Class cursa

Created on Oct 27, 2012

Author: John Roche

The cursa class the main script of the cursa application
It is a facade between the cursa application and the UI layer.
The cursa class should be called to implement domain logic.
Domain classes should not be called directly

class `cursa.`**`Cursa`**

Bases: **`object`**

`build`(*path*, *courseinfo*)

`push`(*path*, *courseinfo*)

`scaffold`(*config*)

Class module

Created on Jan 16, 2013

Author: John Roche

class `dom.module.`**`Module`**(*modulemeta=None*)

Bases: `object`

The Module class is responsible for creating the file structure for a Module and the orchestration of a build of a Module.
A Module consists of a collection of Topics and is responsible for the management of the Topics in the Module.

`_build_topics`()

Orchestrates the build of each Topic in the Module.
This involves invoking the build() method of each topic.

Check the documentation for a description of the Topic.build() method

`_copy_assets`()

Copy CSS and JavaScript files used in the Module.
The files will be copied to the module assets folder

`_create_index`()

Creates the index.html page that will be the

interface to the module. | This method uses the generate module to create the html page

`_create_module_dirs`(*modulename*)

Create the directories for the Module.
If the path has not been declared the path will be created

Args:

- modulename (str): The name of the module

`_create_module_yaml_template`(*modulename*, *title*)

Create a YAML file with a template that contains the default Topic structure

Args:

- module(str): The name of the module
- title (str): The title of the module

`_create_topic_links`()

Creates a HTML Hypertext link for each topic in the module.

The links will be displayed in the index.html page created in topic._create_index().

`_create_topic_objects`()

Creates the Topic objects from the meta data which should have been configured in the index.yaml file

Raises:

CursaException

`_push_topics`()

Orchestrates the build of each Topic in the Module.
This involves invoking the build() method of each topic.

Check the documentation for a description of the Topic.build() method

`build`()

Orchestrates the build of a full Module.
This involves invoking the self.create_topic_objects(), self.build_topics() and self.create_index() methods
Check the documentation for a description of these methods

`push`()

Orchestrates the build of a full Module.
This involves invoking the self.create_topic_objects(), self.build_topics() and self.create_index() methods
Check the documentation for a description of these methods

`scaffold`(*config*)

Orchestrates the scaffolding of the Module.
This involves creating the directories for the Module, and creating a sample YAML file for the Module

Args:

- config (dict): config should have the structure:

```
{
    'name':
    'title':

}
```

  - name: The name of the lab
  - title: The title of the lab

Class topic

Created on Oct 20, 2012

Author: John Roche

class `dom.topic.`**`Topic`**(*topicmeta=None*)

Bases: `object`

The Topic class is responsible for creating the file structure for the topic and the orchestration of a build of a Topic
A Topic consists of a collection of Labs and is responsible for the management of the Labs in the TOPIC.

`_build_labs`()

Orchestrates the build of each lab in the Topic.
This involves invoking the build() method of each lab. Check the documentation for a description of the Lab.build() method
Each lab object has the lab.src attribute set to point to the first html page in lab

`_clean`()

Delete any files from a previous build of the lab

`_copy_assets`()

Copy CSS and JavaScript files used in the Topic.
The files will be copied to the assets subfolder

of the Topic HTML folder.

`_copy_pres_assets`()

Copy CSS and JavaScript files used in the HTML presentations
The files will be copied to the assets subfolder

of the presentation HTML folder.

`_create_index`()

Creates the index.html page that will be the

interface to the topic. | This method uses the generate module to create the html page

`_create_lab_objects`()

Creates the Lab objects from the meta data which should have been configured in the index.yaml file for this topic.

Raises:

CursaException

`_create_pres_info`()

Creates a Markdown string that has the links

to the HTML presentation and the PDF presentation

Returns:
    String containing Markdown links to the HTML presentations and the PDF presentations or an empty string if there are no presentations in the topic

`_create_topic_dirs`(*modulename*, *topicname*, *rich*)
Create the directories for the topic.

