

---

# **Ajenti**

***Release 1.2.22.23***

August 09, 2015



<b>1</b>	<b>Documentation</b>	<b>1</b>
1.1	FAQ . . . . .	1
1.2	Installation . . . . .	1
1.3	Running Ajenti . . . . .	3
<b>2</b>	<b>Developers</b>	<b>5</b>
2.1	Getting Started with Plugin Development . . . . .	5
2.2	User Interface . . . . .	8
2.3	Plugin resources . . . . .	14
2.4	Notifications . . . . .	15
2.5	Bindings . . . . .	16
2.6	Custom UI Controls . . . . .	20
2.7	Handling HTTP Requests . . . . .	22
2.8	Dashboard Widgets . . . . .	23
<b>3</b>	<b>API Reference</b>	<b>25</b>
3.1	ajenti . . . . .	25
3.2	ajenti.api . . . . .	25
3.3	ajenti.api.http . . . . .	28
3.4	ajenti.api.sensors . . . . .	28
3.5	ajenti.http . . . . .	28
3.6	ajenti.ipc . . . . .	28
3.7	ajenti.middleware . . . . .	29
3.8	ajenti.plugins . . . . .	29
3.9	ajenti.profiler . . . . .	31
3.10	ajenti.ui . . . . .	31
3.11	ajenti.ui.binder . . . . .	35
3.12	ajenti.users . . . . .	37
3.13	ajenti.util . . . . .	38
<b>4</b>	<b>Plugin API Reference</b>	<b>41</b>
4.1	ajenti.plugins.main.api . . . . .	41
4.2	ajenti.plugins.dashboard.api . . . . .	41
4.3	ajenti.plugins.configurator.api . . . . .	42
4.4	ajenti.plugins.db_common.api . . . . .	42
4.5	ajenti.plugins.webserver_common.api . . . . .	42
4.6	ajenti.plugins.packages.api . . . . .	42
4.7	ajenti.plugins.services.api . . . . .	42

4.8	ajenti.plugins.tasks.api . . . . .	43
<b>5</b>	<b>Indices and tables</b>	<b>45</b>
	<b>Python Module Index</b>	<b>47</b>

## 1.1 FAQ

### 1.1.1 How do I add domains/PHP/email accounts/websites?

Pure Ajenti is a **server control** panel, not a **hosting control** panel. You need the Ajenti V add-on for web-hosting stuff: <http://ajenti.org/#product-ajenti-v>.

### 1.1.2 I forgot my password

Open `/etc/ajenti/config.json`, look for your user entry, and replace whole password hash entry with a new plaintext password. Restart Ajenti. Click “save” under “Configuration” to rehash the password.

### 1.1.3 My OS isn't supported, but I'm a brave adventurer

```
pip install ajenti
```

## 1.2 Installation

### 1.2.1 Debian Packages

Ajenti requires Debian 6 or later. Debian 5 might work with Python 2.6 installed.

Debian Squeeze requires squeeze-backports repository: <http://backports.debian.org/Instructions/>

Add repository key:

```
wget http://repo.ajenti.org/debian/key -O- | apt-key add -
```

Add repository to `/etc/apt/sources.list`:

```
echo "deb http://repo.ajenti.org/debian main main debian" >> /etc/apt/sources.list
```

Install the package:

```
apt-get update && apt-get install ajenti
```

Start the service:

```
service ajenti restart
```

### 1.2.2 Ubuntu Packages

Ajenti requires ubuntu 12.04 Precise Pangolin. Previous releases might work with Python upgraded.

Add repository key:

```
wget http://repo.ajenti.org/debian/key -O- | apt-key add -
```

Add repository to `/etc/apt/sources.list`:

```
echo "deb http://repo.ajenti.org/ng/debian main main ubuntu" >> /etc/apt/sources.list
```

Install the package:

```
apt-get update && apt-get install ajenti
```

Start the service:

```
service ajenti restart
```

### 1.2.3 RPM Packages

Ajenti requires EPEL repositories: <http://fedoraproject.org/wiki/EPEL>

Add repository key:

```
wget http://repo.ajenti.org/ajenti-repo-1.0-1.noarch.rpm  
rpm -i ajenti-repo-1.0-1.noarch.rpm
```

Install the package:

```
yum install ajenti
```

Start the service:

```
service ajenti restart
```

---

**Note:** Package does not match intended download?

```
yum clean metadata
```

---

### 1.2.4 FreeBSD Installation

Prerequisites:

```
cd /usr/ports/devel/py-gevent; make install clean;  
cd /usr/ports/devel/py-lxml; make install clean;  
cd /usr/ports/devel/py-pip; make install clean;  
cd /usr/ports/net/py-ldap2; make install clean;  
cd /usr/ports/security/stunnel; make install clean;
```

Download and install latest Ajenti build from PYPI:

```
pip install ajenti
```

Install rc.d script:

```
wget https://raw.githubusercontent.com/ajenti/ajenti/master/packaging/files/ajenti-bsd -O /etc/rc.d/ajenti
```

## 1.3 Running Ajenti

### 1.3.1 Starting service

Packages install binary *ajenti-panel* and initscript *ajenti*. You can ensure the service is running:

```
service ajenti restart
```

or:

```
/etc/init.d/ajenti restart
```

Ajenti can be run in a verbose debug mode:

```
ajenti-panel -v
```

The panel will be available on **HTTPS** port **8000** by default. The default username is **root**, and the password is **admin**

### 1.3.2 Commandline options

- **-c, --config <file>** - Use given config file instead of default
- **-v** - Debug/verbose logging
- **-d, --daemon** - Run in background (daemon mode)
- **--set-platform <id>** - Override OS detection

### 1.3.3 Debugging

Running `ajenti` with `-v` enables additional logging and Exconsole emergency console (see <https://github.com/Eugeny/exconsole>).

Exconsole can be triggered by a crash, sending SIGQUIT or pressing `Ctrl-\` on the controlling terminal.





---

## 2.1 Getting Started with Plugin Development

### 2.1.1 Prerequisites

The following are the absolutely minimal set of software required to build and run Ajenti:

- git
- coffee-script (use NPM)
- lessc (use NPM)

If you don't have CoffeeScript or LESS compiler, you won't be able to make changes to Ajenti CSS/JS files. In this case, download sources from PyPI, which includes compiled CSS/JS resources.

Debian/Ubuntu extras:

- apt-show-versions
- python-dbus (ubuntu)

### 2.1.2 Setting up

Download the source:

```
git clone git://github.com/ajenti/ajenti.git -b 1.x
```

(or download them from PyPI: <https://pypi.python.org/pypi/ajenti>)

Install the dependencies:

```
[sudo] pip install -Ur requirements.txt
```

Launch Ajenti in debug mode:

```
make run
```

Navigate to <http://localhost:8000/>.

Press Ctrl-at any time to launch an interactive Python shell and Ctrl-D to resume Ajenti.

CoffeeScript and LESS files will be recompiled automatically when you refresh the page; Python code will not. Additional debug information will be available in the console output and browser console.

Ajenti source code includes various example plugins under **Demo** category; their source is available in `ajenti/plugins/test` directory.

### 2.1.3 Creating new plugin package

New plugins can be placed in both `<source>/ajenti/plugins/` (if you expect inclusion in the source tree) and `/var/lib/ajenti/plugins`.

Each plugin package consists of few Python modules, which contain `ajenti.api.plugin` classes (*plugins*). Packages also may contain *static files*, *CoffeeScript* and *LESS code*, and XML user interface layouts:

```
* ajenti
  * plugins
    * test
      * content
        * css
          - 1.less
        * js
          - 2.coffee
      * static
        - 3.png
    * layout
      - 4.xml
  - __init__.py
  - main.py
```

### 2.1.4 Plugins

To get started, create an empty directory `<source>/ajenti/plugins/test`.

