# Titanium Modules Information

### *Actions*


- Requirements
- Description
- Invocation
- Mobile Modules
- Desktop Modules
- Module Installation and Notification Listening
- Sample of use


#### Requirements

The examples in this page use the ***Prototype*** library, which is included by default inside a portal. However, the same concepts may be applied using other implementations.

#### Description

These set of actions can retrieve a Titanium Modules information from the Studio. The actions support both Mobile modules and Desktop modules.

#### Invocation

This command is executed immediately in a synchronous way.

#### Mobile Modules

The following *dispatch* call will get the Titanium Mobile Modules information:

```
modulesInfo = dispatch($H({
  controller : 'portal.titanium.modules',
  action : "getMobileModules"
}).toJSON()).evalJSON();
```

The returned *mobules information* JSON is a Map object that has two roots:

1. **global_modules** - Holds the modules that were installed into the Titanium SDK folder (<sdk-dir>/modules).
2. **project_modules** - Holds the modules that were installed into the opened *Titanium Mobile Projects* in the workspace.

### *Global Modules*

As described above, the *Global Modules* are the modules that were installed into the Titanium SDK modules directory.
The hash value for the **global_modules** key holds an *array* of module-descriptions. Each item in this array is a hash that holds the following information:

| key | description |
|---|---|
| name | The name of the module |
| platforms | The platforms that this module has support for (an array) |
| versions | The versions that this module has support for (an array) |
| type | The module-type ('*global*' or '*project*', and in this case - only '*global*') |

### *Project's Modules*

The Studio collects all the modules information from the *Titanium Mobile Projects* in the workspace. Modules that were located under a *modules* d irectory in a project are considered as installed-modules for that project.
The hash value for the **project_modules** key holds a *hash* that maps from a *project-name* to an array of modules. Each module in that array holds the same information as described above, with the minor difference in the *'type'* field.

### *JSON Structure Illustration*

Here is an illustration that describes the form of the returned JSON object.



### Desktop Modules

The following *dispatch* call will get the Titanium Desktop Modules information:

```
modulesInfo = dispatch($H({
  controller : 'portal.titanium.modules',
  action : "getDesktopModules"
}).toJSON()).evalJSON();
```

The returned *mobules information* JSON is a Map object that holds the *versions* of the installed modules as *keys*, and holds an array of *module-names* as values.
For example:

```
{"1.1.0" : ["php", "ruby", "python", "javascript", "tiapp"...]}
```

### Module Installation and Notification Listening

You can call to install a module by passing it a URL.
For example:

```
var response = dispatch($H({
    controller : 'portal.titanium.modules',
    action : "installModule",
    args : '["' + moduleURL.value + '"]'
}).toJSON());
```

Note that calling this method will eventually trigger a notification from the studio side when the installation is done.
The notification key is ***"modules"***. In the *studio3-sdk* samples, we register a listener to that event like that:

```
eventsDispatcher.addObserver("modules", function(e) {
    portal.modules.update(e);
});
```

### Sample of use

See ***studio3-sdk*** repository (*modules.js*)