`_create_topic_md`()

Creates the index.md file that will be used

to generate the index.html interface to the Topic | This method uses the generate module to create the html page

`_create_topic_yaml_template`(*module*, *topic*, *title*)
Create a YAML file with a template that contains the default Topic structure

Args:

- module(str): The name of the module
- topic (str): The name of the topic
- title (str): The title of the Topic

`_get_pres_css_assets`()

Creates a List of the CSS assets needed for a

presentation

Returns:
    List of CSS assets

`_get_pres_js_assets`()

Creates a List of the Javascript assets needed for a

presentation

Returns:
    List of Javascript assets

`_pres_text`(*pres_meta*)

Creates a PDF presentation of the markdown input
This method uses the generate module to create the PDF presentation

Args:

- pres_meta (dict): Contains the meta data to create the presentation text

_presentation()

Creates a HTML presentation of the markdown input
This method uses the generate module to create the presentation

_push_index()

Sends the Topic outline to Moodle.
The outline is sent as a label resource.
The label content is specified the Topic index.md file

_push_labs()

Orchestrates the push of each lab in the Topic.
This involves invoking the push() method of each lab.

Check the documentation for a description of the Lab.push() method

_push_pdf_presentation(*pres*)

Sends a PDF presentation as a resource to Moodle

_push_presentation(*pres*)

Sends a HTML presentation as a resource to Moodle

_scaffold_labs()

Orchestrates the scaffolding of each lab in the Topic.
This involves invoking the scaffold() method of each lab.

Check the documentation for a description of the Lab.scaffold() method

build()

Orchestrates the build of a full Topic.
This involves invoking the self._create_lab_objects(),

self._build_labs(), self._create_index() and self._presentation methods Check the documentation for a description of these methods

push()

Sends the Topic to a Moodle LMS using a Web service.
The HTML presentation, the PDF version of the presentation and and all the labs are sent.
To use this functionality the clarity plugin for Moodle must be installed in your Moodle site

Raises:
ValueError

`scaffold`(*config*)

Orchestrates the scaffolding of the Topic.
This involves creating the directories for the Topic, and creating a sample YAML file for the Topic

Args:

- config (dict): config should have the structure:

  ```
  {
          'name':
          'title':
          'module':
          'rich':
    }
  ```

  - name: The name of the lab
  - title: The title of the lab
  - module: Which module the lab belongs to
  - rich: Whether the topic index.html will have images and multimedia content

Class Lab

Created on Oct 20, 2012

Author: John Roche

class dom.lab.**Lab**(*labmeta=None*)

Bases: **object**

The Lab class is responsible for creating the file structure for a Lab and the orchestration of a build of a Lab.
A Lab consists of a collection of Steps and is responsible for the management of the Steps in the Lab.

_build_steps(*css*)

Builds each step in the lab.

The outcome of a call to this method should
result in HTML output from the Markdown files

Args:

- css (list): The css to use in the step

_clean()

Delete any files from a previous build of the lab

_copy_assets()

Copy CSS and JavaScript files used in the Lab.
The files will be copied to the assets subfolder

of the Lab HTML folder.

_copy_steps_to_path(*path*)

Copies all the steps in the lab to a new path, along with all CSS,

JS, archives, and images

_create_lab_dirs(*modulename*, *topicname*, *labname*)

Create the directories for the Lab.
If the path has not been declared the path will be created

Args:

- modulename (str): The name of the module
- topicname (str): The name of the topic
- labname (str): The name of the lab

_create_lab_info()

Creates the lab info

Returns (dict):

Lab info containing:

- Markdown from the first step
- Markdown link to the first step
- Markdown link to the pdf

`_create_lab_yaml_template`(*module*, *topic*, *lab*, *title*)

Create a YAML file with a template that contains the default Lab structure

Args:

- module(str): The name of the module
- topic (str): The name of the topic
- lab (str): The name of the lab
- title (str): The title of the lab

`_create_md_templates`()

Create sample markdown files for the Lab.
Files will contain the default template used.
The markdown files will take the names - Exercises.md - Objectives.md - RENAME-STEP.md

and will be stored in the md folder of the Lab directory structure

`_create_nav_bar`()

Creates the HTML Navigation Bar for the Lab. Polls each Step in the Lab for the step list element.