Place a file called `__init__.py` there:

```
from ajenti.api import *
from ajenti.plugins import *

info = PluginInfo(
    title='Test',
    icon=None,
    dependencies=[
        PluginDependency('main'),
    ],
)

def init():
    import main
```

In the same directory, create module `main.py`. The comments explain the concept behind plugins architecture:

```
from ajenti.api import *

@interface
class IShape (object):
    """
    This is an interface, specifying the methods required.
```

```

"""
def number_of_corners(self):
    pass

@plugin
class Square (BasePlugin, IShape):
    """
    A sample implementation, note the inheritance from both BasePlugin (optional but gives extra opt.
    """

    def init(self):
        """
        init() methods are automatically called for plugins, maintaining inheritance hierarchy
        """
        print 'Square # %s initialized' % id(self)

    def number_of_corners(self):
        return 4

@plugin
class Circle (BasePlugin, IShape):
    def number_of_corners(self):
        return 0

print 'IShape is implemented by', IShape.get_class()
foo = IShape.get() # get/create any instance of any IShape implementation
# or, more verbose, IShape.get_class().new()
print 'foo corners:', foo.number_of_corners()

# The instances are by default singleton:
print foo == IShape.get() # True

# But you can create separate ones:
foo2 = IShape.get_class().new()
print foo == foo2 # False, different instances

for another_foo in IShape.get_all(): # iterate over all possible IShape implementations
    print '\n%s says:' % another_foo, another_foo.number_of_corners()

print IShape.get_instances() # lists all three active IShape instances

```

**Output:**

```

IShape is implemented by <class 'ajenti.plugins.test.main.Square'>
Square #24838864 initialized
foo corners: 4
True
Square #24838928 initialized
False

<ajenti.plugins.test.main.Square object at 0x17b02d0> says: 4
<ajenti.plugins.test.main.Circle object at 0x17b0390> says: 0
[<ajenti.plugins.test.main.Square object at 0x17b02d0>, <ajenti.plugins.test.main.Square object at 0x

```

Learn about more interface and plugin methods here: [ajenti.api.plugin](#)

Continue to [User Interface](#)

## 2.2 User Interface

### 2.2.1 Theory

The whole Ajenti UI is a DOM tree of `ajenti.ui.UIElement` objects. After each update, the UI tree is serialized into JSON and sent to browser, where HTML DOM is assembled from it with the help of CoffeeScript code. Unlike conventional web apps, Ajenti is a stateful machine, which means you adopt a simple workflow similar to developing desktop apps, not websites.

### 2.2.2 Example

```
from ajenti.api import *
from ajenti.plugins.main.api import SectionPlugin
from ajenti.ui import on

@plugin
class TestPlugin (SectionPlugin):
    def init(self):
        self.title = 'Test' # those are not class attributes and can be only set in or after init()
        self.icon = 'question'
        self.category = 'Demo'

        """
        UI Inflater searches for the named XML layout and inflates it into
        an UIElement object tree
        """
        self.append(self.ui.inflate('test:main'))

        self.counter = 0
        self.refresh()

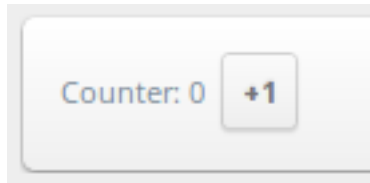
    def refresh(self):
        """
        Changing element properties automatically results
        in an UI updated being issued to client
        """
        self.find('counter-label').text = 'Counter: %i' % self.counter

    @on('increment-button', 'click')
    def on_button(self):
        """
        This method is called every time a child element
        with ID 'increment-button' fires a 'click' event
        """
        self.counter += 1
        self.refresh()
```

Add a subdirectory layout and place a file named `main.xml` there:

```
<body> <!-- an overall plugin container panel -->
  <pad> <!-- adds whitespace padding -->
    <hc> <!-- horizontal container -->
      <label id="counter-label" />
      <button id="increment-button" text="+1" style="mini" />
    </hc>
  </pad>
</body>
```

Now restart Ajenti. The new plugin **Test** will be visible under **Demo** category. Clicking the **+1** button will increase the counter.



The visible part of plugin is an `UIElement`, inherited from `ajenti.plugins.main.api.SectionPlugin`.

When you click the button, the 'click' even is fired down the UI tree. The first method to have correctly decorated `@on` method will handle the event. Alternatively, you can set event handler on the element itself by adding this code to `init`:

```
self.find('increment-button').on('click', self.on_button)
```

## 2.2.3 List of UI Elements

### 2.2.4 Containers

#### **<box>: Box**

Simplest container ever, can be scrollable

```
@p('width', default=None)
@p('height', default=None)
@p('scroll', default=False, type=bool)
```

#### **<pad>: Whitespace**

Adds a padding on four sides.

#### **<indent>: Indentation**

Adds a padding on two sides.

#### **<right>: Pull-to-right**

Pulls its content to right with `float: right`

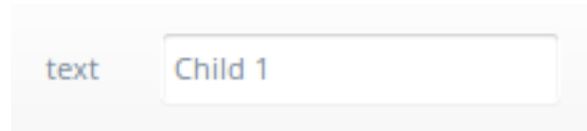
### <hc>: Horizontal Container

A horizontal stacking container

### <vc>: Vertical Container

A vertical stacking container

### <formline>: Form Line



Container for form controls, has a caption

```
@p('text', default='', bindtypes=[str, unicode])
```

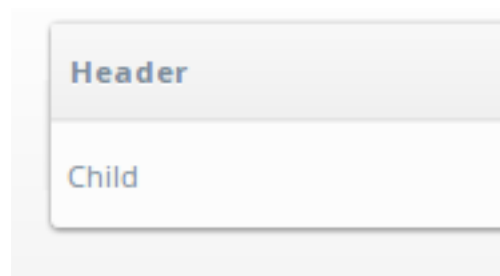
### <formgroup>: Form Group



Provides a nice form section separator

```
@p('text', default='', bindtypes=[str, unicode])
```

### <dt>, <dtr>, <dth> <dtd>: Data Table



A lined table

```
<dt>
  <dtr>
    <dth text="Header" />
  </dtr>
  <dtr>
    <dtd>
      <label text="Child" />
```

```

    </dtd>
  </dtr>
</dt>

```

### <collapserow>: Collapsible Table Row

A click-to expand table row

```

<dt>
  <collapserow>
    <label text="Header Child" />
    <label text="Body Child" />
  </collapserow>
</dt>

```

First child is a header and always visible. Second is the collapsible body.

```
@p('expanded', default=False, type=bool, bindtypes=[bool])
```

### <lt>, <ltr>, <ltd>: Layout Table

An invisible layout grid (no padding).

