target id="p_pi_2"       x="335" y="220"    w="32" h="32"/>
        <target id="p_sigma"      x="315" y="170"    w="32" h="32"/>
        <target id="p_pi_star_1"  x="290" y="40"     w="32" h="32"/>
        <target id="p_pi_star_2"  x="340" y="40"     w="32" h="32"/>
        <target id="p_sigma_star" x="315" y="0"      w="32" h="32"/>

        <!-- positions of names of energy levels -->
        <target id="s_sigma_name"      x="400" y="425"  w="32" h="32"/>
        <target id="s_sigma_star_name" x="400" y="290"  w="32" h="32"/>
        <target id="p_pi_name"         x="400" y="220"  w="32" h="32"/>
        <target id="p_sigma_name"      x="400" y="170"  w="32" h="32"/>
        <target id="p_pi_star_name"    x="400" y="40"   w="32" h="32"/>
        <target id="p_sigma_star_name" x="400" y="0"    w="32" h="32"/>

  </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = [
{
  'draggables': ['1'],
  'targets': [
    's_left', 's_right', 's_sigma', 's_sigma_star', 'p_pi_1', 'p_pi_2'
  ],
  'rule': 'exact'
}, {
  'draggables': ['7'],
  'targets': ['p_left_1', 'p_left_2', 'p_right_1','p_right_2'],
  'rule': 'exact'
}, {
  'draggables': ['11'],
  'targets': ['s_sigma_name', 'p_sigma_name'],
  'rule': 'exact'
}, {
  'draggables': ['13'],
  'targets': ['s_sigma_star_name', 'p_sigma_star_name'],
  'rule': 'exact'
}, {
  'draggables': ['15'],
  'targets': ['p_pi_name'],
  'rule': 'exact'
}, {
  'draggables': ['16'],
  'targets': ['p_pi_star_name'],
  'rule': 'exact'
}]

if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

<customresponse>
    <text>
        <h4>[Many draggables per target]</h4><br/>
        <h4>Move two Stars and three Ants to most left target
            and one Label3 and four Label2 to most right target.</h4><br/>
    </text>

    <drag_and_drop_input img="/static/images/cow.png" target_outline="true" one_per_target="false">
        <draggable id="1" label="Label 1" can_reuse="true" />
        <draggable id="name_with_icon" label="Ant" icon="/static/images/images_list/ant.jpg" can_reus
        <draggable id="with_icon" label="Cloud" icon="/static/images/images_list/cloud.jpg" can_reuse
        <draggable id="5" label="Label2" can_reuse="true" />
        <draggable id="2" label="Drop" icon="/static/images/images_list/drop.jpg" can_reuse="true" />
        <draggable id="name_label_icon3" label="Grass" icon="/static/images/images_list/grass.jpg" ca
        <draggable id="name4" label="Star" icon="/static/images/images_list/star.png" can_reuse="true
        <draggable id="7" label="Label3" can_reuse="true" />

        <target id="t1" x="20" y="20" w="90" h="90"/>
        <target id="t2" x="300" y="100" w="90" h="90"/>
        <target id="t3" x="150" y="40" w="50" h="50"/>

    </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = [
{
    'draggables': ['name4'],
    'targets': [
        't1', 't1'
    ],
    'rule': 'exact'
},
{
    'draggables': ['name_with_icon'],
    'targets': [
        't1', 't1', 't1'
    ],
    'rule': 'exact'
},
{
    'draggables': ['5'],
    'targets': [
        't2', 't2', 't2', 't2'
    ],
    'rule': 'exact'
},
{
    'draggables': ['7'],
    'targets': [
        't2'
    ],
    'rule': 'exact'
}
]
if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

<customresponse>
    <text>
        <h4>[Draggables can be placed anywhere on base image]</h4><br/>
        <h4>
            Place -Grass- in the middle of the image and -Ant- in the
            right upper corner.</h4><br/>
    </text>

    <drag_and_drop_input img="/static/images/cow.png" >
        <draggable id="1" label="Label 1" can_reuse="true" />
        <draggable id="ant" label="Ant" icon="/static/images/images_list/ant.jpg" can_reuse="true" /
        <draggable id="with_icon" label="Cloud" icon="/static/images/images_list/cloud.jpg" can_reuse
        <draggable id="5" label="Label2" can_reuse="true" />
        <draggable id="2" label="Drop" icon="/static/images/images_list/drop.jpg" can_reuse="true" /
        <draggable id="grass" label="Grass" icon="/static/images/images_list/grass.jpg" can_reuse="tr
        <draggable id="name4" label="Star" icon="/static/images/images_list/star.png" can_reuse="true
        <draggable id="7" label="Label3" can_reuse="true" />

    </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = {
    'grass': [[300, 200], 200],
    'ant': [[500, 0], 200]
}
if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
```

```
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

<customresponse>
    <text>
        <h4>[Another anyof example]</h4><br/>
        <h4>Please identify the Carbon and Oxygen atoms in the molecule.</h4><br/>
    </text>

    <drag_and_drop_input img="/static/images/images_list/ethglycol.jpg" target_outline="true" one_per
        <draggable id="l1_c" label="Carbon" can_reuse="true" />
        <draggable id="l2" label="Methane" can_reuse="true" />
        <draggable id="l3_o" label="Oxygen" can_reuse="true" />
        <draggable id="l4" label="Calcium" can_reuse="true" />
        <draggable id="l7" label="Hydrogen" can_reuse="true" />

        <target id="t1_o" x="10" y="67" w="100" h="100"/>
        <target id="t2" x="133" y="3" w="70" h="70"/>
        <target id="t3" x="2" y="384" w="70" h="70"/>
        <target id="t4" x="95" y="386" w="70" h="70"/>
        <target id="t5_c" x="94" y="293" w="91" h="91"/>
        <target id="t6_c" x="328" y="294" w="91" h="91"/>
        <target id="t7" x="393" y="463" w="70" h="70"/>
        <target id="t8" x="344" y="214" w="70" h="70"/>
        <target id="t9_o" x="445" y="162" w="100" h="100"/>
        <target id="t10" x="591" y="132" w="70" h="70"/>

    </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = [
{
    'draggables': ['l3_o'],
    'targets': ['t1_o', 't9_o'],
    'rule': 'exact'
},
{
    'draggables': ['l1_c'],
    'targets': ['t5_c', 't6_c'],
    'rule': 'exact'
}
]
if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

<customresponse>
    <text>
        <h4>[Exact number of draggables for a set of targets.]</h4><br/>
        <h4>Drag two Grass and one Star to first or second positions, and three Cloud to any of the t
        <br/>
    </text>
    <drag_and_drop_input img="/static/images/cow.png" target_outline="true" one_per_target="false">
        <draggable id="1" label="Label 1" can_reuse="true" />
        <draggable id="name_with_icon" label="Ant" icon="/static/images/images_list/ant.jpg" can_reus
        <draggable id="with_icon" label="Cloud" icon="/static/images/images_list/cloud.jpg" can_reuse
        <draggable id="5" label="Label2" can_reuse="true" />
        <draggable id="2" label="Drop" icon="/static/images/images_list/drop.jpg" can_reuse="true" />
        <draggable id="name_label_icon3" label="Grass" icon="/static/images/images_list/grass.jpg" ca
        <draggable id="name4" label="Star" icon="/static/images/images_list/star.png" can_reuse="true
        <draggable id="7" label="Label3" can_reuse="true" />

        <target id="t1" x="20" y="20" w="90" h="90"/>
        <target id="t2" x="300" y="100" w="90" h="90"/>
        <target id="t3" x="150" y="40" w="50" h="50"/>

    </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = [
{
    'draggables': ['name_label_icon3', 'name_label_icon3'],
    'targets': ['t1', 't3'],
    'rule': 'unordered_equal+number'
},
{
    'draggables': ['name4'],
    'targets': ['t1', 't3'],
    'rule': 'anyof+number'
},
{
    'draggables': ['with_icon', 'with_icon', 'with_icon'],
    'targets': ['t1', 't2', 't3'],
    'rule': 'anyof+number'
}
]
if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

<customresponse>
    <text>
        <h4>[As many as you like draggables for a set of targets.]</h4><br/>
        <h4>Drag some Grass to any of the targets, and some Stars to either first or last target.</h
        <br/>
    </text>

    <drag_and_drop_input img="/static/images/cow.png" target_outline="true" one_per_target="false">
        <draggable id="1" label="Label 1" can_reuse="true" />
        <draggable id="name_with_icon" label="Ant" icon="/static/images/images_list/ant.jpg" can_reus
        <draggable id="with_icon" label="Cloud" icon="/static/images/images_list/cloud.jpg" can_reuse
        <draggable id="5" label="Label2" can_reuse="true" />
        <draggable id="2" label="Drop" icon="/static/images/images_list/drop.jpg" can_reuse="true" />
        <draggable id="name_label_icon3" label="Grass" icon="/static/images/images_list/grass.jpg" ca
        <draggable id="name4" label="Star" icon="/static/images/images_list/star.png" can_reuse="true
        <draggable id="7" label="Label3" can_reuse="true" />
        <target id="t1" x="20" y="20" w="90" h="90"/>
        <target id="t2" x="300" y="100" w="90" h="90"/>
        <target id="t3" x="150" y="40" w="50" h="50"/>

    </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = [
{
    'draggables': ['name_label_icon3'],
    'targets': ['t1', 't2', 't3'],
    'rule': 'anyof'
},
{
    'draggables': ['name4'],
    'targets': ['t1', 't2'],
    'rule': 'anyof'
}
]
if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

</problem>
```

#### Examples of targets on draggables

```
<problem display_name="Drag and drop demos chem features: drag and drop icons or labels
    to proper positions." attempts="10">

<customresponse>
    <text>
        <h4>[Simple grading example: draggables on draggables]</h4><br/>
        <h4>Describe carbon molecule in LCAO-MO.</h4><br/>
        <br/>
    </text>

    <drag_and_drop_input img="/static/images/images_list/lcao-mo/lcao-mo.jpg" target_outline="true" >

        <!-- filled bond -->
        <draggable id="up_and_down" icon="/static/images/images_list/lcao-mo/u_d.png" can_reuse="true
        <!-- up bond -->
        <draggable id="up"  icon="/static/images/images_list/lcao-mo/up.png" can_reuse="true" />

        <draggable id="s" icon="/static/images/images_list/lcao-mo/orbital_single.png" label="s orbit
            <target id="1" x="0" y="0" w="32" h="32"/>
        </draggable>

        <draggable id="p" icon="/static/images/images_list/lcao-mo/orbital_triple.png" can_reuse="tru
            <target id="1" x="0" y="0" w="32" h="32"/>
            <target id="2" x="34" y="0" w="32" h="32"/>
            <target id="3" x="68" y="0" w="32" h="32"/>
        </draggable>
        <!-- positions of electrons and electron pairs -->
        <target id="s_l" x="130" y="360" w="32" h="32"/>
        <target id="s_r" x="505" y="360" w="32" h="32"/>
        <target id="p_l" x="80"  y="100" w="100" h="32"/>
        <target id="p_r" x="465" y="100" w="100" h="32"/>

  </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = [
    {
        'draggables': ['p'],
        'targets': ['p_l', 'p_r'],
        'rule': 'unordered_equal'
    },
    {
        'draggables': ['s'],
        'targets': ['s_l', 's_r'],
        'rule': 'unordered_equal'
    },
    {
      'draggables': ['up_and_down'],
      'targets': [
        's_l[s][1]', 's_r[s][1]'
      ],
      'rule': 'unordered_equal'
    },
    {
      'draggables': ['up'],
      'targets': [
        'p_l[p][1]', 'p_l[p][3]', 'p_r[p][1]', 'p_r[p][3]'
      ],
      'rule': 'unordered_equal'
    }
]

if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

<customresponse>
    <text>
        <h4>[Complex grading example: draggables on draggables]</h4><br/>
        <h4>Describe carbon molecule in LCAO-MO.</h4>
        <br/>
    </text>

    <drag_and_drop_input img="/static/images/images_list/lcao-mo/lcao-mo-clean.jpg" target_outline="t

        <!-- filled bond -->
        <draggable id="up_and_down" icon="/static/images/images_list/lcao-mo/u_d.png" can_reuse="true
        <!-- up bond -->
        <draggable id="up" icon="/static/images/images_list/lcao-mo/up.png" can_reuse="true" />

        <!-- images that should not be dragged -->
        <draggable id="down" icon="/static/images/images_list/lcao-mo/d.png" can_reuse="true" />