Returns:

String representation of the Navigation Bar or empty string on fail

`_create_steps`()

Creates the Step objects

`_create_text`(*css*)

Creates a PDF file that contains the combined content

of all the steps in a lab | This method uses the generate module to create the PDF file

`_get_mdfile_names`()

Gathers markdown files in the md directory and uses the meta data from the files to create the Step meta data

Note: Method only to be used when creating output using the prefix approach.

Returns:

List (String) containing the markdown file names in the md folder of this lab

`_moodleify_lab`(*path*)

Configures each step in the Lab for being pushed to Moodle
This involves changing local navigation links to navigation links that will work in the Moodle environment
The moodlyify() method of each step in the lab is invoked
Check the documentation for a description of this method

`_pushlabel`()

Sends the label resource to Moodle for this lab
The label is usually the first step in the lab - the objectives.md - which outlines the lab

Raises:
    CursaException

`_pushpdf`()

Sends the PDF version of this lab as a resource to Moodle

`_pushsteps`()

Sends the steps (HTML) version of this lab as a resource to Moodle

including all accompanying files

`build`()

Orchestrates the build of the Lab.
This involves invoking the self._create_steps(), and self._build_steps() methods
Check the documentation for a description of these methods

Returns (dict):
    Lab info containing:

- Markdown from the first step
- Markdown link to the first step
- Markdown link to the pdf

`push`()

Sends the lab to a Moodle LMS using a Web service.
The lab is sent as three resources.
The HTML steps, the PDF and a Label consisting of the objectives Markdown.
To use this functionality the clarity plugin for Moodle must be installed in your Moodle site

Raises:
    ValueError

`scaffold`(*config*)

Orchestrates the scaffolding of the Lab.
This involves creating the directories for the Lab, creating sample markdown files for

the Lab and creating a sample YAML file for the Lab

Args:

- config (dict): config should have the
  structure:

  ```
  {
          'name':
          'title':
          'module':
          'topic':
    }
  ```

  - name: The name of the lab
  - title: The title of the lab
  - module: Which module the lab belongs to
  - topic: Which topic the lab belongs to

Class step

Created on Nov 20, 2012

Author: John Roche

class `dom.step.`**`Step`**(*stepmeta=None*, *filename=None*, *labpath=None*)

Bases: `object`

The Step Class models a step in a Lab.
A Step object models a Markdown file in the Cursa system.
The Step class is responsible for calling the generator module to create the output format.

`_mark_active`(*navBar*)

Changes the class name on the this steps list element to active. This will activate CSS in the HTML output.

args:
navBar (String): Contains the Navigation Bar for the Lab

Returns:
String representation of the Navigation Bar with the list element in the navbar marked as active

`build`(*navBar*, *css*)

The build method uses the generator module to produce the HTML output from the Markdown file that this step models

args:

- navBar (String): Contains the Navigation Bar for the Lab
- css (list): The css to use in the step

`moodleify`(*path*, *topicpath*, *topic*, *courseid*, *sectionid*)

Configures the step for being pushed to Moodle
This involves changing local navigation links to navigation links that will work in the Moodle environment

args:

path (str): The base folder of the lab. This is a temp
folder being used for preparing the steps

topicpath (str): The topic path of the containing Topic

topic (str): The topic name

courseid (str): The course id number

sectionid (str): The section id number

Module generator

Created on Nov 29, 2012

Author: John Roche

The generator module is a facade between the cursa application
and the output generation modules, tools and applications that cursa uses to create the output from the content files

`dom.generator.`**`_create_lab_html`**(*steps*, *labpath*)

Parses the content from each step in the list steps into HTML

and combines the HTML into one String

Args:

- steps (List): Contains the meta data about all steps in
  the lab
- labpath (str): file path to Lab

Returns:

- (str): Generated HTML from Markdown of the combined steps in the
  lab

`dom.generator.`**`_create_pdf`**(*args*, *data*)

Creates a PDF from a string containing HTML

Args:

- args (List): The list of arguments to the wkhtmltopdf
  application
- data (str): The HTML string

`dom.generator.`**`_do_lab_text`**(*labparams*)

Creates a PDF file that contains the combined content

of all the steps in a lab

Args:

- labparams (dict): labparams should have the
  structure:

```
{
    ‘title’:
    ‘steps’:
    ‘labpath’:
    ‘css’:
```

}

- title (str): Name of the PDF file
- steps (List): Contains the meta data about all steps in the lab
- labpath (str): file path to Lab
- css (List): List of CSS assets for the lab

`dom.generator.`**`_do_module`**(*moduleparams*)

Creates a HTML file, index.html, that is the

interface to the module.

Args:

- moduleparams (dict): moduleparams should have the structure:

```
{
    ‘title’:
    ‘modulepath’:
    ‘topiclinks’:

}
```

  - title: Title of the module
  - modulepath: Path to Module
  - topiclinks: Hypertext links to each topic in the module

`dom.generator.`**`_do_pres`**(*topicparams*)

Creates a HTML presentation

Args:

- topicparams (dict): topicparams should have the structure:

```
{
    ‘title’:
    ‘content’:
    ‘prespath’:
    ‘css’:

}
```

  - title of the topic
  - content: The content file name
  - prespath: Path to presentations folder
  - css: The CSS for the presentation

`dom.generator.`**`_do_pres_text`**(*topicparams*)

Creates a PDF version of a HTML presentation

Args:

- topicparams (dict): topicparams should have the structure:

  ```
  {
      'title':
      'content':
      'prespath'

  }
  ```

  - title of the topic
  - content: The content file name
  - prespath: Path to presentations folder
  - css: The CSS for the presentation text

`dom.generator.`**`_do_step`**(*stepparams*)

Creates a HTML file that contains the content

stored in the Markdown file

Args:

- stepparams (dict): stepparams should have the
  structure:

  ```
  {
      'title':
      'labpath':
      'navbar':
      'css' :

  }
  ```

  - title: Name of Markdown file
  - labpath: Path to Lab containing step
  - navbar: Navigation bar for the Lab
  - css: The css for the step

`dom.generator.`**`_do_topic`**(*topicparams*)

Creates a HTML file - index.html - that is the

interface to the topic.

Args:

- topicparams (dict): topicparams should have the structure:

  ```
  {
      'title':
  ```

        ‘content’:
        ‘topicpath’:
        ‘topicinfo’:

    }

- title: Title of the topic,
- content: The content file name,
- topicpath: Path to topic,
- topicinfo: markdown with info about the labs
    and presenations in the topic

`dom.generator.`**`_generate_html`**(*content*)

Generates a HTML string from a Markdown string

Args:

- content (Unicode str): Markdown content

Returns:

- (Unicode str): HTML form of the Markdown content

`dom.generator.`**`_make_imageurls_absolute`**(*content*, *labpath*)

Changes any Markdown image syntax with a relative src

to an absolute src

Note:

e.g.

![](../img/image.png)
will be replaced with
![](absolute/path/to/img/image.png)

Args:

- content (str): The Markdown content

Returns:

- (str): The content with img src set to absolute path

`dom.generator.`**`_parse_markdown`**(*md_location*)

Parses Markdown that will be converted to HTML

Args:

- md_location (str): File name with file path to location of
    the Markdown file

Returns:

- (str): Generated HTML from Markdown

dom.generator.**`_parse_pres_markdown`**(*md*)

Parses Markdown that will be converted to a HTML Presentation

> **Note:**
>
> Cursa Presentation Markdown has new syntax delimiting the start and end of a presentation slide.
> The new delimiting syntax is 3 equal signs, e.g. ===
>
> ##Standard Markdown here
>
> ===
>
> Equal signs are replaced with HTML section tags, e.g.
>
> <section class=”slide”>
>
> <h2>Standard Markdown here</h2>
>
> </section>
>
> **Args:**
>
> - md (str): The markdown content
>
> **Returns:**
>
> - (str): Generated HTML from Markdown

dom.generator.**`_remove_output_dest`**(*output_dest*)