### <sortabletd>: Sortable Data Table



User will be able to reorder rows

```

<sortabletd>
  <dtr>
    <dtd>
      <label text="Child 1" />
    </dtd>
  </dtr>
  <dtr>
    <dtd>
      <label text="Child 2" />
    </dtd>
  </dtr>
  <dtr>
    <dtd>
      <label text="Child 3" />
    </dtd>
  </dtr>

```

```
    </td>
  </tr>
</sortabletd>

@p('sortable', default=True, type=bool)
@p('order', default='', type=str)
```

The **order** property holds the reordered element indexes ([2, 1, 3] as seen on the image)

### <tabs>, <tab>: Tabs



User will be able to reorder rows

```
<tabs>
  <tab title="1">
    <label text="Child 1" />
  </tab>
  <tab title="2">
    <label text="Child 2" />
  </tab>
  <tab title="3">
    <label text="Child 3" />
  </tab>
</tabs>

<tabs>:
@p('active', default=0)

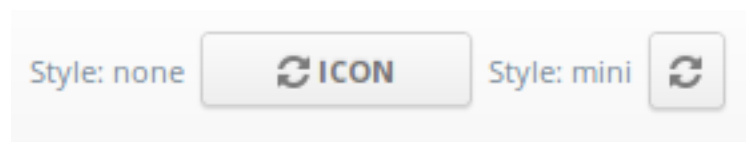
<title>:
@p('title', default='', bindtypes=[str, unicode])
```

## 2.2.5 Standard Controls

### <label>: Label

@p('text', default='', bindtypes=[str, unicode, int, float])

### <button>: Button





```
@p('text', default='', bindtypes=[str, unicode])
@p('icon', default=None)
@p('warning', default=None) # display a warning text before click

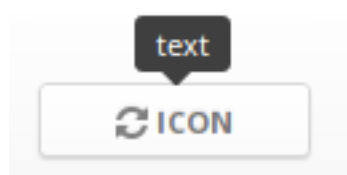
click() # fired on click
```

### <icon>: Inline Icon

Icon IDs in Ajenti are coming from this page: <http://fortawesome.github.io/Font-Awesome/icons/>

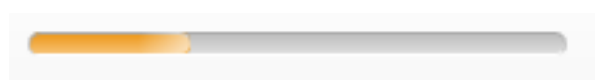
```
@p('icon', default=None, bindtypes=[str, unicode])
```

### <tooltip>: Tooltip



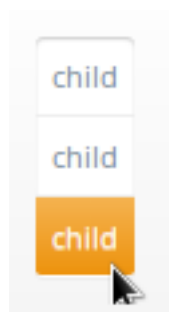
```
@p('text', default='', bindtypes=[str, unicode, int])
```

### <progressbar>: Progress Bar



```
@p('width', default=None)
@p('value', default=0, type=float, bindtypes=[float]) # between 0.0 and 1.0
```

### <list>, <listitem>: Lists



A list with clickable items:

```
<list>
  <listitem>
    <label text="child" />
  </listitem>
  <listitem>
    <label text="child" />
  </listitem>
```

```
<listitem>
  <label text="child" />
</listitem>
</list>

<listitem>:
click() # fired on click
```

## 2.2.6 Inputs

### <textbox>: Textbox

```
@p('value', default='', bindtypes=[str, unicode, int])
@p('type', default='text') # or 'integer'
```

### <editable>: Editable Label



A label that becomes textbox when clicked:

```
@p('value', default='', bindtypes=[str, unicode])
@p('icon', default=None)
```

### <checkbox>: Checkbox

```
@p('text', default='')
@p('value', default=False, bindtypes=[bool])
```

### <dropdown>: Dropdown Select

```
@p('labels', default=[], type=list)
@p('values', default=[], type=list)
@p('value', default='', bindtypes=[str, int, unicode])
```

### <combobox>: Combo Box

```
@p('labels', default=[], type=list)
@p('values', default=[], type=list)
@p('separator', default=None, type=str) # if set, combobox becomes autocomplete-multiple-input-box
@p('value', default='', bindtypes=[str, unicode])
```

## 2.3 Plugin resources

Plugin resource files are contained under `content` directory nested in the plugin directory. The `content` directory can optionally contain `css`, `js` and `static` directories holding files of respective types.

Ajenti will accept following filename extensions. injected resources will be added to <head> of web UI. cleaned resources will be deleted before build. compile resources will be pre-processed using applicable compiler.

- /content/js/\*.js - source JS (compile)
- /content/css/\*.css - source JS (compile)
- /content/js/\*.coffee - source CoffeeScript (compile)
- /content/css/\*.less - source LESS (compile)
- /content/css/\*.i.less - source LESS included somewhere else (ignored)
- /content/js/\*.m.js - pre-built JS (injected)
- /content/css/\*.m.css - pre-built CSS (injected)
- /content/js/\*.c.js - compiled JS (injected, cleaned)
- /content/css/\*.c.css - compiled CSS (injected, cleaned)

Resources under /static/ are served over HTTP at the following URL: /ajenti:static/<plugin-name>/<resource-path>, e.g.: /ajenti:static/main/favicon.png.

## 2.4 Notifications

### 2.4.1 Example

Code:

```
from ajenti.api import plugin
from ajenti.plugins.main.api import SectionPlugin
from ajenti.ui import on

@plugin
class Test (SectionPlugin):
    def init(self):
        self.title = 'Notifications'
        self.icon = 'smile'
        self.category = 'Demo'

        self.append(self.ui.inflate('test_notifications:main'))
        self.find('style').labels = self.find('style').values = ['info', 'warning', 'error']

    @on('show', 'click')
    def on_show(self):
        self.context.notify(self.find('style').value, self.find('text').value)
```

Layout:

```
<body>
  <pad>
    <vc>
      <formline text="Text">
        <textbox id="text" />
      </formline>
      <formline text="Style">
        <dropdown id="style" />
```

```
        </formline>
        <formline>
            <button icon="ok" id="show" text="Show" />
        </formline>
    </vc>
</pad>
</body>
```

Download this example

## 2.5 Bindings

Binding mechanism lets you bind your Python objects directly to UI elements and build CRUD interfaces in minutes.

Example: <https://github.com/Eugeniy/ajenti/blob/dev/ajenti/plugins/test/binder/main.py>

### 2.5.1 Simple bindings

Code:

```
from ajenti.api import plugin
from ajenti.plugins.main.api import SectionPlugin
from ajenti.ui import on
from ajenti.ui.binder import Binder

class Settings (object): # use new-style object at all times!
    def __init__(self):
        self.label_text = ''
        self.label_bold = False
        self.label_style = ''

@plugin
class Test (SectionPlugin):
    def init(self):
        self.title = 'Bindings'
        self.icon = 'smile'
        self.category = 'Demo'

        self.append(self.ui.inflate('test_bindings:main'))

        self.settings = Settings()

        # Bind the settings object to the section UI element (self)
        self.binder = Binder(self.settings, self)
        self.binder.populate()

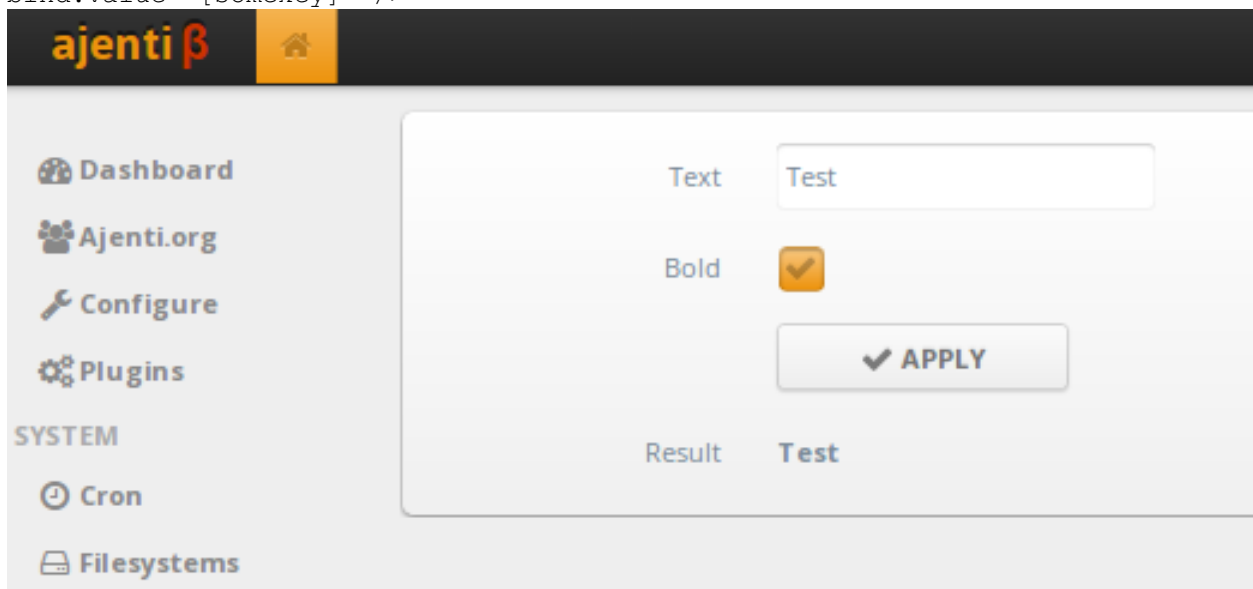
    @on('apply', 'click')
    def on_apply(self):
        self.binder.update() # update objects from UI
        self.settings.label_style = 'bold' if self.settings.label_bold else ''
        self.binder.populate() # update UI with objects
```

Here, the `Settings` object acts as a data model. `ajenti.ui.binder.Binder` object connects data with UI. `autodiscover` method scans the UI for bindable elements, `populate` method updates UI with the data from bound objects, and `update` method applies UI changes to objects.

