

# Planning.Domains

Christian Muise · cjmuise@mit.edu · www.haz.ca · MIT

## API.PLANNING.DOMAINS

v0.2

General query path for GET and POST requests (API\_PATH):

```
GET api.planning.domains/:format/classical
```

:format can be **xml** or **json**. General JSON response format:

```
{ "error": False,
  "message": "Success!",
  "result": ... }
```

Get incumbent plan:

```
GET API_PATH/plan/{prob-id}
```

Submit new plan with parameters:

```
POST API_PATH/submitplan/{prob-id}
```

```
plan  String of IPC-style plan
email User email (for the glory!)
```

## Collections

*Collections represent a set of domains.*

### Collection Object Attributes

collection_id	Unique ID
collection_name	Name of the collection
description	Short description
domain_set	List of domain ID's
tags	List of tags

Get a list of all collections:

```
GET API_PATH/collections
```

Get a particular collection:

```
GET API_PATH/collection/{col-id}
```

Search using the following attributes:

```
GET API_PATH/collections/search?opt1=val1&opt2=val2
```

```
collection_name  Name to match the collection on
tags             Comma separated tag list
```

## Domains

*Domains represent a set of problems.*

### Domain Object Attributes

domain_id	Unique ID
domain_name	Name of the collection
description	Description of domain origin
tags	List of tags

Get a list of all domains:

```
GET API_PATH/domains
```

Get a list of all domains in a collection:

```
GET API_PATH/domains/{col-id}
```

Get a particular domain:

```
GET API_PATH/domain/{dom-id}
```

Search using the following attributes:

```
GET API_PATH/domains/search?opt1=val1&opt2=val2
```

```
domain_name  Name to match the domain on
tags         Comma separated tag list
```

## Problems

*Problems represent a domain / problem PDDL pair*

### Problem Object Attributes

problem_id	ID of this problem
domain_id	ID of this problem's domain
domain	Name of the domain for this problem
problem	Name of problem (e.g., prob1.pddl)
domain_url	Remote location of the domain file
probleme_url	Remote location of the problem file
domain_path	Relative path to the domain file
problem_path	Relative path to the problem file
tags	List of tags for the problem

Additionally, the following statistics are included in a problem object, with `_description` appended for more info:

upper_bound	Upper bound on the optimal plan
lower_bound	Lower bound on the optimal plan
average_effective_width	Average classical planning width
max_effective_width	Maximum classical planning width

Get a list of all problems:

```
GET API_PATH/problems
```

Get a list of all problems in a domain:

```
GET API_PATH/problems/{dom-id}
```

Get a particular problem:

```
GET API_PATH/problem/{prob-id}
```

Search using the following attributes:

```
GET API_PATH/problems/search?opt1=val1&opt2=val2
```

domain	Matches a given domain id
domain_name	Matches on the domain name
problem_name	Matches on the problem name
average_effective_width	Set the average width
max_effective_width	Set the max width
min_lower_bound	Floor on the lower bound
max_lower_bound	Ceiling on the lower bound
min_upper_bound	Floor on the upper bound
max_upper_bound	Ceiling on the upper bound
tags	Comma separated tag list
openbounds	( = [0,1] ) Problems with open or closed lower/upper bounds

As an example, the following finds all problems with an maximum width of 1 (i.e., "easy" to find a solution for), and also have open bounds (i.e., "hard" to prove optimality for):

```
http://api.planning.domains/json/classical/problems/
search?max_effective_width=1&openbounds=1
```

## SOLVER.PLANNING.DOMAINS

The solver can be used to compute plans for classical problems through a JSON interface. Simple tests can also be done by going to the following URL with appropriate paths:

```
http://solver.planning.domains/solve
?domain=<domain url>&problem=<prob url>
```

General JSON POST queries:

POST solver.planning.domains/solve

POST solver.planning.domains/validate

POST solver.planning.domains/solve-and-validate

### JSON Query Parameters

domain	Either URL or raw PDDL for domain
problem	Either URL or raw PDDL for problem
probID	API ID to supersede domain and problem
is_url	Set to <b>true</b> if using URLs
plan	IPC format plan (just for /validate)

Depending on the endpoint, various attributes will be returned. If a plan was computed, the service will attempt to produce a grounding of all plan actions. The returned object will have status set to either **ok** or **error** and result set to either an error message or an object using the following attributes:

### Solved Object Attributes

length	Number of actions
output	Planner output
parse_status	Status of the plan parsing (e.g., <b>ok</b> )
type	Either <b>simple</b> or <b>full</b>
cost	Total plan cost
val_stdout	VAL standard output
val_stderr	VAL standard error
val_status	Either <b>valid</b> or <b>err</b>
error	Indication of any VAL error
plan	List of action objects which are just action names (if type is <b>simple</b> ) or objects with attributes name and action (the latter being a string for the ground action)

## EDITOR.PLANNING.DOMAINS (PLUGINS)

Plugins can be loaded dynamically using only a URL.

### Anatomy of a Plugin (JavaScript file)

```
define(function () {
  return {
    name: "Plan-o-matic 1000",
    author: "John Smith",
    email: "yeah@right.com",
    description: "A plugin template.",

    // Called when loaded or enabled
    initialize: function() { },

    // Called when disabled
    disable: function() { },

    // Used to save settings
    save: function() { return {}; },

    // Restore any previous settings
    load: function(settings) { }
  };
});
```

### Menu Interface

Used to modify the top menu bar

```
window.add_menu(name, id, icon);
  name  Name for the menu
  id    HTML ID for later reference
  icon  Bootstrap glyphicon string1

window.add_menu_button( /*args*/ );
  name  Name for the menu
  id    HTML id for later reference
  icon  Bootstrap glyphicon string1
  cb_string  String of function call (no " permitted)
  parent_menu (optional) ID for parent menu

window.remove_menu_or_button(id);
  id  HTML ID for menu or button
```

## Collection of Plugins

A meta plugin can be used to load other (meta or standard) plugins, along with predefined plugin-specific settings.

### Anatomy of a Meta Plugin

```
define(function () {
  return {
    // Mandatory flag
    meta: true,

    // List of meta / normal plugins
    plugins: {
      'plugin-foo': {
        url: 'http://path.to.plugin/',
        settings: {} },
      // ...
    };
  });
});
```

Generate code snippets for the editor:

```
window.add_snippet(snippet, trigger);
  snippet  Cloud9 style snippet2
  trigger  Text to trigger the auto-complete
```

Create new tabs for custom views:

```
window.new_tab(name, callback);
  name  Name for the new tab
  callback  Function that is called with the new view's
           HTML ID (shown when tab is selected)
```

Inject custom CSS styling:

```
window.inject_styles(css_style);
  css_style  String of CSS to be included
```

<sup>1</sup>: <http://getbootstrap.com/components/#glyphicons>

<sup>2</sup>: <https://cloud9-sdk.readme.io/docs/snippets>