Remove the file at the path

dom.generator.**`_untar_macosx_directory`**()

Before a call to wkhtmltopdf on OSX can be made,

some dependencies need to be extracted from archive. | This is becuse PyInstaller does not allow directories to be bundled

into the executable so an archive of the directory was bundled

dom.generator.**`_write_view_to_file`**(*file_to_create*, *data*)

Creates a file and writes data to that file.
This function uses codecs.open() and ensures unicode compatability. | data must be a string, not Binary data

Args:

- file_to_create (str): File name with file path
- data (str): The data to write to the file

Raises:

CursaException

`dom.generator.`**`generate`**(*stepparams=None*, *labparams=None*, *topicparams=None*, *moduleparams=None*)

This function is the single public method that the system can use to interact the output generation modules.
generate will decide what action to take based on the parameters passed and create the output.

args:

- stepparams (Dict): A dictionary object containing the
  params for the Step.

- labparams (Dict): A dictionary object containing the
  params for the Lab.

- topicparams (Dict): A dictionary object containing the
  params for the Topic. topicparams should have a key named 'category' which indicates which output to create valid entries for category are;

    - 'index'
    - 'text'
    - 'pres_text'
    - 'label'

- moduleparams (Dict): A dictionary object containing the
  params for the Topic.

Module utils

Created on Oct 20, 2012

Author: John Roche

The utils module contains a set of utility functions used in the system.

`dom.utils.`**`clean`**(*path*)

Delete all the files in the directory

Args:

- path (Str): The file path to the directory

Raises:
CursaException

`dom.utils.`**`clean_directories`**(*directories*)

Delete all the files in the list of directories

Args:

- directories (List): A list containing paths to the directories

`dom.utils.`**`copy_files`**(*fileArray*, *dest*)

Creates each file in the filearray to dest

Args:

- fileArray (List): List of files to copy
- dest (str): The destination path to copy the files to

Raises:
CursaException

`dom.utils.`**`create_course_info`**(*path*)

Create the course info for a Module, Topic or Lab,

from the given path | The course info defines what Module, Topic or Lab the

current course section being built belongs to.

The cursa path convention can be used to compute this info

e.g.

Given a path;

/cursahome/modulename/topics/topicname/labs/labname

The course info;

{

module: modulename, topic: topicname, lab: labname

}

can be computed from the path

Args:

- path (str): current working directory

Returns:

- (dict): The course info

dom.utils.**create_file**(*filePath*)

utilty function for creating a file.

args:
filePath (String):
String indicating the file path to create.

Raises:
CursaException

dom.utils.**create_files**(*filePathArray*)

utilty function for creating multiple files.

args:
filePathArray (Array):
String Array containing files paths to create.

Raises:
CursaException

dom.utils.**create_push_params**(*ob*, *resourcetype*, *displayname=''*, *mainfile=''*, *mainfilepath=''*, *labeltext=''*, *labeltextformat=''*)

prepares the params for a push to the Moodle web service

Args:

- ob (Object): The calling object, Module, Topic or Lab
- resourcetype (str): file or label
- displayname (str): The name beside the resource in Moodle
- mainfile (str): The file that will be linked to in Moodle
  This is usually the first step in the lab or the presentation HTML file
- labeltext (str): The content for the label
- labelformat (str): HTML or markdown

Returns:

Dict of params

dom.utils.**create_yaml_template**(*path*, *template*)

utilty function for writing YAML template to a file

args:

path (str):

Path to create the YAML file

template (str):

Template to write to the YAML file

Raises:

CursaException

dom.utils.**delete_files**(*filelist*, *path*)

Delete all the filenames in filelist

Args:

- filelist (List): List containing the names of the files
- path (Str): The file path to the directory