Layout:

```
<body>
  <pad>
    <vc>
      <formline text="Text">
        <textbox bind="label_text" />
      </formline>
      <formline text="Bold">
        <checkbox bind="label_bold" />
      </formline>
      <formline>
        <button icon="ok" id="apply" text="Apply" />
      </formline>
      <formline text="Result">
        <label bind:text="label_text" bind:style="label_style" />
      </formline>
    </vc>
  </pad>
</body>
```

We have added `bind` attributes to the elements which are to be auto-populated with values. If you want to bind multiple properties, use XML attributes like `bind:text` or `bind:style`. Dictionary values and `__getattr__` powered indexes can be bound by enclosing the key name in square brackets, e.g.: `<label bind:value="[somekey]" />`



If you would like to continue binding on a nested object, use `binder:context` attribute:

```
<body>
  <vc>
    <label bind:value="simple_str_field" />    <!-- data.simple_str_field -->
    <box binder:context="object_field">
      <label bind:value="objects_str_field" />    <!-- data.object_field.objects_str_field -->
    </box>
    <box binder:context="dict_field">
```

```
        <label bind:value="[dict_key]" /> <!-- data.dict_field['dict_key'] -->
    </box>
</vc>
</body>
```

## 2.5.2 Collection Bindings

Ajenti supports following collection bindings:

- Binding iterable to list of elements (*ajenti.ui.binder.ListAutoBinding*)
- Binding dict to key-annotated elements (*ajenti.ui.binder.DictAutoBinding*)
- Binding iterable with a child template (*ajenti.ui.binder.CollectionAutoBinding*)

Code:

```
import json

from ajenti.api import plugin
from ajenti.plugins.main.api import SectionPlugin
from ajenti.ui import on
from ajenti.ui.binder import Binder

class Person (object):
    def __init__(self, name, **kwargs):
        self.name = name
        self.params = kwargs

    def __repr__(self):
        return json.dumps({'name': self.name, 'params': self.params})

@plugin
class Test (SectionPlugin):
    def init(self):
        self.title = 'Collection Bindings'
        self.icon = 'smile'
        self.category = 'Demo'

        self.append(self.ui.inflate('test_bindings_collections:main'))

        andy = Person('andy', phone='123')
        bob = Person('bob', phone='321')

        self.obj_list = (andy, bob)
        self.obj_collection = [andy, bob]

        # This callback is used to autogenerate a new item with 'Add' button
        self.find('collection').new_item = lambda c: Person('new person', phone='000')

        self.binder = Binder(self, self)
        self.refresh()

    def refresh(self):
        self.binder.update()
        self.raw_data = repr(self.obj_collection)
```

```

self.binder.populate()

@on('apply', 'click')
def on_apply(self):
    self.refresh()

```

## Layout:

```

<body>
  <pad>
    <vc>
      <formline text="bind:list">
        <bind:list bind="obj_list">
          <box>
            <label bind="name" />
          </box>
          <box>
            <label bind="name" />
          </box>
        </bind:list>
      </formline>

      <formline text="bind:collection">
        <bind:collection bind="obj_collection" id="collection">
          <vc>
            <dt bind="__items">
              <dtr>
                <dth text="Name" />
                <dth text="Phone" />
                <dth />
              </dtr>
            </dt>
            <button icon="plus" style="mini" bind="__add" />
          </vc>

          <bind:template>
            <dtr>
              <dtd> <textbox bind="name" /> </dtd>

              <dtd>
                <bind:dict bind="params">
                  <textbox bind="phone" />
                </bind:dict>
              </dtd>

              <dtd> <button icon="remove" style="mini" bind="__delete" /> </dtd>
            </dtr>
          </bind:template>

        </bind:collection>
      </formline>

      <formline text="Raw data">
        <label bind="raw_data" />
      </formline>

      <formline>
        <button icon="ok" id="apply" text="Apply" />
      </formline>

```

```

    </vc>
  </pad>
</body>

```

Note the special bind attribute values used in `bind:collection`:

- `__items` denotes the container for items
- `__add` denotes a button which will generate a new item (optional)
- `__remove` denotes a button which will remove an item (optional)

## 2.6 Custom UI Controls

You can create any type of a reusable UI control. Remember to take a look at default controls in `ajenti/plugins/main` for guidance.

### 2.6.1 Example

In this example, we'll create a HTML5 slider control.

Code:

```

from ajenti.api import plugin
from ajenti.plugins.main.api import SectionPlugin
from ajenti.ui import on, p, UIElement

@plugin
class Test (SectionPlugin):
    def init(self):
        self.title = 'Controls'
        self.icon = 'smile'
        self.category = 'Demo'
        self.append(self.ui.inflate('test_controls:main'))

    @on('check', 'click')

```



```

def on_show(self):
    self.context.notify('info', 'Value is %i' % self.find('slider').value)

@p('value', type=int, default=0)
@plugin
class Slider (UIElement):
    typeid = 'slider'

```

Layout:

```

<body>
  <pad>
    <vc>
      <formline text="Slider">
        <slider id="slider" value="0" />
      </formline>
      <formline>
        <button icon="ok" id="check" text="Get value" />
      </formline>
    </vc>
  </pad>
</body>

```

Control class is decorated with `ajenti.ui.p()` for each of its properties. The main client-side logic is implemented through CoffeeScript code (though you can try to get away with pure-JS).

CoffeeScript:

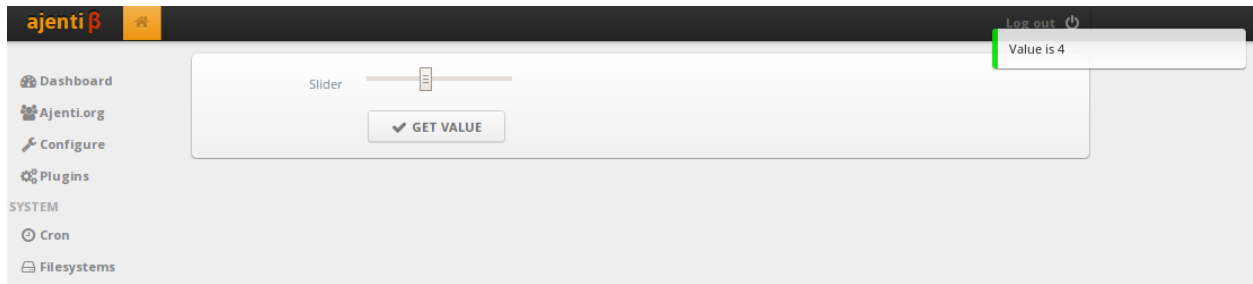
```

class window.Controls.slider extends window.Control
  createDom: () ->
    # createDom() must return HTML
    """
      <div>
        <input type="range" min="0" max="10" />
      </div>
    """

  setupDom: (dom) ->
    # setupDom may attach event handler and perform other DOM manipulations
    # use this.properties hash to populate control with its current state
    super(dom)
    @input = $(@dom).find('input')
    @input.val(@properties.value)

  detectUpdates: () ->
    # detectUpdates() should return a hash containing only changed properties
    # be sure to not report unchanged properties since this will lead to infinite update loops
    r = {}
    value = parseInt(@input.val())
    if value != @properties.value
      r.value = value
    return r