        <draggable id="s" icon="/static/images/images_list/lcao-mo/orbital_single.png" label="s orbit
            <target id="1" x="0" y="0" w="32" h="32"/>
        </draggable>

        <draggable id="p" icon="/static/images/images_list/lcao-mo/orbital_triple.png" can_reuse="tru
            <target id="1" x="0" y="0" w="32" h="32"/>
            <target id="2" x="34" y="0" w="32" h="32"/>
            <target id="3" x="68" y="0" w="32" h="32"/>
        </draggable>

        <draggable id="s-sigma" icon="/static/images/images_list/lcao-mo/orbital_single.png" label="s
            <target id="1" x="0" y="0" w="32" h="32"/>
        </draggable>

        <draggable id="s-sigma*" icon="/static/images/images_list/lcao-mo/orbital_single.png" label="
            <target id="1" x="0" y="0" w="32" h="32"/>
        </draggable>

        <draggable id="p-pi" icon="/static/images/images_list/lcao-mo/orbital_double.png" label="p-pi
            <target id="1" x="0" y="0" w="32" h="32"/>
            <target id="2" x="34" y="0" w="32" h="32"/>
        </draggable>

        <draggable id="p-sigma" icon="/static/images/images_list/lcao-mo/orbital_single.png" label="p
            <target id="1" x="0" y="0" w="32" h="32"/>
        </draggable>

        <draggable id="p-pi*" icon="/static/images/images_list/lcao-mo/orbital_double.png" label="p-p
            <target id="1" x="0" y="0" w="32" h="32"/>
            <target id="2" x="34" y="0" w="32" h="32"/>
        </draggable>

        <draggable id="p-sigma*" icon="/static/images/images_list/lcao-mo/orbital_single.png" label="
            <target id="1" x="0" y="0" w="32" h="32"/>
        </draggable>

        <!-- positions of electrons and electron pairs -->
        <target id="s-left-target" x="130" y="360" w="32" h="32"/>
        <target id="s-right-target" x="505" y="360" w="32" h="32"/>
        <target id="s-sigma-target" x="315" y="425" w="32" h="32"/>
        <target id="s-sigma*-target" x="315" y="290" w="32" h="32"/>
        <target id="p-left-target" x="80"  y="100" w="100" h="32"/>
        <target id="p-right-target" x="480" y="100" w="100" h="32"/>
        <target id="p-pi-target" x="300" y="220" w="66" h="32"/>
        <target id="p-sigma-target" x="315" y="170" w="32" h="32"/>
        <target id="p-pi*-target" x="300" y="40" w="66" h="32"/>
        <target id="p-sigma*-target" x="315" y="0" w="32" h="32"/>

    </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = [
{'draggables': ['p'], 'targets': ['p-left-target', 'p-right-target'], 'rule': 'unordered_equal'},
{'draggables': ['s'], 'targets': ['s-left-target', 's-right-target'], 'rule': 'unordered_equal'},
{'draggables': ['s-sigma'], 'targets': ['s-sigma-target'], 'rule': 'exact'},
{'draggables': ['s-sigma*'], 'targets': ['s-sigma*-target'], 'rule': 'exact'},
{'draggables': ['p-pi'], 'targets': ['p-pi-target'], 'rule': 'exact'},
{'draggables': ['p-sigma'], 'targets': ['p-sigma-target'], 'rule': 'exact'},
{'draggables': ['p-pi*'], 'targets': ['p-pi*-target'], 'rule': 'exact'},
{'draggables': ['p-sigma*'], 'targets': ['p-sigma*-target'], 'rule': 'exact'},
{
    'draggables': ['up_and_down'],
    'targets': ['s-left-target[s][1]', 's-right-target[s][1]', 's-sigma-target[s-sigma][1]', 's-sigma
    'rule': 'unordered_equal'
},
{
    'draggables': ['up'],
    'targets': ['p-left-target[p][1]', 'p-left-target[p][2]', 'p-right-target[p][2]', 'p-right-target
    'rule': 'unordered_equal'
}
]

if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
</customresponse>

<customresponse>
    <text>
        <h4>[Complex grading example: no draggables on draggables]</h4><br/>
        <h4>Describe carbon molecule in LCAO-MO.</h4>
        <br/>
    </text>

    <drag_and_drop_input img="/static/images/images_list/lcao-mo/lcao-mo.jpg" target_outline="true">

        <!-- filled bond -->
        <draggable id="1" icon="/static/images/images_list/lcao-mo/u_d.png" can_reuse="true" />

        <!-- up bond -->
        <draggable id="7"  icon="/static/images/images_list/lcao-mo/up.png" can_reuse="true" />

        <!-- sigma -->
        <draggable id="11" icon="/static/images/images_list/lcao-mo/sigma.png" can_reuse="true" />

        <!-- sigma* -->
        <draggable id="13" icon="/static/images/images_list/lcao-mo/sigma_s.png" can_reuse="true" />

        <!-- pi -->
        <draggable id="15" icon="/static/images/images_list/lcao-mo/pi.png" can_reuse="true" />

        <!-- pi* -->
        <draggable id="16" icon="/static/images/images_list/lcao-mo/pi_s.png" can_reuse="true" />

        <!-- images that should not be dragged -->
        <draggable id="17" icon="/static/images/images_list/lcao-mo/d.png" can_reuse="true" />

        <!-- positions of electrons and electron pairs -->
        <target id="s_left"       x="130" y="360"    w="32" h="32"/>
        <target id="s_right"      x="505" y="360"    w="32" h="32"/>
        <target id="s_sigma"      x="320" y="425"    w="32" h="32"/>
        <target id="s_sigma_star" x="320" y="290"    w="32" h="32"/>
        <target id="p_left_1"     x="80"  y="100"    w="32" h="32"/>
        <target id="p_left_2"     x="125" y="100"    w="32" h="32"/>
        <target id="p_left_3"     x="175" y="100"    w="32" h="32"/>
        <target id="p_right_1"    x="465" y="100"    w="32" h="32"/>
        <target id="p_right_2"    x="515" y="100"    w="32" h="32"/>
        <target id="p_right_3"    x="560" y="100"    w="32" h="32"/>
        <target id="p_pi_1"       x="290" y="220"    w="32" h="32"/>
        <target id="p_pi_2"       x="335" y="220"    w="32" h="32"/>
        <target id="p_sigma"      x="315" y="170"    w="32" h="32"/>
        <target id="p_pi_star_1"  x="290" y="40"     w="32" h="32"/>
        <target id="p_pi_star_2"  x="340" y="40"     w="32" h="32"/>
        <target id="p_sigma_star" x="315" y="0"      w="32" h="32"/>

        <!-- positions of names of energy levels -->
        <target id="s_sigma_name"      x="400" y="425"  w="32" h="32"/>
        <target id="s_sigma_star_name" x="400" y="290"  w="32" h="32"/>
        <target id="p_pi_name"         x="400" y="220"  w="32" h="32"/>
        <target id="p_sigma_name"      x="400" y="170"  w="32" h="32"/>
        <target id="p_pi_star_name"    x="400" y="40"   w="32" h="32"/>
        <target id="p_sigma_star_name" x="400" y="0"    w="32" h="32"/>

  </drag_and_drop_input>

    <answer type="loncapa/python"><![CDATA[
correct_answer = [
{
  'draggables': ['1'],
  'targets': [
    's_left', 's_right', 's_sigma', 's_sigma_star', 'p_pi_1', 'p_pi_2'
  ],
  'rule': 'exact'
}, {
  'draggables': ['7'],
  'targets': ['p_left_1', 'p_left_2', 'p_right_2','p_right_3'],
  'rule': 'exact'
}, {
  'draggables': ['11'],
  'targets': ['s_sigma_name', 'p_sigma_name'],
  'rule': 'exact'
}, {
  'draggables': ['13'],
  'targets': ['s_sigma_star_name', 'p_sigma_star_name'],
  'rule': 'exact'
}, {
  'draggables': ['15'],
  'targets': ['p_pi_name'],
  'rule': 'exact'
}, {
  'draggables': ['16'],
  'targets': ['p_pi_star_name'],
  'rule': 'exact'
}]

if draganddrop.grade(submission[0], correct_answer):
    correct = ['correct']
else:
    correct = ['incorrect']
]]></answer>
```

```
</customresponse>

</problem>
```

## 1.3.2 XML format of graphical slider tool [xmodule]

### Format description

Graphical slider tool (GST) main tag is:

```
<graphical_slider_tool> BODY </graphical_slider_tool>
```

`graphical_slider_tool` tag must have two children tags: `render` and `configuration`.

#### Render tag

Render tag can contain usual html tags mixed with some GST specific tags:

```
<slider/> - represents jQuery slider for changing a parameter's value
<textbox/> - represents a text input field for changing a parameter's value
<plot/> - represents Flot JS plot element
```

Also GST will track all elements inside `<render></render>` where `id` attribute is set, and a corresponding parameter referencing that `id` is present in the configuration section below. These will be referred to as dynamic elements.

The contents of the <render> section will be shown to the user after all occurrences of:

```
<slider var="{parameter name}" [style="{CSS statements}"] />
<textbox var="{parameter name}" [style="{CSS statements}"] />
<plot [style="{CSS statements}"] />
```

have been converted to actual sliders, text inputs, and a plot graph. Everything in square brackets is optional. After initialization, all text input fields, sliders, and dynamic elements will be set to the initial values of the parameters that they are assigned to.

`{parameter name}` specifies the parameter to which the slider or text input will be attached to.

[style="{CSS statements}"] specifies valid CSS styling. It will be passed directly to the browser without any parsing.

There is a one-to-one relationship between a slider and a parameter. I.e. for one parameter you can put only one `<slider>` in the `<render>` section. However, you don't have to specify a slider - they are optional.

There is a many-to-one relationship between text inputs and a parameter. I.e. for one parameter you can put many '<textbox>' elements in the `<render>` section. However, you don't have to specify a text input - they are optional.

You can put only one `<plot>` in the `<render>` section. It is not required.

**Slider tag** Slider tag must have `var` attribute and optional `style` attribute:

```
<slider var='a' style="width:400px;float:left;" />
```

After processing, slider tags will be replaced by jQuery UI sliders with applied `style` attribute.

`var` attribute must correspond to a parameter. Parameters can be used in any of the `function` tags in `functions` tag. By moving slider, value of parameter `a` will change, and so result of function, that depends on parameter `a`, will also change.

**Textbox tag** Texbox tag must have `var` attribute and optional `style` attribute:

```
<textbox var="b" style="width:50px; float:left; margin-left:10px;" />
```

After processing, textbox tags will be replaced by html text inputs with applied `style` attribute. If you want a readonly text input, then you should use a dynamic element instead (see section below "HTML tagsd with ID").

`var` attribute must correspond to a parameter. Parameters can be used in any of the `function` tags in `functions` tag. By changing the value on the text input, value of parameter `a` will change, and so result of function, that depends on parameter `a`, will also change.

**Plot tag** Plot tag may have optional `style` attribute:

```
<plot style="width:50px; float:left; margin-left:10px;" />
```

After processing plot tags will be replaced by Flot JS plot with applied `style` attribute.

**HTML tags with ID (dynamic elements)** Any HTML tag with ID, e.g. `<span id="answer_span_1">` can be used as a place where result of function can be inserted. To insert function result to an element, element ID must be included in `function` tag as `el_id` attribute and `output` value must be `"element"`:

```
<function output="element" el_id="answer_span_1">
                function add(a, b, precision) {
                    var x = Math.pow(10, precision || 2);
                    return (Math.round(a * x) + Math.round(b * x)) / x;
                }

                return add(a, b, 5);
</function>
```

### Configuration tag

The configuration tag contains parameter settings, graph settings, and function definitions which are to be plotted on the graph and that use specified parameters.

Configuration tag contains two mandatory tag `functions` and `parameters` and may contain another `plot` tag.

**Parameters tag** `Parameters` tag contains `parameter` tags. Each `parameter` tag must have `var`, `max`, `min`, `step` and `initial` attributes:

```
<parameters>
        <param var="a" min="-10.0" max="10.0" step="0.1" initial="0" />
        <param var="b" min="-10.0" max="10.0" step="0.1" initial="0" />
</parameters>
```

`var` attribute links min, max, step and initial values to parameter name.

`min` attribute is the minimal value that a parameter can take. Slider and input values can not go below it.

`max` attribute is the maximal value that a parameter can take. Slider and input values can not go over it.

`step` attribute is value of slider step. When a slider increase or decreases the specified parameter, it will do so by the amount specified with 'step'

`initial` attribute is the initial value that the specified parameter should be set to. Sliders and inputs will initially show this value.

The parameter's name is specified by the `var` property. All occurrences of sliders and/or text inputs that specify a `var` property, will be connected to this parameter - i.e. they will reflect the current value of the parameter, and will be updated when the parameter changes.

If at lest one of these attributes is not set, then the parameter will not be used, slider's and/or text input elements that specify this parameter will not be activated, and the specified functions which use this parameter will not return a numeric value. This means that neglecting to specify at least one of the attributes for some parameter will have the result of the whole GST instance not working properly.

**Functions tag** For the GST to do something, you must defined at least one function, which can use any of the specified parameter values. The function expects to take the `x` value, do some calculations, and return the `y` value. I.e. this is a 2D plot in Cartesian coordinates. This is how the default function is meant to be used for the graph.

There are other special cases of functions. They are used mainly for outputting to elements, plot labels, or for custom output. Because the return a single value, and that value is meant for a single element, these function are invoked only with the set of all of the parameters. I.e. no `x` value is available inside them. They are useful for showing the current value of a parameter, showing complex static formulas where some parameter's value must change, and other useful things.

The different style of function is specified by the `output` attribute.

Each function must be defined inside `function` tag in `functions` tag:

```
<functions>
    <function output="element" el_id="answer_span_1">
        function add(a, b, precision) {
            var x = Math.pow(10, precision || 2);
            return (Math.round(a * x) + Math.round(b * x)) / x;
        }

        return add(a, b, 5);
    </function>
</functions>
```

The parameter names (along with their values, as provided from text inputs and/or sliders), will be available inside all defined functions. A defined function body string will be parsed internally by the browser's JavaScript engine and converted to a true JS function.

The function's parameter list will automatically be created and populated, and will include the `x` (when `output` is not specified or is set to `"graph"`), and all of the specified parameter values (from sliders and text inputs). This means that each of the defined functions will have access to all of the parameter values. You don't have to use them, but they will be there.

Examples:

```
<function>
    return x;
</function>

<function dot="true" label="\(y_2\)">
    return (x + a) * Math.sin(x * b);
</function>

<function color="green">
    function helperFunc(c1) {
        return c1 * c1 - a;
    }
    return helperFunc(x + 10 * a * b) + Math.sin(a - x);
</function>
```

Required parameters:

```
function body:

A string composing a normal JavaScript function
except that there is no function declaration
(along with parameters), and no closing bracket.

So if you normally would have written your
JavaScript function like this:

    function myFunc(x, a, b) {
        return x * a + b;
    }

here you must specify just the function body
(everything that goes between '{' and '}'). So,
you would specify the above function like so (the
bare-bone minimum):

    <function>return x * a + b;</function>

VERY IMPORTANT: Because the function will be passed
to the browser as a single string, depending on implementation
specifics, the end-of-line characters can be stripped. This
means that single line JavaScript comments (starting with "//")
can lead to the effect that everything after the first such comment
will be treated as a comment. Therefore, it is absolutely
necessary that such single line comments are not used when
defining functions for GST. You can safely use the alternative
multiple line JavaScript comments (such comments start with "/*"
and end with "*/).

VERY IMPORTANT: If you have a large function body, and decide to
split it into several lines, than you must wrap it in "CDATA" like
so:

    <function>
    <![CDATA[
        var dNew;

        dNew = 0.3;

        return x * a + b - dNew;
    ]]>
    </function>
```

Optional parameters:

```
color:  Color name ('red', 'green', etc.) or in the form of
        '#FFFF00'. If not specified, a default color (different
        one for each graphed function) will be given by Flot JS.
line:   A string - 'true' or 'false'. Should the data points be
        connected by a line on the graph? Default is 'true'.
dot:    A string - 'true' or 'false'. Should points be shown for
        each data point on the graph? Default is 'false'.
bar:    A string - 'true' or 'false'. When set to 'true', points
        will be plotted as bars.
label:  A string. If provided, will be shown in the legend, along
```

```
        with the color that was used to plot the function.
output: 'element', 'none', 'plot_label', or 'graph'. If not defined,
        function will be plotted (same as setting 'output' to 'graph').
        If defined, and other than 'graph', function will not be
        plotted, but it's output will be inserted into the element
        with ID specified by 'el_id' attribute.
el_id:  Id of HTML element, defined in '<render>' section. Value of
        function will be inserted as content of this element.
disable_auto_return: By default, if JavaScript function string is written
                     without a "return" statement, the "return" will be
                     prepended to it. Set to "true" to disable this
                     functionality. This is done so that simple functions
                     can be defined in an easy fashion (for example, "a",
                     which will be translated into "return a").
update_on: A string - 'change', or 'slide'. Default (if not set) is
           'slide'. This defines the event on which a given function is
           called, and its result is inserted into an element. This
           setting is relevant only when "output" is other than "graph".
```

**When specifying `el_id`, it is essential to set "output" to one of**

**element - GST will invoke the function, and the return of it will be** inserted into a HTML element with id specified by `el_id`.

**none - GST will simply inoke the function. It is left to the instructor** who writes the JavaScript function body to update all necesary HTML elements inside the function, before it exits. This is done so that extra steps can be preformed after an HTML element has been updated with a value. Note, that because the return value from this function is not actually used, it will be tempting to omit the "return" statement. However, in this case, the attribute "disable_auto_return" must be set to "true" in order to prevent GST from inserting a "return" statement automatically.

**plot_label - GST will process all plot labels (which are strings), and** will replace the all instances of substrings specified by `el_id` with the returned value of the function. This is necessary if you want a label in the graph to have some changing number. Because of the nature of Flot JS, it is impossible to achieve the same effect by setting the "output" attribute to "element", and including a HTML element in the label.

The above values for "output" will tell GST that the function is meant for an HTML element (not for graph), and that it should not get an 'x' parameter (along with some value).

**[Note on MathJax and labels]** Independently of this module, will render all TeX code within the `<render>` section into nice mathematical formulas. Just remember to wrap it in one of:

```
\(  and  \)  -  for inline formulas (formulas surrounded by
                standard text)
\[  and  \]  -  if you want the formula to be a separate line
```

It is possible to define a label in standard TeX notation. The JS library MathJax will work on these labels also because they are inserted on top of the plot as standard HTML (text within a DIV).

If the label is dynamic, i.e. it will contain some text (numeric, or other) that has to be updated on a parameter's change, then one can define a special function to handle this. The "output" of such a function must be set to "none", and the JavaScript code inside this function must update the MathJax element by itself. Before exiting, MathJax typeset function should be called so that the new text will be re-rendered by MathJax. For example:

```
<render>
    ...
    <span id="dynamic_mathjax"></span>
</render>
```

```
...
<function output="none" el_id="dynamic_mathjax">
<![CDATA[
    var out_text;

    out_text = "\\[\\mathrm{Percent \\space of \\space treated \\space with \\space YSS=\\frac{"
      +(treated_men*10)+"\\space men *"
      +(your_m_tx_yss/100)+"\\space prev. +\\space "
      +((100-treated_men)*10)+"\\space women *"
      +(your_f_tx_yss/100)+"\\space prev.}"
      +"{1000\\space total\\space treated\\space patients}"
      +"="+drummond_combined[0][1]+"\\%}\\]";
      mathjax_for_prevalence_calcs+="\\[\\mathrm{Percent \\space of \\space untreated \\space with \
      +(untreated_men*10)+"\\space men *"
      +(your_m_utx_yss/100)+"\\space prev. +\\space "
      +((100-untreated_men)*10)+"\\space women *"
      +(your_f_utx_yss/100)+"\\space prev.}"
      +"{1000\\space total\\space untreated\\space patients}"
      +"="+drummond_combined[1][1]+"\\%}\\]";

    $("#dynamic_mathjax").html(out_text);

    MathJax.Hub.Queue(["Typeset",MathJax.Hub,"dynamic_mathjax"]);
]]>
</function>
...
```

**Plot tag** `Plot` tag inside `configuration` tag defines settings for plot output.

Required parameters:

```
xrange: 2 functions that must return value. Value is constant (3.1415)
        or depend on parameter from parameters section:
            <xrange>
                <min>return 0;</min>
                <max>return 30;</max>
            </xrange>
                                    or
            <xrange>
                <min>return -a;</min>
                <max>return a;</max>
            </xrange>

        All functions will be calculated over domain between xrange:min
        and xrange:max. Xrange depending on parameter is extremely
        useful when domain(s) of your function(s) depends on parameter
        (like circle, when parameter is radius and you want to allow
        to change it).
```

Optional parameters:

```
num_points: Number of data points to generated for the plot. If
            this is not set, the number of points will be
            calculated as width / 5.

bar_width: If functions are present which are to be plotted as bars,
           then this parameter specifies the width of the bars. A
           numeric value for this parameter is expected.
```

```
bar_align: If functions are present which are to be plotted as bars,
           then this parameter specifies how to align the bars relative
           to the tick. Available values are "left" and "center".

xticks,
yticks:    3 floating point numbers separated by commas. This
           specifies how many ticks are created, what number they
           start at, and what number they end at. This is different
           from the 'xrange' setting in that it has nothing to do
           with the data points - it control what area of the
           Cartesian space you will see. The first number is the
           first tick's value, the second number is the step
           between each tick, the third number is the value of the
           last tick. If these configurations are not specified,
           Flot will chose them for you based on the data points
           set that he is currently plotting. Usually, this results
           in a nice graph, however, sometimes you need to fine
           grain the controls. For example, when you want to show
           a fixed area of the Cartesian space, even when the data
           set changes. On it's own, Flot will recalculate the
           ticks, which will result in a different graph each time.
           By specifying the xticks, yticks configurations, only
           the plotted data will change - the axes (ticks) will
           remain as you have defined them.

xticks_names, yticks_names:
          A JSON string which represents a mapping of xticks, yticks
          values to some defined strings. If specified, the graph will
          not have any xticks, yticks except those for which a string
          value has been defined in the JSON string. Note that the
          matching will be string-based and not numeric. I.e. if a tick
          value was "3.70" before, then inside the JSON there should be
          a mapping like {..., "3.70": "Some string", ...}. Example:

              <xticks_names>
              <![CDATA[
              {
                  "1": "Treated", "2": "Not Treated",
                  "4": "Treated", "5": "Not Treated",
                  "7": "Treated", "8": "Not Treated"
              }
              ]]>
              </xticks_names>

              <yticks_names>
              <![CDATA[
                  {"0": "0%", "10": "10%", "20": "20%", "30": "30%", "40": "40%", "50": "50%"}
              ]]>
              </yticks_names>

xunits,
yunits:   Units values to be set on axes. Use MathJax. Example:
            <xunits>\(cm\)</xunits>
            <yunits>\(m\)</yunits>

moving_label:
          A way to specify a label that should be positioned dynamically,
          based on the values of some parameters, or some other factors.
          It is similar to a <function>, but it is only valid for a plot
          because it is drawn relative to the plot coordinate system.

          Multiple "moving_label" configurations can be provided, each one
          with a unique text and a unique set of functions that determine
          it’s dynamic positioning.

          Each "moving_label" can have a "color" attribute (CSS color notation),
          and a "weight" attribute. "weight" can be one of "normal" or "bold",
          and determines the styling of moving label’s text.

          Each "moving_label" function should return an object with a ’x’
          and ’y properties. Within those functions, all of the parameter
          names along with their value are available.

          Example (note that "return" statement is missing; it will be automatically
          inserted by GST):

              <moving_label text="Co" weight="bold" color="red">
              <![CDATA[  {’x’: -50, ’y’: c0};]]>
              </moving_label>

asymptote:
          Add a vertical or horizontal asymptote to the graph which will
          be dynamically repositioned based on the specified function.

          It is similar to the function in that it provides a JavaScript body function
          string. This function will be used to calculate the position of the asymptote
          relative to the axis specified by the "type" parameter.

          Required parameters:
              type:
                    Which axis should the asymptote be plotted against. Available values
                    are "x" and "y".