Raises:

CursaException

dom.utils.**delete_tmpdir_and_archive**(*archiveinfo*)

Deletes a temporary directory and an zip archive

Args:

- archiveinfo (Dict): Containing the zip path and temporary directory path

dom.utils.**gather_file_names**(*path*)

Gather all the file names in the given path directory

Args:

- path (str): The file path to the directory

Returns:

- (List): List of file names

dom.utils.**get_css_assets**(*extra_css=None*)

Creates a List of the basic CSS assets needed in a HTML page

Args:

- extra_css (List): List of extra css to add to the page

Returns:

List of CSS assets

`dom.utils.`**`get_img_assets`**(*extra_css*)

`dom.utils.`**`get_js_assets`**()

Creates a List of the basic CSS assets needed in a HTML page

Returns:
List of Javascript assets

`dom.utils.`**`log`**(*msg*)

Stub message for logging
Prints to stdout for now

Args:

- msg (str): A string to print to the user interface

`dom.utils.`**`make_dir`**(*directory*)
utilty function for creating a directory

args:
directory (String):
String indicating the directory path to create.

Raises:
CursaException

`dom.utils.`**`make_dirs`**(*directoryArray*)
utilty function for creating multiple directories

args:
directoryArray (Array):
String Array containing the directory paths to create.

Raises:
CursaException

`dom.utils.`**`make_tmpdir_with_archive`**(*path*)

Creates a temporary directory and an zip archive of the temporary directory

Args:

- path (str): The file path to the directory

Returns:

- (Dict): Containing the zip path and temporary directory path

`dom.utils.`**`read_data_from_file`**(*file_location*)

Reads data from a file.
This function uses codecs.open() and ensures unicode compatability. | Function will only read from text files

Args:

- file_location (str): File name with file path

Returns:

- (str): The data from the file

Raises:
    CursaException

`dom.utils.`**`write_to_file`**(*path*, *data*)

utilty function for creating a file and writing data to the file

args:

path (str):
    Path to create the file

data (str):
    Data to write to the YAML file

Raises:
    CursaException

Class Build

Created on 18 Jan 2013

Author: John Roche

class `ui.build.`**`Build`**

Bases: **`object`**

The Build class is an implementation of a command line interface command in the cursa application.
A command class is responsible for defining the command e.g. which switches and arguments the command can receive from the command line interface.
As a command class it has implemented a static setup(namespace) method and a call(namespace) method. The call(namespace) method must implement the functionality of the command from the command line interface.
The class attributes title (str) and help (str) must also be implemented

`call`(*namespace*)

Implements the functionality of the command from the command line interface. This is the public method that the CommandLine class invokes when instructed to perform a command.

`help` = *'Build from YAML file'*

classmethod **`setup`**(*command_parser*)

Defines the command. This method should define which switches and arguments the command can receive from the command line interface Registers the class with the CommandLine class by setting an attribute in the namespace returned from the parse_args method.

`title` = *'build'*

Class CommandLine

Created on 14 Jan 2013

Author: John Roche

class `ui.cli.`**`CommandLine`**(*namespace=None*, *args=None*)

Bases: **`object`**

The CommandLine class is the class that controls the command line interface which the user interacts with when using the cursa application.
Each command is modelled as a separate class in the cursa application.
The CommandLine class is responsible for registering each command class with the command line instruction and for invoking the call() method of each command class when called from the command line interface.

Each command class must implement a static setup(namespace) method and a call(namespace) method.

`_do_command`(*namespace*)

Invokes the call(namespace) method of the class responsible for handling the command. The call(namespace) method should implement the required actions for the command.

args:

namespace (object): namespace object containing the information

about the command entered at the command line interface. The command attribute contains the class that is responsible for managing that command.

`_setup_commands`()

Registers the command classes with the command line object. Invokes the setup static method of each command class passing a parser object as an argument. The command class can use the parser object to define the command and command arguments.