```



Download this example

## 2.7 Handling HTTP Requests

### 2.7.1 Example

This example illustrates various HTTP responses. Try following URLs:

- <http://localhost:8000/ajenti:demo/notify?text=hello>
- <http://localhost:8000/ajenti:demo/respond/redirect>
- [http://localhost:8000/ajenti:demo/respond/server\\_error](http://localhost:8000/ajenti:demo/respond/server_error)
- <http://localhost:8000/ajenti:demo/respond/ok>
- <http://localhost:8000/ajenti:demo/respond/file>

Code:

```
from ajenti.api import plugin, BasePlugin
from ajenti.api.http import HttpPlugin, url

@plugin
class HttpDemo (BasePlugin, HttpPlugin):
    @url('/ajenti:demo/notify')
    def get_page(self, context):
        if context.session.identity is None:
            context.respond_redirect('/')
        self.context.notify('info', context.query.getvalue('text', ''))
        context.respond_ok()
        return ''

    @url('/ajenti:demo/respond/(?P<what>.+)' )
    def get_response(self, context, what=None):
        if what == 'ok':
            context.respond_ok()
            return 'Hello!'
        if what == 'redirect':
            return context.respond_redirect('/')
        if what == 'server_error':
            return context.respond_server_error()
        if what == 'forbidden':
            return context.respond_forbidden()
        if what == 'not_found':
            return context.respond_not_found()
```

```
if what == 'file':
    return context.file('/etc/issue')
```

Download this example

## 2.8 Dashboard Widgets

Dashboard plugin API exposes two widget types: *ajenti.plugins.dashboard.api.DashboardWidget* and *ajenti.plugins.dashboard.api.ConfigurableWidget*.

### 2.8.1 Simple widget

Example:

```
from ajenti.api import plugin
from ajenti.plugins.dashboard.api import DashboardWidget

@plugin
class HelloWidget (DashboardWidget):
    name = 'Hello Widget'
    icon = 'comment'

    def init(self):
        self.append(self.ui.inflate('hello_widget:widget'))
        self.find('text').text = 'Hello'
```

Layout:

```
<hc>
  <box width="20">
    <icon id="icon" icon="comment" />
  </box>

  <box width="90">
    <label id="name" style="bold" text="Widget" />
  </box>

  <label id="text" />
</hc>
```

### 2.8.2 Configurable widget

Configurable widgets have a dict config, configuration dialog and a configuration button.

Example (real dashboard Text widget):

```
from ajenti.api import plugin
from ajenti.plugins.dashboard.api import ConfigurableWidget

@plugin
class TextWidget (ConfigurableWidget):
    name = _('Text')
    icon = 'font'
```

```
def on_prepare(self):
    # probably not configured yet!
    self.append(self.ui.inflate('dashboard:text'))

def on_start(self):
    # configured by now
    self.find('text').text = self.config['text']

def create_config(self):
    return {'text': ''}

def on_config_start(self):
    # configuration begins now, a chance to fill the configuration dialog
    pass

def on_config_save(self):
    self.config['text'] = self.dialog.find('text').value
```

Layout:

```
<hc>
  <label id="text" text="---" />

  <dialog id="config-dialog" visible="False">
    <pad>
      <formline text="{Text}">
        <textbox id="text" />
      </formline>
    </pad>
  </dialog>
</hc>
```

---

## API Reference

---

### 3.1 ajenti

`ajenti.config = None`  
Loaded config, is a *reconfigure.items.ajenti.AjentiData*

`ajenti.platform = None`  
Current platform

`ajenti.platform_string = None`  
Human-friendly platform name

`ajenti.platform_unmapped = None`  
Current platform without “Ubuntu is Debian”-like mapping

`ajenti.installation_uid = None`  
Unique installation ID

`ajenti.version = None`  
Ajenti version

`ajenti.server = None`  
Web server

`ajenti.debug = False`  
Debug mode

`ajenti.init ()`

`ajenti.exit ()`

`ajenti.restart ()`

### 3.2 ajenti.api

**class** `ajenti.api.PluginInfo (**kwargs)`  
Describes a loaded plugin package

**class** `ajenti.api.BasePlugin`  
A base plugin class that provides *AppContext* and *classconfig* functionality.

**classconfig\_editor = None**  
Override this in your class with an *ajenti.plugins.configurator.api.ClassConfigEditor* derivative

**classconfig\_name = None**

Override this in your class if you want this plugin to be configurable through Configure > Plugins

**classconfig\_root = False**

When True, classconfig will be stored in root's config section disregarding current user

**context = None**

Automatically receives a reference to the current *AppContext*

**create\_classconfig ()**

**default\_classconfig = None**

Override this in your class with a default config object (must be JSON-serializable)

**init ()**

Do your initialization here. Correct bottom-to-up inheritance call order guaranteed.

**load\_classconfig ()**

Loads the content of classconfig attribute from the user's configuration section.

**open\_content (path, mode='r')**

Provides access to plugin-specific files from /content directory of the package

**Parameters**

- **path (str)** – path relative to package's /content
- **mode (str)** – Python file access mode

**Returns** An open file object

**Return type** file

**save\_classconfig ()**

Saves the content of classconfig attribute into the user's configuration section.

**class** ajenti.api.**AppContext** (parent, httpcontext)

A session-specific context provided to everyone who inherits *BasePlugin*.

**session**

current HTTP session: ajenti.middleware.Session

**user**

current logged in user: reconfigure.items.ajenti.UserData

Methods injected by MainPlugin:

**notify (text)**

**Parameters text** – Notification text to show

**launch (id, \*args, \*\*kwargs)**

**Parameters id** – Intent ID to be launched

ajenti.api.**plugin (cls)**

A decorator to create plugin classes:

```
@plugin
class SomePlugin (ISomething):
    pass
```

If the class has a `verify` method returning `bool`, it's invoked. If the method returned `False`, plugin is rejected and removed from implementation lists.

If the class has a `platforms` attribute, which is a list of supported platform names, it's compared against the current runtime OS platform. If the current platform is not in the list, plugin is also rejected.

Following class methods are injected.

**.get** (*context=<current context>*)

**Returns** any existing instance or creates a new one

**.new** (*\*args, context=<current context>, \*\*kwargs*)

**Returns** a new instance. Use this method instead of constructor, since it invokes the proper initialization chain and registers the instance

**Return type** class, None

`ajenti.api.rootcontext` (*cls*)

Enforces use of root PluginContext by default for `.get()` and `.new()` classmethods.

`ajenti.api.notrack` (*cls*)

Disables instance tracking of plugin (and derivative) instances within PluginContext via `get/get_all` and similar methods.

**Return type** class

`ajenti.api.notrack_this` (*cls*)

Disables instance tracking of plugin instances within PluginContext via `get/get_all` and similar methods.

**Return type** class

`ajenti.api.track` (*cls*)

Enables previously disabled instance tracking of plugin.

**Return type** class

`ajenti.api.persistent` (*cls*)

Makes this plugin non-GCable

**Return type** class

`ajenti.api.extract_context` ()

An utility function that extracts and returns the nearest `AppContext` from the current call stack.

**Return type** `ajenti.plugins.PluginContext`, None

**exception** `ajenti.api.NoImplementationsError`

`ajenti.api.interface` (*cls*)

A decorator to create plugin interfaces:

```
@interface
class ISomething (object):
    def contract(self):
        pass
```

Following class methods are injected:

**.get** (*context=<current context>*)

**Returns** any existing instance or creates a new one

**.get\_all** (*context=<current context>*)

**Returns** list of instances for each implementation

**.get\_class** ()

**Returns** any implementation class

`.get_classes()`

**Returns** list of implementation classes

`.get_instances(context=<current context>)`

**Returns** list of all existing instances

**Return type** class

### 3.3 ajenti.api.http

### 3.4 ajenti.api.sensors

**class** `ajenti.api.sensors.Sensor`

Base class for a Sensor. Sensors measure system status parameters and can be queried from other plugins.