          Optional parameters:
              color:
                    The color of the line. A valid CSS color string is expected.
```

### Example

#### Plotting, sliders and inputs

```
<vertical>
    <graphical_slider_tool>
      <render>

          <h2>Graphic slider tool: full example.</h2>
            <p>
                A simple equation
                    \(
                        y_1 = 10 \times b \times \frac{sin(a \times x) \times sin(b \times x)}{cos(b
                    \)
                can be plotted.
            </p>
```

```
        <!-- make text and input or slider at the same line -->
        <div>
          <p style="float:left;"> Currently \(a\) is</p>
          <!-- readonly input for a -->
          <span id="a_readonly" style="width:50px; float:left; margin-left:10px;"/>
        </div>
        <!-- clear:left will make next text to begin from new line -->
        <p style="clear:left">   This one
              \(
                  y_2 = sin(a \times x)
              \)
          will be overlayed on top.
        </p>
        <div>
          <p style="float:left;">Currently \(b\) is </p>
          <textbox var="b" style="width:50px; float:left; margin-left:10px;"/>
        </div>
        <div style="clear:left;">
          <p style="float:left;">To change \(a\) use:</p>
          <slider var="a" style="width:400px;float:left;margin-left:10px;"/>
        </div>
        <div style="clear:left;">
          <p style="float:left;">To change \(b\) use:</p>
          <slider var="b" style="width:400px;float:left;margin-left:10px;"/>
        </div>
        <plot style='clear:left;width:600px;padding-top:15px;padding-bottom:20px;'/>
        <div style="clear:left;height:50px;">
         <p style="float:left;">Second input for b:</p>
         <!-- editable input for b -->
          <textbox var="b" style="color:red;width:60px;float:left;margin-left:10px;"/>
        </div >
  </render>

  <configuration>

    <parameters>
        <param var="a" min="90" max="120" step="10" initial="100" />
        <param var="b" min="120" max="200" step="2.3" initial="120" />
    </parameters>

    <functions>

      <function color="#0000FF" line="false" dot="true" label="\(y_1\)">
        return 10.0 * b * Math.sin(a * x) * Math.sin(b * x) / (Math.cos(b * x) + 10);
      </function>
      <function color="red" line="true" dot="false" label="\(y_2\)">
        <!-- works w/o return, if function is single line -->
        Math.sin(a * x);
      </function>
      <function color="#FFFF00" line="false" dot="false" label="unknown">
        function helperFunc(c1) {
          return c1 * c1 - a;
        }

        return helperFunc(x + 10 * a * b) + Math.sin(a - x);
      </function>
      <function output="element" el_id="a_readonly">a</function>
    </functions>
```

```
        <plot>

          <xrange>
              <min>return 0;</min>
              <!-- works w/o return -->
              <max>30</max>
          </xrange>

          <num_points>120</num_points>

          <xticks>0, 3, 30</xticks>
          <yticks>-1.5, 1.5, 13.5</yticks>

          <xunits>\(cm\)</xunits>
          <yunits>\(m\)</yunits>
        </plot>
      </configuration>
    </graphical_slider_tool>
</vertical>
```

### Update of html elements, no plotting

```
<vertical>
    <graphical_slider_tool>
        <render>
            <h2>Graphic slider tool: Output to DOM element.</h2>

            <p>a + b = <span id="answer_span_1"></span></p>

            <div style="clear:both">
                <p style="float:left;margin-right:10px;">a</p>
                <slider var='a' style="width:400px;float:left;"/>
                <textbox var='a' style="width:50px;float:left;margin-left:15px;"/>
            </div>

            <div style="clear:both">
                <p style="float:left;margin-right:10px;">b</p>
                <slider var='b' style="width:400px;float:left;"/>
                <textbox var='b' style="width:50px;float:left;margin-left:15px;"/>
            </div>
            <br/><br/><br/>
            <plot/>
        </render>
        <configuration>
            <parameters>
                <param var="a" min="-10.0" max="10.0" step="0.1" initial="0" />
                <param var="b" min="-10.0" max="10.0" step="0.1" initial="0" />
            </parameters>

            <functions>
                <function output="element" el_id="answer_span_1">
                    function add(a, b, precision) {
                        var x = Math.pow(10, precision || 2);
                        return (Math.round(a * x) + Math.round(b * x)) / x;
                    }
                    return add(a, b, 5);
                </function>
            </functions>
        </configuration>
    </graphical_slider_tool>
</vertical>
```

### Circle with dynamic radius

```
<vertical>
    <graphical_slider_tool>
      <render>
        <h2>Graphic slider tool: Dynamic range and implicit functions.</h2>

        <p>You can make x range (not ticks of x axis) of functions to depend on
          parameter value. This can be useful when function domain depends
          on parameter.</p>
        <p>Also implicit functons like circle can be plotted as 2 separate
            functions of same color.</p>
         <div style="height:50px;">
         <slider var='a' style="width:400px;float:left;"/>
         <textbox var='a' style="float:left;width:60px;margin-left:15px;"/>
       </div>
        <plot style="margin-top:15px;margin-bottom:15px;"/>
      </render>
      <configuration>
        <parameters>
            <param var="a" min="5" max="25" step="0.5" initial="12.5" />
        </parameters>
        <functions>
          <function color="red">Math.sqrt(a * a - x * x)</function>
          <function color="red">-Math.sqrt(a * a - x * x)</function>
        </functions>
        <plot>
          <xrange>
            <!-- dynamic range -->
              <min>-a</min>
              <max>a</max>
          </xrange>
          <num_points>1000</num_points>
          <xticks>-30, 6, 30</xticks>
          <yticks>-30, 6, 30</yticks>
        </plot>
      </configuration>
    </graphical_slider_tool>
</vertical>
```

### Example of a bar graph

```
<vertical>
    <graphical_slider_tool>
      <render>
        <h2>Graphic slider tool: Bar graph example.</h2>

        <p>We can request the API to plot us a bar graph.</p>
        <div style="clear:both">
          <p style="width:60px;float:left;">a</p>
          <slider var='a' style="width:400px;float:left;"/>
          <textbox var='a' style="width:50px;float:left;margin-left:15px;"/>
          <br /><br /><br />
          <p style="width:60px;float:left;">b</p>
          <slider var='b' style="width:400px;float:left;"/>
          <textbox var='b' style="width:50px;float:left;margin-left:15px;"/>
        </div>
          <plot style="clear:left;"/>
      </render>
      <configuration>
        <parameters>
            <param var="a" min="-100" max="100" step="5" initial="25" />
            <param var="b" min="-100" max="100" step="5" initial="50" />
        </parameters>
        <functions>
          <function bar="true" color="blue" label="Men">
            <![CDATA[if (((x>0.9) && (x<1.1)) || ((x>4.9) && (x<5.1))) { return Math.sin(a * 0.01 * M
            else {return undefined;}]]>
          </function>
          <function bar="true" color="red" label="Women">
            <![CDATA[if (((x>1.9) && (x<2.1)) || ((x>3.9) && (x<4.1))) { return Math.cos(b * 0.01 * M
            else {return undefined;}]]>
          </function>
          <function bar="true" color="green" label="Other 1">
            <![CDATA[if (((x>1.9) && (x<2.1)) || ((x>3.9) && (x<4.1))) { return Math.cos((b - 10 * a)
            else {return undefined;}]]>
          </function>
          <function bar="true" color="yellow" label="Other 2">
            <![CDATA[if (((x>1.9) && (x<2.1)) || ((x>3.9) && (x<4.1))) { return Math.cos((b + 7 * a)
            else {return undefined;}]]>
          </function>
        </functions>
        <plot>
          <xrange><min>1</min><max>5</max></xrange>
          <num_points>5</num_points>
          <xticks>0, 0.5, 6</xticks>
          <yticks>-1.5, 0.1, 1.5</yticks>
          <xticks_names>
          <![CDATA[
              {
                  "1.5": "Single", "4.5": "Married"
              }
          ]]>
          </xticks_names>
          <yticks_names>
          <![CDATA[
              {
                  "-1.0": "-100%", "-0.5": "-50%", "0.0": "0%", "0.5": "50%", "1.0": "100%"
              }
          ]]>
          </yticks_names>
          <bar_width>0.4</bar_width>
        </plot>
      </configuration>
    </graphical_slider_tool>
</vertical>
```

#### Example of moving labels of graph

```
<vertical>
 <graphical_slider_tool>
      <render>
        <h1>Graphic slider tool: Dynamic labels.</h1>
        <p>There are two kinds of dynamic lables.
        1) Dynamic changing values in graph legends.
        2) Dynamic labels, which coordinates depend on parameters </p>
         <p>a: <slider var="a"/></p>
          <br/>
          <p>b: <slider var="b"/></p>
          <br/><br/>
          <plot style="width:400px; height:400px;"/>
      </render>

      <configuration>
        <parameters>
            <param var="a" min="-10" max="10" step="1" initial="0" />
            <param var="b" min="0" max="10" step="0.5" initial="5" />
        </parameters>
        <functions>
          <function label="Value of a is: dyn_val_1">a * x + b</function>
         <!-- dynamic values in legend -->
         <function output="plot_label" el_id="dyn_val_1">a</function>
        </functions>
        <plot>
          <xrange><min>0</min><max>30</max></xrange>
          <num_points>10</num_points>
          <xticks>0, 6, 30</xticks>
          <yticks>-9, 1, 9</yticks>
          <!-- custom labels with coordinates as any function of parameter -->
          <moving_label text="Dynam_lbl 1" weight="bold">
          <![CDATA[  {'x': 10, 'y': a};]]>
          </moving_label>
          <moving_label text="Dynam lbl 2" weight="bold">
          <![CDATA[  {'x': -6, 'y': b};]]>
          </moving_label>
        </plot>
      </configuration>
    </graphical_slider_tool>
</vertical>
```

### 1.3.3 Xml format of poll module [xmodule]

#### Format description

The main tag of Poll module input is:

**`<poll_question>`** `...` **`</poll_question>`**

`poll_question` can include any number of the following tags: any xml and `answer` tag. All inner xml, except for `answer` tags, we call "question".

#### poll_question tag

Xmodule for creating poll functionality - voting system. The following attributes can be specified for this tag:

```
name - Name of xmodule.
[display_name| AUTOGENERATE] - Display name of xmodule. When this attribute is not defined - display
[reset | False] - Can reset/revote many time (value = True/False)
```

#### answer tag

Define one of the possible answer for poll module. The following attributes can be specified for this tag:

```
id - unique identifier (using to identify the different answers)
```

Inner text - Display text for answer choice.

### Example

#### Examples of poll

```
<poll_question name="second_question" display_name="Second question">
    <h3>Age</h3>
    <p>How old are you?</p>
    <answer id="less18">&lt; 18</answer>
    <answer id="10_25">from 10 to 25</answer>
    <answer id="more25">&gt; 25</answer>
</poll_question>
```

#### Examples of poll with unable reset functionality

```
<poll_question name="first_question_with_reset" display_name="First question with reset"
    reset="True">
    <h3>Your gender</h3>
    <p>You are man or woman?</p>
    <answer id="man">Man</answer>
    <answer id="woman">Woman</answer>
</poll_question>
```

## 1.3.4 Xml format of conditional module [xmodule]

### Format description

The main tag of Conditional module input is:

```
<conditional> ... </conditional>
```

`conditional` can include any number of any xmodule tags (`html`, `video`, `poll`, etc.) or `show` tags.

#### conditional tag

The main container for a single instance of Conditional module. The following attributes can be specified for this tag:

```
sources - location id of required modules, separated by ';'
[message | ""] - message for case, where one or more are not passed. Here you can use variable {link

[submitted] - map to 'is_submitted' module method.
(pressing RESET button makes this function to return False.)