`commands` = *[<class 'ui.build.Build'>, <class 'ui.scaffold.Scaffold'>, <class 'ui.push.Push'>]*

Class Push

Created on 11 Apr 2013

Author: John Roche

class `ui.push.`**`Push`**

Bases: **`object`**

The Push class is an implementation of a command line interface command in the cursa application.
As a command class it has implemented a static setup(namespace) method and a call(namespace) method. The call(namespace) method must implement the functionality of the command from the command line interface.
As a command class it has implemented a static setup(namespace) method and a call(namespace) method.
The class attributes title (str) and help (str) must also be implemented

`call`(*namespace*)

Implements the functionality of the command from the command line interface. This is the public method that the CommandLine class invokes when instructed to perform a command.

`help` = *'Push a Lab, Topic, or Module to Moodle'*

classmethod **`setup`**(*command_parser*)

Defines the command. This method should define which switches and arguments the command can receive from the command line interface Registers the class with the CommandLine class by setting an attribute in the namespace returned from the parse_args method.

`title` = *'push'*

Class Scaffold

Created on 19 Jan 2013

Author: John Roche

class `ui.scaffold.`**`Scaffold`**

Bases: **`object`**

The Scaffold class is an implementation of a command line interface command in the cursa application.
As a command class it has implemented a static setup(namespace) method and a call(namespace) method. The call(namespace) method must implement the functionality of the command from the command line interface.
As a command class it has implemented a static setup(namespace) method and a call(namespace) method.
The class attributes title (str) and help (str) must also be implemented

`call`(*namespace*)

Implements the functionality of the command from the command line interface. This is the public method that the CommandLine class invokes when instructed to perform a command.

`help` = *'Create file structure and templates\n for the specified module, topic or lab'*

classmethod **`setup`**(*command_parser*)

Defines the command. This method should define which switches and arguments the command can receive from the command line interface Registers the class with the CommandLine class by setting an attribute in the namespace returned from the parse_args method.

`title` = *'scaffold'*

Module view

Created on 2 Feb 2013

Author: John Roche

`dom.view.`**`_create_bottom`**()

`dom.view.`**`_create_content`**(*content*)

`dom.view.`**`_create_step_text_bottom`**()

`dom.view.`**`_create_top`**(*title*)

`dom.view.`**`_modulecss`**()

`dom.view.`**`_modulejs`**()

`dom.view.`**`_pres_text_view`**(*sections*, *css*)

`dom.view.`**`_pres_view`**(*title*, *content*, *css*)

`dom.view.`**`_step_text_css`**(*css*)

`dom.view.`**`_step_text_js`**()

`dom.view.`**`_stepcss`**(*css*)

`dom.view.`**`_stepjs`**()

`dom.view.`**`_topiccss`**()

`dom.view.`**`_topicjs`**()

`dom.view.`**`get_lab_text_view`**(*content*, *css*)

`dom.view.`**`get_module_view`**(*title*, *content*)

`dom.view.`**`get_pres_text_view`**(*sections*, *css*)

`dom.view.`**`get_pres_view`**(*title*, *content*, *css*)

`dom.view.`**`get_step_view`**(*title*, *content*, *navbar*, *css*)

`dom.view.`**`get_topic_view`**(*title*, *content*)

Class WebServiceClient

Created on 11 Apr 2013

Author: John Roche

class `dom.webserviceclient.`**`WebServiceClient`**

Bases: **`object`**

classdocs

`moodleaddress` = *'http://127.0.0.1:8080/moodle/'*

`send_resource`(*webservice_params*, *filepath=None*)

Sends a resource to Moodle.
This function uses the custom clarity plugin

Args:

- webservice_params (dict):

  {

  ‘courseid’: The course id,
  ‘sectionid’: The section id,
  ‘type’: file, label,
  ‘displayname’: The name beside the resource,
  ‘mainfile’: The main file. To be used with folder to create lab or presentation,
  ‘mainfilepath’: The path to the main file from the root of the folder. Must start and end with /,
  ‘labeltext’: The label text,
  ‘labeltextformat’: The label text format. markdown or html,

  }

Raises:

CursaException