**static find** (*id*)

Returns a Sensor by name

**Parameters** *id* (*str*) – sensor ID

**Return type** *Sensor*, None

**get\_variants** ()

Override this and return a list of available variants.

**Return type** list

**id** = None

**init** ()

**measure** (*variant=None*)

Override this and perform the measurement.

**Parameters** *variant* (*str*, *None*) – variant to measure

**Return type** int, float, tuple, list, dict, str

**timeout** = 0

**value** (*variant=None*)

Returns sensor's measurement for a specific *variant*. Sensors can have multiple variants; for example, disk usage sensor accepts device name as a variant.

**Parameters** *variant* (*str*, *None*) – variant to measure

**Return type** int, float, tuple, list, dict, str

### 3.5 ajenti.http

### 3.6 ajenti.ipc

**class** `ajenti.ipc.IPCHandler`

Interface for custom IPC endpoints



```

get_name ()
    Should return short identifier of IPC endpoint:
    $ ajenti-ipc <endpoint-name> <args>

```

**Rtype** *str*

```

handle (args)
    Override to handle IPC requests

```

**Parameters** *args* (*list*) – list of *str* parameters

```

class ajenti.ipc.IPCServer

```

```

    run ()

```

```

    start ()

```

## 3.7 ajenti.middleware

## 3.8 ajenti.plugins

```

exception ajenti.plugins.PluginLoadError

```

```

exception ajenti.plugins.PluginFormatError

```

```

    describe ()

```

```

exception ajenti.plugins.PluginCrashed (e)

```

```

    describe ()

```

```

class ajenti.plugins.Dependency

```

```

    exception Unsatisfied

```

```

        describe ()

```

```

        reason ()

```

```

        Dependency.build_exception ()

```

```

        Dependency.check ()

```

```

        Dependency.satisfied ()

```

```

        Dependency.value

```

```

class ajenti.plugins.ModuleDependency (module_name)

```

```

    exception Unsatisfied

```

```

        reason ()

```

```

        ModuleDependency.description = 'Python module'

```

```
ModuleDependency.is_satisfied()
class ajenti.plugins.PluginDependency(plugin_name)

    exception Unsatisfied

        reason()

PluginDependency.description = 'Plugin'
PluginDependency.is_satisfied()
class ajenti.plugins.BinaryDependency(binary_name)

    exception Unsatisfied

        reason()

BinaryDependency.description = 'Application binary'
BinaryDependency.is_satisfied()
class ajenti.plugins.FileDependency(file_name)

    exception Unsatisfied

        reason()

FileDependency.description = 'File'
FileDependency.is_satisfied()
class ajenti.plugins.PluginContext
    Container of interfaces and plugins

    get_instance(cls)
    get_instances(cls)
    instantiate(cls, *args, **kwargs)
    vacuum_instances()
class ajenti.plugins.PluginManager
    Handles plugin loading and unloading

    blacklist = []
    context = <ajenti.plugins.PluginContext object>
    extra_location = '/var/lib/ajenti/plugins'
    get_all()
    get_implementations(iface)
    get_order()
    get_plugins_root()
    load(name)
        Loads given plugin
```

```

load_all()
load_recursive(name)
register_implementation(impl)
register_interface(iface)
resolve_path(name)

```

### 3.9 ajenti.profiler

```

ajenti.profiler.get_profiles()
    Returns all accumulated profiling values

ajenti.profiler.profile_end(name=None)
    Ends a profiling interval with specific name

ajenti.profiler.profile_start(name)
    Starts a profiling interval with specific name Profiling data is sent to the client with next data batch.

ajenti.profiler.profiled(namefx=None)

```

### 3.10 ajenti.ui

```

class ajenti.ui.UI
    The root UI object, one per session

    clear_updates()
        Marks all pending updates as processed

    create(typeid, *args, **kwargs)
        Creates an element by its type ID.

        Parameters typeid (str) – type ID

    dispatch_event(uid, event, params=None)
        Dispatches an event to an element with given UID

        Parameters

        • uid (int) – element UID
        • event (str) – event name
        • params (dict, None) – event arguments

    find(id)

        Parameters id (str) – element ID

        Returns nearest element with given ID

        Return type UIElement, None

    find_uid(uid)

        Parameters uid (int) – element UID

        Returns nearest element with given unique ID

        Return type UIElement, None

```

**has\_updates** ()  
Checks for pending UI updates

**Return type** bool

**inflate** (*layout*)

**Parameters** **layout** (*str*) – layout spec: “<plugin id>:<layout file name without extension>”

**Returns** an inflated element tree of the given layout XML name

**Return type** *UIElement*

**init** ()

**render** ()  
Renders the UI into JSON

**Return type** dict

**class** `ajenti.ui.UIElement` (*ui*, *typeid=None*, *children=[]*, *\*\*kwargs*)  
Base UI element class

**append** (*child*)  
Appends a child

**Parameters** **child** (*UIElement*) – child

**bind**  
Bound property name

**bindtransform**  
Value transformation function for one-direction bindings

**broadcast** (*method*, *\*args*, *\*\*kwargs*)  
Calls *method* on every member of the subtree

**Parameters** **method** (*str*) – method

**clear\_updates** ()  
Marks all pending updates as processed

**client**  
Whether this element’s events are only processed on client side

**clone** (*set\_ui=None*, *set\_context=None*)  
**Returns** a deep copy of the element and its children. Property values are shallow copies.  
**Return type** *UIElement*

**contains** (*element*)  
Checks if the *element* is in the subtree of *self*

**Parameters** **element** (*UIElement*) – element

**delete** ()  
Detaches this element from its parent

**dispatch\_event** (*uid*, *event*, *params=None*)  
Dispatches an event to an element with given UID

**Parameters**

- **uid** (*int*) – element UID
- **event** (*str*) – event name

- **params** (*dict, None*) – event arguments

**empty** ()

Detaches all child elements

**event** (*event, params=None*)

Invokes handler for *event* on this element with given *\*\*params*

**Parameters**

- **event** (*str*) – event name
- **params** (*dict, None*) – event arguments

**find** (*id*)

**Parameters** *id* (*str*) – element ID

**Returns** the nearest child with given ID or None

**Return type** *UIElement*, None

**find\_type** (*typeid*)

**Returns** the nearest child with given type ID or None

**Return type** *UIElement*, None

**find\_uid** (*uid*)

**Parameters** *uid* (*int*) – element UID

**Returns** the nearest child with given UID or None

**Return type** *UIElement*, None

**has\_updates** ()

Checks for pending UI updates

**id**

Element ID

**init** ()

**invalidate** ()

**nearest** (*predicate, exclude=None, descend=True*)

Returns the nearest child which matches an arbitrary predicate lambda

**Parameters**

- **predicate** (*function*) – lambda element: bool
- **exclude** (*function, None*) – lambda element: bool - excludes matching branches from search
- **descend** (*bool*) – whether to descend inside matching elements

**on** (*event, handler, \*args*)

Binds event with ID *event* to handler. *\*args* will be passed to the handler. :param event: event :type event: str :param handler: handler :type handler: function

**path\_to** (*element*)

**Returns** a list of elements forming a path from self to element

**Return type** list

**post\_clone** ()

**property\_definitions****remove** (*child*)

Detaches the child

**Parameters** **child** (*UIElement*) – child**render** ()

Renders this element and its subtree to JSON

**Return type** dict**reverse\_event** (*event, params=None*)

Raises the event on this element by feeding it to the UI root (so that @on methods in ancestors will work).