[correct] - map to 'is_correct' module method
[attempted] - map to 'is_attempted' module method
[poll_answer] - map to 'poll_answer' module attribute
[voted] - map to 'voted' module attribute
```

#### show tag

Symlink to some set of xmodules. The following attributes can be specified for this tag:

```
sources - location id of modules, separated by ';'
```

### Example

#### Examples of conditional depends on poll

```
<conditional sources="i4x://MITx/0.000x/poll_question/first_real_poll_seq_with_reset" poll_answer="ma
message="{link} must be answered for this to become visible.">
    <html>
        <h2>You see this, cause your vote value for "First question" was "man"</h2>
    </html>
</conditional>
```

#### Examples of conditional depends on poll (use <show> tag)

```
<conditional sources="i4x://MITx/0.000x/poll_question/first_real_poll_seq_with_reset" poll_answer="ma
message="{link} must be answered for this to become visible.">
    <html>
        <show sources="i4x://MITx/0.000x/problem/test_1; i4x://MITx/0.000x/Video/Avi_resources; i4x:/
    </html>
</conditional>
```

#### Examples of conditional depends on problem

```
<conditional sources="i4x://MITx/0.000x/problem/Conditional:lec27_Q1" attempted="True">
    <html display_name="HTML for attempted problem">You see this, cause "lec27_Q1" is attempted.</htm
</conditional>
<conditional sources="i4x://MITx/0.000x/problem/Conditional:lec27_Q1" attempted="False">
    <html display_name="HTML for not attempted problem">You see this, cause "lec27_Q1" is not attempt
</conditional>
```

### 1.3.5 Xml format of “Word Cloud” module [xmodule]

#### Format description

The main tag of Word Cloud module input is:

```
<word_cloud />
```

The following attributes can be specified for this tag:

```
[display_name| AUTOGENERATE] - Display name of xmodule. When this attribute is not defined - display
[num_inputs| 5] - Number of inputs.
[num_top_words| 250] - Number of max words, which will be displayed.
[display_student_percents| True] - Display usage percents for each word on the same line together wit
```

**Note:** Percent is shown always when mouse over the word in cloud.

**Note:** Possible answer for boolean type attributes: True – “True”, “true”, “T”, “t”, “1” False – “False”, “false”, “F”, “f”, “0”

**Note:** If you want to use the same word cloud (the same storage of words), you must use the same display_name value.

#### Code Example

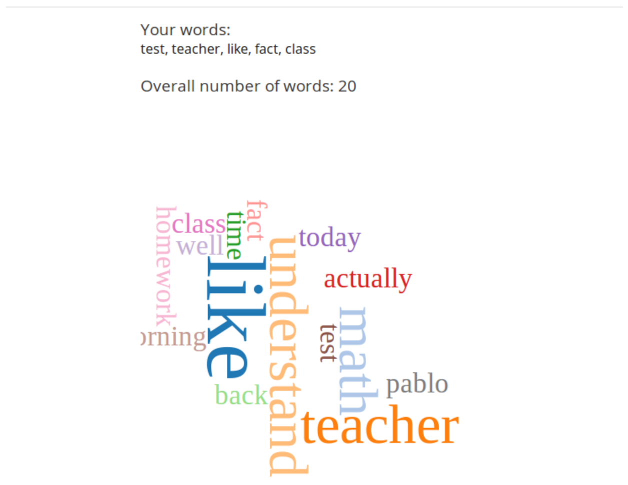
##### Examples of word_cloud without all attributes (all attributes get by default)

```
<word_cloud />
```

##### Examples of poll with all attributes

```
<word_cloud display_name="cloud" num_inputs="10" num_top_words="100" />
```

### Screenshots


Your words:
test, teacher, like, fact, class

Overall number of words: 20

## 1.3.6 CustomResponse XML and Python Script

This document explains how to write a CustomResponse problem. CustomResponse problems execute Python script to check student answers and provide hints.

There are two general ways to create a CustomResponse problem:

### Answer tag format

One format puts the Python code in an `<answer>` tag:

```
<problem>
    <p>What is the sum of 2 and 3?</p>

    <customresponse expect="5">
    <textline math="1" />
    </customresponse>

    <answer>
# Python script goes here
    </answer>
</problem>
```

**The Python script interacts with these variables in the global context:**

- `answers`: An ordered list of answers the student provided. For example, if the student answered `6`, then `answers[0]` would equal `6`.
- `expect`: The value of the `expect` attribute of `<customresponse>` (if provided).
- `correct`: An ordered list of strings indicating whether the student answered the question correctly. Valid values are `"correct"`, `"incorrect"`, and `"unknown"`. You can set these values in the script.
- `messages`: An ordered list of message strings that will be displayed beneath each input. You can use this to provide hints to users. For example `messages[0] = "The capital of California is Sacramento"` would display that message beneath the first input of the response.

- overall_message: A string that will be displayed beneath the entire problem. You can use this to provide a hint that applies to the entire problem rather than a particular input.

Example of a checking script:

```
if answers[0] == expect:
    correct[0] = 'correct'
    overall_message = 'Good job!'
else:
    correct[0] = 'incorrect'
    messages[0] = 'This answer is incorrect'
    overall_message = 'Please try again'
```

**Important**: Python is picky about indentation. Within the <answer> tag, you must begin your script with no indentation.

### Script tag format

The other way to create a CustomResponse is to put a “checking function” in a <script> tag, then use the cfn attribute of the <customresponse> tag:

```
<problem>
    <p>What is the sum of 2 and 3?</p>

    <customresponse cfn="check_func" expect="5">
    <textline math="1" />
    </customresponse>

    <script type="loncapa/python">
def check_func(expect, ans):
    # Python script goes here
    </script>
</problem>
```

**Important**: Python is picky about indentation. Within the <script> tag, the def check_func(expect, ans): line must have no indentation.

**The check function accepts two arguments:**

- expect is the value of the expect attribute of <customresponse> (if provided)
- answer is either:
  - The value of the answer the student provided, if there is only one input.
  - An ordered list of answers the student provided, if there are multiple inputs.

There are several ways that the check function can indicate whether the student succeeded. The check function can return any of the following:

- True: Indicates that the student answered correctly for all inputs.
- False: Indicates that the student answered incorrectly. All inputs will be marked incorrect.
- A dictionary of the form: { 'ok': True, 'msg': 'Message' } If the dictionary’s value for ok is set to True, all inputs are marked correct; if it is set to False, all inputs are marked incorrect. The msg is displayed beneath all inputs, and it may contain XHTML markup.
- A dictionary of the form

```
{ 'overall_message': 'Overall message',
    'input_list': [
        { 'ok': True, 'msg': 'Feedback for input 1'},
        { 'ok': False, 'msg': 'Feedback for input 2'},
        ... ] }
```

The last form is useful for responses that contain multiple inputs. It allows you to provide feedback for each input individually, as well as a message that applies to the entire response.

Example of a checking function:

```
def check_func(expect, answer_given):
    check1 = (int(answer_given[0]) == 1)
    check2 = (int(answer_given[1]) == 2)
    check3 = (int(answer_given[2]) == 3)
    return {'overall_message': 'Overall message',
                'input_list': [
                    { 'ok': check1, 'msg': 'Feedback 1'},
                    { 'ok': check2, 'msg': 'Feedback 2'},
                    { 'ok': check3, 'msg': 'Feedback 3'} ] }
```

The function checks that the user entered `1` for the first input, `2` for the second input, and `3` for the third input. It provides feedback messages for each individual input, as well as a message displayed beneath the entire problem.

## 1.3.7 Symbolic Response

This document plans to document features that the current symbolic response supports. In general it allows the input and validation of math expressions, up to commutativity and some identities.

### Features

**This is a partial list of features, to be revised as we go along:**

- sub and superscripts: an expression following the ^ character indicates exponentiation. To use superscripts in variables, the syntax is `b_x__d` for the variable `b` with subscript `x` and super `d`.

  An example of a problem:

  ```
  <symbolicresponse expect="a_b^c + b_x__d" size="30">
    <textline math="1"
     preprocessorClassName="SymbolicMathjaxPreprocessor"
     preprocessorSrc="/static/js/capa/symbolic_mathjax_preprocessor.js"/>
  </symbolicresponse>
  ```

  It's a bit of a pain to enter that.

- The script-style math variant. What would be outputted in latex if you entered `\mathcal{N}`. This is used in some variables.

  An example:

  ```
  <symbolicresponse expect="scriptN_B + x" size="30">
    <textline math="1"/>
  </symbolicresponse>
  ```

  There is no fancy preprocessing needed, but if you had superscripts or something, you would need to include that part.

## 1.3.8 JS Input

> **NOTE** *Do not use this feature yet! Its attributes and behaviors may change without any concern for backwards compatibility. Moreover, it has only been tested in a very limited context. If you absolutely must, contact Julian (julian@edx.org). When the feature stabilizes, this note will be removed.*

This document explains how to write a JSInput input type. JSInput is meant to allow problem authors to easily turn working standalone HTML files into problems that can be integrated into the edX platform. Since it's aim is flexibility, it can be seen as the input and client-side equivalent of CustomResponse.

A JSInput input creates an iframe into a static HTML page, and passes the return value of author-specified functions to the enclosing response type (generally CustomResponse). JSInput can also stored and retrieve state.

### Format

A jsinput problem looks like this:

```
   <problem>
       <script type="loncapa/python">
def all_true(exp, ans): return ans == "hi"
       </script>
       <customresponse cfn="all_true">
           <jsinput gradefn="gradefn"
               height="500"
               get_statefn="getstate"
               set_statefn="setstate"
               html_file="/static/jsinput.html"/>
       </customresponse>
   </problem>
```

The accepted attributes are:

| Attribute Name | Value Type | Required? | Default |
|---|---|---|---|
| html_file | Url string | Yes | None |
| gradefn | Function name | Yes | *gradefn* |
| set_statefn | Function name | No | None |
| get_statefn | Function name | No | None |
| height | Integer | No | *500* |
| width | Integer | No | *400* |

### Required Attributes

#### html_file

The *html_file* attribute specifies what html file the iframe will point to. This should be located in the content directory.

The iframe is created using the sandbox attribute; while popups, scripts, and pointer locks are allowed, the iframe cannot access its parent's attributes.

The html file should contain an accesible gradefn function. To check whether the gradefn will be accessible to JSInput, check that, in the console,:

`"`gradefn"`

**Returns the right thing. When used by JSInput, *gradefn* is called with::** *gradefn*.call(*obj*)

Where *obj* is the object-part of *gradefn*. For example, if *gradefn* is *myprog.myfn*, JSInput will call *myprog.myfun.call(myprog)*. (This is to ensure "*this*" continues to refer to what *gradefn* expects.)

Aside from that, more or less anything goes. Note that currently there is no support for inheriting css or javascript from the parent (aside from the Chrome-only *seamless* attribute, which is set to true by default).

#### gradefn

The *gradefn* attribute specifies the name of the function that will be called when a user clicks on the "Check" button, and which should return the student's answer. This answer will (unless both the get_statefn and set_statefn attributes are also used) be passed as a string to the enclosing response type. In the customresponse example above, this means cfn will be passed this answer as *ans*.

If the *gradefn* function throws an exception when a student attempts to submit a problem, the submission is aborted, and the student receives a generic alert. The alert can be customised by making the exception name *Waitfor Exception*; in that case, the alert message will be the exception message.

**IMPORTANT** : the *gradefn* function should not be at all asynchronous, since this could result in the student's latest answer not being passed correctly. Moreover, the function should also return promptly, since currently the student has no indication that her answer is being calculated/produced.

### Option Attributes

The *height* and *width* attributes are straightforward: they specify the height and width of the iframe. Both are limited by the enclosing DOM elements, so for instance there is an implicit max-width of around 900.

In the future, JSInput may attempt to make these dimensions match the html file's dimensions (up to the aforementioned limits), but currently it defaults to *500* and *400* for *height* and *width*, respectively.

#### set_statefn

Sometimes a problem author will want information about a student's previous answers ("state") to be saved and reloaded. If the attribute *set_statefn* is used, the function given as its value will be passed the state as a string argument whenever there is a state, and the student returns to a problem. It is the responsibility of the function to then use this state approriately.

The state that is passed is:

1. The previous output of *gradefn* (i.e., the previous answer) if *get_statefn* is not defined.
2. The previous output of *get_statefn* (see below) otherwise.

It is the responsibility of the iframe to do proper verification of the argument that it receives via *set_statefn*.

#### get_statefn

Sometimes the state and the answer are quite different. For instance, a problem that involves using a javascript program that allows the student to alter a molecule may grade based on the molecule's hidrophobicity, but from the hidrophobicity it might be incapable of restoring the state. In that case, a *separate* state may be stored and loaded by *set_statefn*. Note that if *get_statefn* is defined, the answer (i.e., what is passed to the enclosing response type) will be a json string with the following format:

```
{
    answer: `[answer string]`
    state: `[state string]`
}
```

It is the responsibility of the enclosing response type to then parse this as json.

### 1.3.9 Formula Equation Input

Tag: `<formulaequationinput />`

The formula equation input is a math input type used with Numerical and Formula responses only. It is not to be used with Symoblic Response. It is comparable to a `<textline math="1"/>` but with a different means to display the math. It lets the platform validate the student's input as she types.

This is achieved by periodically sending the typed expression and requesting its preview from the LMS. It parses the expression (using the same parser as the parser it uses to eventually evaluate the response for grading), and sends back an OK'd copy.

The basic appearance is that of a textbox with a preview box below it. The student types in math (see note below for syntax), and a typeset preview appears below it. Even complicated math expressions may be entered in.

For more information about the syntax, look in the course author's documentation, under Appendix E, the section about Numerical Responses.

#### Format

The XML is rather simple, it is a `<formulaequationinput />` tag with an optional `size` attribute, which defines the size (i.e. the width) of the input box displayed to students for typing their math expression. Unlike `<textline />`, there is no `math` attribute and adding such will have no effect.

To see an example of the input type in context:

```
<problem>
  <p>What base is the decimal numeral system in?</p>
  <numericalresponse answer="10">
    <formulaequationinput />
  </numericalresponse>

  <p>Write an expression for the product of R_1, R_2, and the inverse of R_3.</p>
  <formularesponse type="ci" samples="R_1,R_2,R_3@1,2,3:3,4,5#10" answer="R_1*R_2/R_3">
    <responseparam type="tolerance" default="0.00001"/>
    <formulaequationinput size="40" />
  </formularesponse>
</problem>
```

CHAPTER

TWO

# INTERNAL DATA FORMATS

These document describe how we store course structure, student state/progress, and events internally. Useful for developers or researchers who interact with our raw data exports.

## 2.1 Student Info and Progress Data

The following sections detail how edX stores student state data internally, and is useful for developers and researchers who are examining database exports. This information includes demographic information collected at signup, course enrollment, course progress, and certificate status.

Conventions to keep in mind:

- We currently use MySQL 5.1 with InnoDB tables
- All strings are stored as UTF-8.
- All datetimes are stored as UTC.
- Tables that are built into the Django framework are not documented here unless we use them in unconventional ways.

All of our tables will be described below, first in summary form with field types and constraints, and then with a detailed explanation of each field. For those not familiar with the MySQL schema terminology in the table summaries:

***Type*** This is the kind of data it is, along with the size of the field. When a numeric field has a length specified, it just means that's how many digits we want displayed – it has no affect on the number of bytes used.

| Value | Meaning |
|---|---|
| *int* | 4 byte integer. |
| *smallint* | 2 byte integer, sometimes used for enumerated values. |
| *tinyint* | 1 byte integer, but usually just used to indicate a boolean field with 0 = False and 1 = True. |
| *varchar* | String, typically short and indexable. The length is the number of chars, not bytes (so unicode friendly). |
| *longtext* | A long block of text, usually not indexed. |
| *date* | Date |
| *date-time* | Datetime in UTC, precision in seconds. |

*Null*

| Value | Meaning |
|---|---|
| *YES* | *NULL* values are allowed |
| *NO* | *NULL* values are not allowed |

**Note:** Django often just places blank strings instead of NULL when it wants to indicate a text value is optional. This is used more meaningful for numeric and date fields.

| | Value | Meaning |
|---|---|---|
| ***Key*** | *PRI* | Primary key for the table, usually named *id*, unique |
| | *UNI* | Unique |
| | *MUL* | Indexed for fast lookup, but the same value can appear multiple times. A Unique index that allows *NULL* can also show up as *MUL*. |

### 2.1.1 User Information

#### *auth_user*

The *auth_user* table is built into the Django web framework that we use. It holds generic information necessary for basic login and permissions information. It has the following fields:

```
+------------------------------+--------------+------+-----+
| Field                        | Type         | Null | Key |
+------------------------------+--------------+------+-----+
| id                           | int(11)      | NO   | PRI |
| username                     | varchar(30)  | NO   | UNI |
| first_name                   | varchar(30)  | NO   |     | # Never used
| last_name                    | varchar(30)  | NO   |     | # Never used
| email                        | varchar(75)  | NO   | UNI |
| password                     | varchar(128) | NO   |     |
| is_staff                     | tinyint(1)   | NO   |     |
| is_active                    | tinyint(1)   | NO   |     |
| is_superuser                 | tinyint(1)   | NO   |     |
| last_login                   | datetime     | NO   |     |
| date_joined                  | datetime     | NO   |     |
| status                       | varchar(2)   | NO   |     | # No longer used
| email_key                    | varchar(32)  | YES  |     | # No longer used
| avatar_type                  | varchar(1)   | NO   |     | # No longer used
| country                      | varchar(2)   | NO   |     | # No longer used
| show_country                 | tinyint(1)   | NO   |     | # No longer used
| date_of_birth                | date         | YES  |     | # No longer used
| interesting_tags             | longtext     | NO   |     | # No longer used
| ignored_tags                 | longtext     | NO   |     | # No longer used
| email_tag_filter_strategy    | smallint(6)  | NO   |     | # No longer used
| display_tag_filter_strategy  | smallint(6)  | NO   |     | # No longer used
| consecutive_days_visit_count | int(11)      | NO   |     | # No longer used
+------------------------------+--------------+------+-----+
```

#### *id*

Primary key, and the value typically used in URLs that reference the user. A user has the same value for *id* here as they do in the MongoDB database's users collection. Foreign keys referencing *auth_user.id* will often be named *user_id*, but are sometimes named *student_id*.

#### *username*

The unique username for a user in our system. It may contain alphanumeric, _, @, +, . and - characters. The username is the only information that the students give about themselves that we currently expose to other students. We have never allowed people to change their usernames so far, but that's not something we guarantee going forward.

#### *first_name*

**Note:** Not used; we store a user's full name in *auth_userprofile.name* instead.

#### *last_name*

**Note:** Not used; we store a user's full name in *auth_userprofile.name* instead.

#### *email*

Their email address. While Django by default makes this optional, we make it required, since it's the primary mechanism through which people log in. Must be unique to each user. Never shown to other users.

#### *password*

A hashed version of the user's password. Depending on when the password was last set, this will either be a SHA1 hash or PBKDF2 with SHA256 (Django 1.3 uses the former and 1.4 the latter).

#### *is_staff*

This value is *1* if the user is a staff member *of edX* with corresponding elevated privileges that cut across courses. It does not indicate that the person is a member of the course staff for any given course. Generally, users with this flag set to 1 are either edX program managers responsible for course delivery, or edX developers who need access for testing and debugging purposes. People who have *is_staff = 1* get instructor privileges on all courses, along with having additional debug information show up in the instructor tab.

Note that this designation has no bearing with a user's role in the forums, and confers no elevated privileges there.

Most users have a *0* for this value.

#### *is_active*

This value is *1* if the user has clicked on the activation link that was sent to them when they created their account, and *0* otherwise. Users who have *is_active = 0* generally cannot log into the system. However, when users first create their account, they are automatically logged in even though they are not active. This is to let them experience the site immediately without having to check their email. They just get a little banner at the top of their dashboard reminding them to check their email and activate their account when they have time. If they log out, they won't be able to log back in again until they've activated. However, because our sessions last a long time, it is theoretically possible for someone to use the site as a student for days without being "active".

Once *is_active* is set to *1*, the only circumstance where it would be set back to *0* would be if we decide to ban the user (which is very rare, manual operation).

#### *is_superuser*

Value is *1* if the user has admin privileges. Only the earliest developers of the system have this set to *1*, and it's no longer really used in the codebase. Set to 0 for almost everybody.

#### *last_login*

A datetime of the user's last login. Should not be used as a proxy for activity, since people can use the site all the time and go days between logging in and out.

#### *date_joined*

Date that the account was created (NOT when it was activated).

#### *(obsolete fields)*

All the following fields were added by an application called Askbot, a discussion forum package that is no longer part of the system:

- *status*
- *email_key*
- *avatar_type*
- *country*
- *show_country*
- *date_of_birth*
- *interesting_tags*
- *ignored_tags*
- *email_tag_filter_strategy*
- *display_tag_filter_strategy*
- *consecutive_days_visit_count*

Only users who were part of the prototype 6.002x course run in the Spring of 2012 would have any information in these fields. Even with those users, most of this information was never collected. Only the fields that are automatically generated have any values in them, such as tag settings.

These fields are completely unrelated to the discussion forums we currently use, and will eventually be dropped from this table.

### *auth_userprofile*

The *auth_userprofile* table is mostly used to store user demographic information collected during the signup process. We also use it to store certain additional metadata relating to certificates. Every row in this table corresponds to one row in *auth_user*:

```
+--------------------+--------------+------+-----+
| Field              | Type         | Null | Key |
+--------------------+--------------+------+-----+
| id                 | int(11)      | NO   | PRI |
| user_id            | int(11)      | NO   | UNI |
| name               | varchar(255) | NO   | MUL |
| language           | varchar(255) | NO   | MUL | # Prototype course users only
| location           | varchar(255) | NO   | MUL | # Prototype course users only
| meta               | longtext     | NO   |     |
| courseware         | varchar(255) | NO   |     | # No longer used
| gender             | varchar(6)   | YES  | MUL | # Only users signed up after prototype
| mailing_address    | longtext     | YES  |     | # Only users signed up after prototype
| year_of_birth      | int(11)      | YES  | MUL | # Only users signed up after prototype
| level_of_education | varchar(6)   | YES  | MUL | # Only users signed up after prototype
| goals              | longtext     | YES  |     | # Only users signed up after prototype
| allow_certificate  | tinyint(1)   | NO   |     |
+--------------------+--------------+------+-----+
```

There is an important split in demographic data gathered for the students who signed up during the MITx prototype phase in the spring of 2012, and those that signed up afterwards.

#### *id*

Primary key, not referenced anywhere else.

#### *user_id*

A foreign key that maps to *auth_user.id*.

#### *name*

String for a user's full name. We make no constraints on language or breakdown into first/last name. The names are never shown to other students. Foreign students usually enter a romanized version of their names, but not always.

It used to be our policy to require manual approval of name changes to guard the integrity of the certificates. Students would submit a name change request and someone from the team would approve or reject as appropriate. Later, we decided to allow the name changes to take place automatically, but to log previous names in the *meta* field.

#### *language*

User's preferred language, asked during the sign up process for the 6.002x prototype course given in the Spring of 2012. This information stopped being collected after the transition from MITx to edX happened, but we never removed the values from our first group of students. Sometimes written in those languages.

#### *location*

User's location, asked during the sign up process for the 6.002x prototype course given in the Spring of 2012. We weren't specific, so people tended to put the city they were in, though some just specified their country and some got as specific as their street address. Again, sometimes romanized and sometimes written in their native language. Like *language*, we stopped collecting this field when we transitioned from MITx to edX, so it's only available for our first batch of students.

#### *meta*

An optional, freeform text field that stores JSON data. This was a hack to allow us to associate arbitrary metadata with a user. An example of the JSON that can be stored here is:

```
{
  "old_names" : [
 ["David Ormsbee", "I want to add my middle name as well.", "2012-11-15T17:28:12.658126"],
 ["Dave Ormsbee", "Dave's too informal for a certificate.", "2013-02-07T11:15:46.524331"]
  ],
  "old_emails" : [["dormsbee@mitx.mit.edu", "2012-10-18T15:21:41.916389"]],
  "6002x_exit_response" : {
 "rating": ["6"],
 "teach_ee": ["I do not teach EE."],
 "improvement_textbook": ["I'd like to get the full PDF."],
 "future_offerings": ["true"],
 "university_comparison":
   ["This course was <strong>on the same level</strong> as the university class."],
 "improvement_lectures": ["More PowerPoint!"],
 "highest_degree": ["Bachelor's degree."],
 "future_classes": ["true"],
 "future_updates": ["true"],
 "favorite_parts": ["Releases, bug fixes, and askbot."]
  }
}
```

The following are details about this metadata. Please note that the fields described below are found as JSON attributes *inside* the *meta* field, and are *not* separate database fields of their own.

***old_names*** A list of the previous names this user had, and the timestamps at which they submitted a request to change those names. These name change request submissions used to require a staff member to approve it before the name change took effect. This is no longer the case, though we still record their previous names.

Note that the value stored for each entry is the name they had, not the name they requested to get changed to. People often changed their names as the time for certificate generation approached, to replace nicknames with their actual names or correct spelling/punctuation errors.

The timestamps are UTC, like all datetimes stored in our system.

***old_emails*** A list of previous emails this user had, with timestamps of when they changed them, in a format similar to *old_names*. There was never an approval process for this.

The timestamps are UTC, like all datetimes stored in our system.

***6002x_exit_response*** Answers to a survey that was sent to students after the prototype 6.002x course in the Spring of 2012. The questions and number of questions were randomly selected to measure how much survey length affected response rate. Only students from this course have this field.

#### *courseware*

This can be ignored. At one point, it was part of a way to do A/B tests, but it has not been used for anything meaningful since the conclusion of the prototype course in the spring of 2012.

#### *gender*

Dropdown field collected during student signup. We only started collecting this information after the transition from MITx to edX, so prototype course students will have *NULL* for this field.

| Value | Meaning |
|---|---|
| *NULL* | This student signed up before this information was collected |
| '' (blank) | User did not specify gender |
| *'f'* | Female |
| *'m'* | Male |
| *'o'* | Other |

#### *mailing_address*

Text field collected during student signup. We only started collecting this information after the transition from MITx to edX, so prototype course students will have *NULL* for this field. Students who elected not to enter anything will have a blank string.

#### *year_of_birth*

Dropdown field collected during student signup. We only started collecting this information after the transition from MITx to edX, so prototype course students will have *NULL* for this field. Students who decided not to fill this in will also have NULL.

#### *level_of_education*

Dropdown field collected during student signup. We only started collecting this information after the transition from MITx to edX, so prototype course students will have *NULL* for this field.

| Value | Meaning |
|---|---|
| *NULL* | This student signed up before this information was collected |
| '' (blank) | User did not specify level of education. |
| *'p'* | Doctorate |
| *'p_se'* | Doctorate in science or engineering (no longer used) |
| *'p_oth'* | Doctorate in another field (no longer used) |
| *'m'* | Master's or professional degree |
| *'b'* | Bachelor's degree |
| *'a'* | Associate's degree |
| *'hs'* | Secondary/high school |
| *'jhs'* | Junior secondary/junior high/middle school |
| *'el'* | Elementary/primary school |
| *'none'* | None |
| *'other'* | Other |

#### *goals*

Text field collected during student signup in response to the prompt, "Goals in signing up for edX". We only started collecting this information after the transition from MITx to edX, so prototype course students will have *NULL* for this field. Students who elected not to enter anything will have a blank string.

#### *allow_certificate*

Set to *1* for most students. This field is set to *0* if log analysis has revealed that this student is accessing our site from a country that the US has an embargo against. At this time, we do not issue certificates to students from those countries.

### *student_courseenrollment*

A row in this table represents a student's enrollment for a particular course run. If they decide to unenroll in the course, we set *is_active* to *False*. We still leave all their state in *courseware_studentmodule* untouched, so they will not lose courseware state if they unenroll and reenroll.

#### *id*

Primary key.

#### *user_id*

Student's ID in *auth_user.id*

#### *course_id*

The ID of the course run they're enrolling in (e.g. *MITx/6.002x/2012_Fall*). You can get this from the URL when you're viewing courseware on your browser.

#### *created*

Datetime of enrollment, UTC.

#### *is_active*

Boolean indicating whether this enrollment is active. If an enrollment is not active, a student is not enrolled in that course. This lets us unenroll students without losing a record of what courses they were enrolled in previously. This was introduced in the 2013-08-20 release. Before this release, unenrolling a student simply deleted the row in *student_courseenrollment*.

#### *mode*

String indicating what kind of enrollment this was. The default is "honor" (honor certificate) and all enrollments prior to 2013-08-20 will be of that type. Other types being considered are "audit" and "verified_id".

### 2.1.2 Courseware Progress

Any piece of content in the courseware can store state and score in the *courseware_studentmodule* table. Grades and the user Progress page are generated by doing a walk of the course contents, searching for graded items, looking up a student's entries for those items in *courseware_studentmodule* via *(course_id, student_id, module_id)*, and then applying the grade weighting found in the course policy and grading policy files. Course policy files determine how much weight one problem has relative to another, and grading policy files determine how much categories of problems are weighted (e.g. HW=50%, Final=25%, etc.).

**Warning: Modules might not be what you expect!**
It's important to understand what "modules" are in the context of our system, as the terminology can be confusing. For the conventions of this table and many parts of our code, a "module" is a content piece that appears in the courseware. This can be nearly anything that appears when users are in the courseware tab: a video, a piece of HTML, a problem, etc. Modules can also be collections of other modules, such as sequences, verticals (modules stacked together on the same page), weeks, chapters, etc. In fact, the course itself is a top level module that contains all the other contents of the course as children. You can imagine the entire course as a tree with modules at every node.
Modules can store state, but whether and how they do so is up to the implemenation for that particular kind of module. When a user loads page, we look up all the modules they need to render in order to display it, and then we ask the database to look up state for those modules for that user. If there is corresponding entry for that user for a given module, we create a new row and set the state to an empty JSON dictionary.

#### *courseware_studentmodule*

The *courseware_studentmodule* table holds all courseware state for a given user. Every student has a separate row for every piece of content in the course, making this by far our largest table:

```
+-------------+--------------+------+-----+
| Field       | Type         | Null | Key |
+-------------+--------------+------+-----+
| id          | int(11)      | NO   | PRI |
| module_type | varchar(32)  | NO   | MUL |
| module_id   | varchar(255) | NO   | MUL |
| student_id  | int(11)      | NO   | MUL |
| state       | longtext     | YES  |     |
| grade       | double       | YES  | MUL | # problem, selfassessment, and combinedopenended use thi
| created     | datetime     | NO   | MUL |
| modified    | datetime     | NO   | MUL |
| max_grade   | double       | YES  |     | # problem, selfassessment, and combinedopenended use thi
| done        | varchar(8)   | NO   | MUL | # ignore this
| course_id   | varchar(255) | NO   | MUL |
+-------------+--------------+------+-----+
```

#### *id*

Primary key. Rarely used though, since most lookups on this table are searches on the three tuple of *(course_id, student_id, module_id)*.

### *module_type*

| | |
|---|---|
| *chapter* | The top level categories for a course. Each of these is usually labeled as a Week in the courseware, but this is just convention. |
| *combine-dope-nended* | A new module type developed for grading open ended questions via self assessment, peer assessment, and machine learning. |
| *condi-tional* | A new module type recently developed for 8.02x, this allows you to prevent access to certain parts of the courseware if other parts have not been completed first. |
| *course* | The top level course module of which all course content is descended. |
| *problem* | A problem that the user can submit solutions for. We have many different varieties. |
| *problem-set* | A collection of problems and supplementary materials, typically used for homeworks and rendered as a horizontal icon bar in the courseware. Use is inconsistent, and some courses use a *sequential* instead. |
| *selfassess-ment* | Self assessment problems. An early test of the open ended grading system that is not in widespread use yet. Recently deprecated in favor of *combinedopenended*. |
| *sequential* | A collection of videos, problems, and other materials, rendered as a horizontal icon bar in the courseware. |
| *timelimit* | A special module that records the time you start working on a piece of courseware and enforces time limits, used for Pearson exams. This hasn't been completely generalized yet, so is not available for widespread use. |
| *videose-quence* | A collection of videos, exercise problems, and other materials, rendered as a horizontal icon bar in the courseware. Use is inconsistent, and some courses use a *sequential* instead. |

There's been substantial muddling of our container types, particularly between sequentials, problemsets, and videosequences. In the beginning we only had sequentials, and these ended up being used primarily for two purposes: creating a sequence of lecture videos and exercises for instruction, and creating homework problem sets. The *problemset* and *videosequence* types were created with the hope that our system would have a better semantic understanding of what a sequence actually represented, and could at a later point choose to render them differently to the user if it was appropriate. Due to a variety of reasons, migration over to this has been spotty. They all render the same way at the moment.

### *module_id*

Unique ID for a distinct piece of content in a course, these are recorded as URLs of the form *i4x://{org}/{course_num}/{module_type}/{module_name}*. Having URLs of this form allows us to give content a canonical representation even as we are in a state of transition between backend data stores.

Table 2.1: Breakdown of example module_id: i4x://MITx/3.091x/problemset/Sample_Problems

| Part | Example | Definition |
|---|---|---|
| *i4x://* | | Just a convention we ran with. We had plans for the domain *i4x.org* at one point. |
| *org* | *MITx* | The organization part of the ID, indicating what organization created this piece of content. |
| *course_num* | *3.091x* | The course number this content was created for. Note that there is no run information here, so you can't know what runs of the course this content is being used for from the *module_id* alone; you have to look at the *courseware_studentmodule.course_id* field. |
| *module_type* | *problemset* | The module type, same value as what's in the *courseware_studentmodule.module_type* field. |
| *module_name* | *Sample_Problems* | The name given for this module by the content creators. If the module was not named, the system will generate a name based on the type and a hash of its contents (ex: *selfassessment_03c483062389*). |

### *student_id*

A reference to *auth_user.id*, this is the student that this module state row belongs to.

### *state*

This is a JSON text field where different module types are free to store their state however they wish.

**Container Modules: *course*, *chapter*, *problemset*, *sequential*, *videosequence*** The state for all of these is a JSON dictionary indicating the user's last known position within this container. This is 1-indexed, not 0-indexed, mostly because it went out that way at one point and we didn't want to later break saved navigation state for users.

**Example: *{"position" : 3}*** When this user last interacted with this course/chapter/etc., they had clicked on the third child element. Note that the position is a simple index and not a *module_id*, so if you rearranged the order of the contents, it would not be smart enough to accomodate the changes and would point users to the wrong place.

The hierarchy goes: *course* > *chapter* > *(problemset* | *sequential* | *videosequence)*

***combinedopenended*** TODO: More details to come.

***conditional*** Conditionals don't actually store any state, so this value is always an empty JSON dictionary (*'{}'*). We should probably remove these entries altogether.

***problem*** There are many kinds of problems supported by the system, and they all have different state requirements. Note that one problem can have many different response fields. If a problem generates a random circuit and asks five questions about it, then all of that is stored in one row in *courseware_studentmodule*.

TODO: Write out different problem types and their state.

***selfassessment*** TODO: More details to come.

***timelimit*** This very uncommon type was only used in one Pearson exam for one course, and the format may change significantly in the future. It is currently a JSON dictionary with fields:

| JSON field | Example | Definition |
|---|---|---|
| *beginning_at* | *1360590255.488154* | UTC time as measured in seconds since UNIX epoch representing when the exam was started. |
| *ending_at* | *1360596632.559758* | UTC time as measured in seconds since UNIX epoch representing the time the exam will close. |
| *accomodation_codes* | *DOUBLE* | (optional) Sometimes students are given more time for accessibility reasons. Possible values are: <br>• *NONE*: no time accommodation <br>• *ADDHALFTIME*: 1.5X normal time allowance <br>• *ADD30MIN*: normal time allowance + 30 minutes <br>• *DOUBLE*: 2X normal time allowance <br>• *TESTING*: extra long period (for testing/debugging) |

### *grade*

Floating point value indicating the total unweighted grade for this problem that the student has scored. Basically how many responses they got right within the problem.

Only *problem* and *selfassessment* types use this field. All other modules set this to *NULL*. Due to a quirk in how rendering is done, *grade* can also be *NULL* for a tenth of a second or so the first time that a user loads a problem. The initial load will trigger two writes, the first of which will set the *grade* to *NULL*, and the second of which will set it to *0*.

### *created*

Datetime when this row was created (i.e. when the student first accessed this piece of content).

### *modified*

Datetime when we last updated this row. Set to be equal to *created* at first. A change in *modified* implies that there was a state change, usually in response to a user action like saving or submitting a problem, or clicking on a navigational element that records its state. However it can also be triggered if the module writes multiple times on its first load, like problems do (see note in *grade*).

### *max_grade*

Floating point value indicating the total possible unweighted grade for this problem, or basically the number of responses that are in this problem. Though in practice it's the same for every entry with the same *module_id*, it is technically possible for it to be anything. The problems are dynamic enough where you could create a random number of responses if you wanted. This a bad idea and will probably cause grading errors, but it is possible.

Another way in which *max_grade* can differ between entries with the same *module_id* is if the problem was modified after the *max_grade* was written and the user never went back to the problem after it was updated. This might happen if a member of the course staff puts out a problem with five parts, realizes that the last part doesn't make sense, and decides to remove it. People who saw and answered it when it had five parts and never came back to it after the changes had been made will have a *max_grade* of *5*, while people who saw it later will have a *max_grade* of *4*.

These complexities in our grading system are a high priority target for refactoring in the near future.

Only *problem* and *selfassessment* types use this field. All other modules set this to *NULL*.

#### *done*

Ignore this field. It was supposed to be an indication whether something was finished, but was never properly used and is just *'na'* in every row.

#### *course_id*

The course that this row applies to, represented in the form org/course/run (ex: *MITx/6.002x/2012_Fall*). The same course content (same *module_id*) can be used in different courses, and a student's state needs to be tracked separately for each course.

## 2.1.3 Certificates

### *certificates_generatedcertificate*

The generatedcertificate table tracks certificate state for students who have been graded after a course completes. Currently the table is only populated when a course ends and a script is run to grade students who have completed the course:

```
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| id            | int(11)      | NO   | PRI | NULL    | auto_increment |
| user_id       | int(11)      | NO   | MUL | NULL    |                |
| download_url  | varchar(128) | NO   |     | NULL    |                |
| grade         | varchar(5)   | NO   |     | NULL    |                |
| course_id     | varchar(255) | NO   | MUL | NULL    |                |
| key           | varchar(32)  | NO   |     | NULL    |                |
| distinction   | tinyint(1)   | NO   |     | NULL    |                |
| status        | varchar(32)  | NO   |     | NULL    |                |
| verify_uuid   | varchar(32)  | NO   |     | NULL    |                |
| download_uuid | varchar(32)  | NO   |     | NULL    |                |
| name          | varchar(255) | NO   |     | NULL    |                |
| created_date  | datetime     | NO   |     | NULL    |                |
| modified_date | datetime     | NO   |     | NULL    |                |
| error_reason  | varchar(512) | NO   |     | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
```

#### *user_id, course_id*

The table is indexed by user and course

#### *status*

Status may be one of these states:

- *unavailable*
- *generating*
- *regenerating*
- *deleting*
- *deleted*
- *downloadable*
- *notpassing*
- *restricted*
- *error*

After a course has been graded and certificates have been issued status will be one of:

- *downloadable*
- *notpassing*
- *restricted*

If the status is *downloadable* then the student passed the course and there will be a certificate available for download.

#### *download_url*

The *download_uuid* has the full URL to the certificate

#### *download_uuid, verify_uuid*

The two uuids are what uniquely identify the download url and the url used to download the certificate.

#### *distinction*

This was used for letters of distinction for 188.1x and is not being used for any current courses

#### *name*

This field records the name of the student that was set at the time the student was graded and the certificate was generated.

#### *grade*

The grade of the student recorded at the time the certificate was generated. This may be different than the current grade since grading is only done once for a course when it ends.

## 2.2 Discussion Forums Data

Discussions in edX are stored in a MongoDB database as collections of JSON documents.

The primary collection holding all posts and comments written by users is *contents*. There are two types of objects stored here, though they share much of the same structure. A *CommentThread* represents a comment that opens a new thread – usually a student question of some sort. A *Comment* is a reply in the conversation started by a *CommentThread*.

### 2.2.1 Shared Attributes

The attributes that *Comment* and *CommentThread* objects share are listed below.

#### *_id*

The 12-byte MongoDB unique ID for this collection. Like all MongoDB IDs, they are monotonically increasing and the first four bytes are a timestamp.

#### *_type*

*CommentThread* or *Comment* depending on the type of object.

#### *anonymous*

If true, this *Comment* or *CommentThread* will show up as written by anonymous, even to those who have moderator privileges in the forums.

#### *anonymous_to_peers*

The idea behind this field was that *anonymous_to_peers = true* would make the the comment appear anonymous to your fellow students, but would allow the course staff to see who you were. However, that was never implemented in the UI, and only *anonymous* is actually used. The *anonymous_to_peers* field is always false.

#### *at_position_list*

No longer used. Child comments (replies) are just sorted by their *created_at* timestamp instead.

#### *author_id*

The user who wrote this. Corresponds to the user IDs we store in our MySQL database as *auth_user.id*

#### *body*

Text of the comment in Markdown. UTF-8 encoded.

#### *course_id*

The full course_id of the course that this comment was made in, including org and run. This value can be seen in the URL when browsing the courseware section. Example: *BerkeleyX/Stat2.1x/2013_Spring*

#### *created_at*

Timestamp in UTC. Example: *ISODate("2013-02-21T03:03:04.587Z")*

#### *updated_at*

Timestamp in UTC. Example: *ISODate("2013-02-21T03:03:04.587Z")*

#### *votes*

Both *CommentThread* and *Comment* objects support voting. *Comment* objects that are replies to other comments still have this attribute, even though there is no way to actually vote on them in the UI. This attribute is a dictionary that has the following inside:

- *up* = list of User IDs that up-voted this comment or thread.
- *down* = list of User IDs that down-voted this comment or thread (no longer used).
- *up_count* = total upvotes received.
- *down_count* = total downvotes received (no longer used).
- *count* = total votes cast.
- *point* = net vote, now always equal to *up_count*.

A user only has one vote per *Comment* or *CommentThread*. Though it's still written to the database, the UI no longer displays an option to downvote anything.

### 2.2.2 CommentThread

The following fields are specific to *CommentThread* objects. Each thread in the forums is represented by one *CommentThread*.

#### *closed*

If true, this thread was closed by a forum moderator/admin.

#### *comment_count*

The number of comment replies in this thread. This includes all replies to replies, but does not include the original comment that started the thread. So if we had:

```
CommentThread: "What's a good breakfast?"
  * Comment: "Just eat cereal!"
  * Comment: "Try a Loco Moco, it's amazing!"
    * Comment: "A Loco Moco? Only if you want a heart attack!"
    * Comment: "But it's worth it! Just get a spam musubi on the side."
```

In that exchange, the *comment_count* for the *CommentThread* is *4*.

#### *commentable_id*

We can attach a discussion to any piece of content in the course, or to top level categories like "General" and "Troubleshooting". When the *commentable_id* is a high level category, it's specified in the course's policy file. When it's a specific content piece (e.g. *600x_l5_p8*, meaning 6.00x, Lecture Sequence 5, Problem 8), it's taken from a discussion module in the course.

#### *last_activity_at*

Timestamp in UTC indicating the last time there was activity in the thread (new posts, edits, etc). Closing the thread does not affect the value in this field.

#### *tags_array*

Meant to be a list of tags that were user definable, but no longer used.

#### *title*

Title of the thread, UTF-8 string.

### 2.2.3 Comment

The following fields are specific to *Comment* objects. A *Comment* is a reply to a *CommentThread* (so an answer to the question), or a reply to another *Comment* (a comment about somebody's answer). It used to be the case that *Comment* replies could nest much more deeply, but we later capped it at just these three levels (question, answer, comment) much in the way that StackOverflow does.

#### *endorsed*

Boolean value, true if a forum moderator or instructor has marked that this *Comment* is a correct answer for whatever question the thread was asking. Exists for *Comments* that are replies to other *Comments*, but in that case *endorsed* is always false because there's no way to endorse such comments through the UI.

#### *comment_thread_id*

What *CommentThread* are we a part of? All *Comment* objects have this.

#### *parent_id*

The *parent_id* is the *_id* of the *Comment* that this comment was made in reply to. Note that this only occurs in a *Comment* that is a reply to another *Comment*; it does not appear in a *Comment* that is a reply to a *CommentThread*.

#### *parent_ids*

The *parent_ids* attribute appears in all *Comment* objects, and contains the *_id* of all ancestor comments. Since the UI now prevents comments from being nested more than one layer deep, it will only ever have at most one element in it. If a *Comment* has no parent, it's an empty list.

## 2.3 Tracking Logs

The following is an inventory of all LMS event types.

This inventory is comprised of a table of Common Fields that appear in all events, a table of Student Event Types which lists all interaction with the LMS outside of the Instructor Dashboard, and a table of Instructor Event Types of all interaction with the Instructor Dashboard in the LMS.

### 2.3.1 Common Fields

This section contains a table of fields common to all events.

| Common Field | Details | Type | Values/Format |
|---|---|---|---|
| `agent` | Browser agent string of the user who triggered the event. | string | |
| `event` | Specifics of the triggered event. | string/JSON | |
| `event_source` | Specifies whether the triggered event originated in the browser or on the server. | string | *'browser'*, *'server'*, *'task'* |
| `event_type` | The type of event triggered. Values depend on `event_source` | string | (see below) |
| `ip` | IP address of the user who triggered the event. | string | |
| `page` | Page user was visiting when the event was fired. | string | *'$URL'* |
| `session` | This key identifies the user's session. May be undefined. | string | 32 digits |
| `time` | Gives the GMT time at which the event was fired. | string | *'YYYY-MM-DDThh:mm:ss.xxxxxx'* |
| `username` | The username of the user who caused the event to fire. This string is empty for anonymous events (i.e., user not logged in). | string | |

### 2.3.2 Event Types

There are two tables of event types – one for student events, and one for instructor events. Table columns describe what each event type represents, which component it originates from, what scripting language was used to fire the event, and what `event` fields are associated with it. The `event_source` field from the "Common Fields" table above distinguishes between events that originated in the browser (in javascript) and events that originated on the server (during the processing of a request).

Event types with several different historical names are enumerated by forward slashes. Rows identical after the second column have been combined, with the corresponding event types enumerated by commas.

## Student Event Types

The Student Event Type table lists the event types logged for interaction with the LMS outside the Instructor Dashboard.

| Event Type |
|---|
| seq_goto new |
| id |
| seq_next new |
| id |
| seq_prev new |
| id |
| problem_check |
| problem_reset |
| problem_show |
| problem_save |
| oe_hide_question / oe_hide_problem peer_grading_hide_question / peer_grading_hide_problem staf |
| oe_show_question / oe_show_problem peer_grading_show_question / peer_grading_show_problem staf |
| rubric_select selection |
| category |
| oe_show_full_feedback oe_show_respond_to_feedback |
| oe_feedback_response_selected |
| page_close |
| play_video code pause_video currentTime |
| speed |
| book *'gotopage', 'prevpage', 'nextpage'* |
| old |
| new |
| showanswer / show_answer |
| problem_check_fail problem_id |
| answers |
| failure |
| problem_check / save_problem_check problem_id |
| answers |
| success |
| attempts |
| correct_map |
| problem_rescore_fail problem_id |
| failure |
| problem_id |
|  |

problem_rescore

| Event Type |
|---|
| orig_score |
| orig_total |
| new_score |
| new_total |
| correct_map |
| success |
| attempts |
| problem_id save_problem_fail |
| failure |
| answers |
| save_problem_success problem_id |
| answers |
| reset_problem_fail problem_id |
| failure |
| reset_problem problem_id |
| new_state |

### *Addendum:* `correct_map` *Fields and Values*

Table of `correct_map` field types and values for the `problem_check` student event type above.

| **`correct_map` field** | **Type** | **Values / Format** | **Null Allowed?** |
|---|---|---|---|
| answer_id | string | | |
| correctness | string | *'correct'*, *'incorrect'* | |
| npoints | integer | Points awarded for this `answer_id`. | yes |
| msg | string | Gives extra message response. | |
| hint | string | Gives optional hint. | yes |
| hintmode | string | None, *'on_request'*, *'always'* | yes |
| queuestate | dict | None when not queued, else *{key:' ', time:' '}* where key is a secret string and time is a string dump of a DateTime object of the form *'%Y%m%d%H%M%S'*. | yes |

### Instructor Event Types

The Instructor Event Type table lists the event types logged for course team interaction with the Instructor Dashboard in the LMS.

| Event Type | Description | Component | Language | `event` Fields | Type | Details |
|---|---|---|---|---|---|---|
| `list-students`, `dump-grades`, `dump-grades-raw`, `dump-grades-csv`, `dump-grades-csv-raw`, `dump-answer-dist-csv`, `dump-graded-assignments-config` | | Instructor Dashboard | Python | | | |
| `rescore-all-submissions`, `reset-all-attempts` | | Instructor Dashboard | Python | `problem` | string | |
| | | | | `course` | string | |
| `delete-student-module-state`, `rescore-student-submission` | | Instructor Dashboard | Python | `problem` | string | |
| | | | | `course` | string | |
| | | | | `student` | string | |
| `reset-student-attempts` | | Instructor Dashboard | Python | `old_attempts` | string | |
| | | | | `student` | string | |
| | | | | `problem` | string | |
| | | | | `instructor` | string | |
| | | | | `course` | string | |
| `get-student-progress-page` | | Instructor Dashboard | Python | `student` | string | |
| | | | | `instructor` | string | |
| | | | | `course` | string | |
| `list-staff`, `list-instructors`, `list-beta-testers` | | Instructor Dashboard | Python | | | |
| `add-instructor`, `remove-instructor` | | Instructor Dashboard | Python | `instructor` | string | |
| `list-forum-admins`, `list-forum-mods`, `list-forum-community-TAs` | | Instructor Dashboard | Python | `course` | string | |
| `remove-forum-admin`, `add-forum-admin`, `remove-forum-mod`, `add-forum-mod`, `remove-forum-community-T` | | Instructor Dashboard | Python | `username` | string | |
| | | | | `course` | string | |
| `psychometrics-histogram-generation` | | Instructor Dashboard | Python | `problem` | string | |
| `add-or-remove-user-group` | | Instructor Dashboard | Python | `event_name` | string | |
| | | | | `user` | string | |
| | | | | `event` | string | |

CHAPTER

THREE

# INDICES AND TABLES

- *search*

# PYTHON MODULE INDEX