**Parameters**

- **event** (*str*) – event name
- **params** (*dict*) – event arguments

**style**

Additional CSS class

**typeid = None****visible**

Visibility of the element

`ajenti.ui.p` (*prop, default=None, bindtypes=[], type=<type 'unicode'>, public=True, doc=None*)Creates an UI property inside an *UIElement*:

```
@p('title')
@p('category', default='Other', doc='Section category name')
@p('active', default=False)
class SectionPlugin (BasePlugin, UIElement):
    typeid = 'main:section'
```

**Parameters**

- **default** (*object*) – Default value
- **bindtypes** (*list*) – List of Python types that can be bound to this property
- **type** (*object*) – expected Python type for this value
- **public** (*bool*) – whether this property is rendered and sent to client
- **doc** (*str, None*) – docstring

**Return type** function`ajenti.ui.on` (*id, event*)

Sets the decorated method to handle indicated event:

```
@plugin
class Hosts (SectionPlugin):
    def init(self):
        self.append(self.ui.inflate('hosts:main'))
        ...

@on('save', 'click')
def save(self):
    self.config.save()
```

**Parameters**

- **id** (*str*) – element ID
- **event** (*str*) – event name

**Return type** function

## 3.11 ajenti.ui.binder

**class** `ajenti.ui.binder.Binding` (*object, attribute, ui*)

A base class for bindings. Binding is a link between a Python object attribute and Ajenti UI element's property.

**Parameters**

- **object** – a Python object
- **attribute** – attribute name
- **ui** – Ajenti `ajenti.ui.UIElement`

**classmethod** `applicable` (*object, attribute*)

**classmethod** `extract` (*object, attribute, ignore\_errors=True*)

`get` ()

**Returns** value of the bound attribute

`populate` ()

Should update the UI with attribute's value

`set` (*value*)

Sets value of the bound attribute

`unpopulate` ()

Should revert UI to normal state

`update` ()

Should update the attribute with data from the UI

**class** `ajenti.ui.binder.PropertyBinding` (*obj, attribute, ui, property=None*)

A simple binding between UI element's property and Python object's attribute

**Parameters** **property** – UI property name. If `None`, property is deduced from `bindtypes`

`populate` ()

`update` ()

**class** `ajenti.ui.binder.ListAutoBinding` (*object, attribute, ui*)

Binds values of a collection to UI element's children consecutively, using `Binder`

`populate` ()

`unpopulate` ()

`update` ()

**class** `ajenti.ui.binder.DictAutoBinding` (*object, attribute, ui*)

Binds values from a dict to UI element's children mapping 'bind' attribute to dict key, using `Binder`

`populate` ()

`unpopulate` ()

**update** ()

**class** `ajenti.ui.binder.CollectionAutoBinding` (*object, attribute, ui*)

Binds values of a collection to UI element's children using a template. The expected UI layout:

```
<xml xmlns:bind="bind">
  <bind:collection id="<binding to this>">
    <container-element bind="__items">
      <!-- instantiated templates will appear here -->
    </container-element>

    <bind:template>
      <!-- a template for one collection item
           it will be bound to item using ajenti.ui.binder.Binder -->
      <label bind="some_property" />

      <button id="__delete" /> <!-- a delete button may appear in the template -->
    </bind:template>

    <button id="__add" /> <!-- an add button may appear inside collection tag -->
  </bind:collection>
</xml>
```

**get\_template** (*item, ui*)

**on\_add** ()

**on\_delete** (*item*)

**populate** ()

**set\_page** (*page=0*)

**unpopulate** ()

**update** ()

**class** `ajenti.ui.binder.Binder` (*object=None, ui=None*)

An automatic object-to-ui-hierarchy binder. Uses `bind` UI property to find what and where to bind. If `object` is not `None`, the `Binder` is also initialized (see `setup(object)`) with this data object.

#### Parameters

- **object** – Python object
- **ui** – UI hierarchy root

**add** (*binding*)

**autodiscover** (*object=None, ui=None*)

**populate** ()

Populates the bindings.

**reset** (*object=None, ui=None*)

Cancels the binding and replaces Python object / UI root.

**setup** (*object=None*)

Initializes the `Binder` with a data object. :type object: object

**unpopulate** ()

Unpopulates the bindings.



**update ()**  
Updates the bindings.

**class** ajenti.ui.binder.**BasicCollectionElement** (*ui*, *typeid=None*, *children=[]*, *\*\*kwargs*)

**binding**  
Collection binding class to use

**filter**  
Called to filter collections values, lambda value: bool

**post\_bind**  
Called after binding is complete, lambda object, collection, ui: None

**post\_item\_bind**  
Called after an item is bound, lambda object, collection, item, item-ui: None

**post\_item\_update**  
Called after an item is updated, lambda object, collection, item, item-ui: None

**values**  
Called to extract values from the collection, lambda collection: []

**class** ajenti.ui.binder.**ListElement** (*ui*, *typeid=None*, *children=[]*, *\*\*kwargs*)

**typeid = 'bind:list'**

**class** ajenti.ui.binder.**CollectionElement** (*ui*, *typeid=None*, *children=[]*, *\*\*kwargs*)

**add\_item**  
Called to append value to the collection, lambda item, collection: None

**delete\_item**  
Called to remove value from the collection, lambda item, collection: None

**new\_item**  
Called to create an empty new item, lambda collection: object ()

**pagesize**

**sorting**  
If defined, used as key function to sort items

**typeid = 'bind:collection'**

## 3.12 ajenti.users

ajenti.users.**restrict** (*permission*)  
Marks a decorated function as requiring permission. If the invoking user doesn't have one, *SecurityError* is raised.

**class** ajenti.users.**PermissionProvider**  
Override to create your own set of permissions

**get\_name ()**  
Should return a human-friendly name for this set of permissions (displayed in Configurator) :rtype: str

**get\_permissions ()**  
Should return a list of permission names

**Return type** list

**exception** `ajenti.users.SecurityError` (*permission*)  
Indicates that user didn't have a required permission.

**permission**  
permission ID

**class** `ajenti.users.UserManager`

**check\_password** (*username, password, env=None*)  
Verifies the given username/password combo

**Return type** bool

**classconfig\_root** = True

**default\_classconfig** = {'sync-provider': ''}

**get\_sync\_provider** (*fallback=False*)

**Return type** `ajenti.usersync.UserSyncProvider`

**has\_permission** (*context, permission*)  
Checks whether the current user has a permission

**Return type** bool

**hash\_password** (*password*)

**Return type** str

**hash\_passwords** ()

**require\_permission** (*context, permission*)  
Checks current user for given permission and raises `SecurityError` if he doesn't have one :type permission: str :raises: SecurityError

**set\_password** (*username, password*)

**set\_sync\_provider** (*provider\_id*)

### 3.13 ajenti.util

`ajenti.util.public` (*f*)  
" Use a decorator to avoid retyping function/class names.

Based on an idea by Duncan Booth: <http://groups.google.com/group/comp.lang.python/msg/11cbb03e09611b8a>

Improved via a suggestion by Dave Angel: <http://groups.google.com/group/comp.lang.python/msg/3d400fb22d8a42e1>

`ajenti.util.str_fsize` (*sz*)  
Formats file size as string (i.e., 1.2 Mb)

`ajenti.util.str_timedelta` (*s*)  
Formats a time delta (i.e., "5 days, 5:06:07")

`ajenti.util.cache_value` (*duration=None*)  
Makes a function lazy.

**Parameters** `duration` (*int*) – cache duration in seconds (default: infinite)

`ajenti.util.platform_select (**values)`

Selects a value from **kwargs** depending on runtime platform

```
service = platform_select(  
    debian='samba',  
    ubuntu='smbd',  
    centos='smbd',  
    default='samba',  
)
```

`ajenti.util.make_report (e)`

Formats a bug report.



---

## Plugin API Reference

---

### 4.1 ajenti.plugins.main.api

### 4.2 ajenti.plugins.dashboard.api

**class** `ajenti.plugins.dashboard.api.ConfigurableWidget` (*ui*, *typeid=None*, *children=[]*, *\*\*kwargs*)

Base class for widgets with a configuration dialog

**begin\_configuration** ()

**create\_config** ()

Should return a default config dict

**init** ()

**on\_config** (*button*)

**on\_config\_save** ()

Called when user is done configuring the widget.

**on\_config\_start** ()

Called when user begins to configure the widget. Should populate the config dialog.

**on\_prepare** ()

Widget should create its UI in this method. Called before **self.config** is created

**on\_start** ()

Widget should populate its UI in this method. **self.config** is now available.

**class** `ajenti.plugins.dashboard.api.DashboardWidget` (*ui*, *typeid=None*, *children=[]*, *\*\*kwargs*)

Base class for widgets (inherits `ajenti.ui.UIElement`).

**config**

current configuration dict of this widget instance

**configurable**

**container**

**hidden = False**

If True, user will not be able to add this widget through dashboard

**icon = None**

Widget icon name

```
index
name = '—'
    Widget type name
save_config()
typeid = 'dashboard:widget'
```

### 4.3 ajenti.plugins.configurator.api

```
class ajenti.plugins.configurator.api.ClassConfigEditor(ui, typeid=None, children=[],
**kwargs)
```

```
title
typeid = 'configurator:classconfig-editor'
```

### 4.4 ajenti.plugins.db\_common.api

### 4.5 ajenti.plugins.webserver\_common.api

### 4.6 ajenti.plugins.packages.api

```
class ajenti.plugins.packages.api.PackageInfo
class ajenti.plugins.packages.api.PackageManager
```

```
do(actions, callback=<function <lambda>>)
get_lists()
init()
refresh()
search(query)
```

### 4.7 ajenti.plugins.services.api

```
class ajenti.plugins.services.api.Service

command(cmd)
icon
restart()
source = 'unknown'
    Marks which ServiceManager owns this object
start()
```

```

    stop ()

class ajenti.plugins.services.api.ServiceManager

    get_one (name)
        Returns a Service by name.

class ajenti.plugins.services.api.ServiceMultiplexor
    Merges together output of all available ServiceManagers.

    get_all (*args, **kwargs)
        Returns all Services.

    get_one (name)
        Returns a Service by name.

    init ()

```

## 4.8 ajenti.plugins.tasks.api

```

class ajenti.plugins.tasks.api.JobDefinition (j={})

    save ()

class ajenti.plugins.tasks.api.Task (**kwargs)
    Base class for custom tasks

    Parameters

    • name – display name

    • ui – full layout name for parameter editor, will be bound to parameter dictionary (so begin
      it with <bind:dict bind="params">)

    • hidden – if True, task won't be available for manual creation

    abort ()

    get_progress ()

    hidden = False

    init ()

    name = '—'

    run (**kwargs)
        Override with your task actions here. Raise TaskError in case of emergency. Check aborted often and
        return if it's True

    set_progress (current, max)

    start ()

    ui = None

class ajenti.plugins.tasks.api.TaskDefinition (j={}, task_class=None)

    get_class ()

    save ()

```

**exception** ajenti.plugins.tasks.api.**TaskError**

**class** ajenti.plugins.tasks.api.**TaskResult**

**ABORTED = 1**

**CRASH = 3**

**ERROR = 2**

**SUCCESS = 0**



---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



## a

- ajenti, 25
- ajenti.api, 25
- ajenti.api.sensors, 28
- ajenti.ipc, 28
- ajenti.plugins, 29
- ajenti.plugins.configurator.api, 42
- ajenti.plugins.dashboard.api, 41
- ajenti.plugins.packages.api, 42
- ajenti.plugins.services.api, 42
- ajenti.plugins.tasks.api, 43
- ajenti.profiler, 31
- ajenti.ui, 31
- ajenti.ui.binder, 35
- ajenti.users, 37
- ajenti.util, 38



## Symbols

.get() (in module ajenti.api), 27  
 .get\_all() (in module ajenti.api), 27  
 .get\_class() (in module ajenti.api), 27  
 .get\_classes() (in module ajenti.api), 28  
 .get\_instances() (in module ajenti.api), 28  
 .new() (in module ajenti.api), 27

## A

abort() (ajenti.plugins.tasks.api.Task method), 43  
 ABORTED (ajenti.plugins.tasks.api.TaskResult attribute), 44  
 add() (ajenti.ui.binder.Binder method), 36  
 add\_item (ajenti.ui.binder.CollectionElement attribute), 37  
 ajenti (module), 25  
 ajenti.api (module), 25  
 ajenti.api.sensors (module), 28  
 ajenti.ipc (module), 28  
 ajenti.plugins (module), 29  
 ajenti.plugins.configurator.api (module), 42  
 ajenti.plugins.dashboard.api (module), 41  
 ajenti.plugins.packages.api (module), 42  
 ajenti.plugins.services.api (module), 42  
 ajenti.plugins.tasks.api (module), 43  
 ajenti.profiler (module), 31  
 ajenti.ui (module), 31  
 ajenti.ui.binder (module), 35  
 ajenti.users (module), 37  
 ajenti.util (module), 38  
 ApplicationContext (class in ajenti.api), 26  
 append() (ajenti.ui.UIElement method), 32  
 applicable() (ajenti.ui.binder.Binding class method), 35  
 autodiscover() (ajenti.ui.binder.Binder method), 36

## B

BasePlugin (class in ajenti.api), 25  
 BasicCollectionElement (class in ajenti.ui.binder), 37  
 begin\_configuration() (ajenti.plugins.dashboard.api.ConfigurableWidget method), 41

BinaryDependency (class in ajenti.plugins), 30  
 BinaryDependency.Unsatisfied, 30  
 bind (ajenti.ui.UIElement attribute), 32  
 Binder (class in ajenti.ui.binder), 36  
 binding (ajenti.ui.binder.BasicCollectionElement attribute), 37  
 Binding (class in ajenti.ui.binder), 35  
 bindtransform (ajenti.ui.UIElement attribute), 32  
 blacklist (ajenti.plugins.PluginManager attribute), 30  
 broadcast() (ajenti.ui.UIElement method), 32  
 build\_exception() (ajenti.plugins.Dependency method), 29

## C

cache\_value() (in module ajenti.util), 38  
 check() (ajenti.plugins.Dependency method), 29  
 check\_password() (ajenti.users.UserManager method), 38  
 classconfig\_editor (ajenti.api.BasePlugin attribute), 25  
 classconfig\_name (ajenti.api.BasePlugin attribute), 25  
 classconfig\_root (ajenti.api.BasePlugin attribute), 26  
 classconfig\_root (ajenti.users.UserManager attribute), 38  
 ClassConfigEditor (class in ajenti.plugins.configurator.api), 42  
 clear\_updates() (ajenti.ui.UI method), 31  
 clear\_updates() (ajenti.ui.UIElement method), 32  
 client (ajenti.ui.UIElement attribute), 32  
 clone() (ajenti.ui.UIElement method), 32  
 CollectionAutoBinding (class in ajenti.ui.binder), 36  
 CollectionElement (class in ajenti.ui.binder), 37  
 command() (ajenti.plugins.services.api.Service method), 42  
 config (ajenti.plugins.dashboard.api.DashboardWidget attribute), 41  
 config (in module ajenti), 25  
 configurable (ajenti.plugins.dashboard.api.DashboardWidget attribute), 41  
 ConfigurableWidget (class in ajenti.plugins.dashboard.api), 41  
 container (ajenti.plugins.dashboard.api.DashboardWidget attribute), 41  
 contains() (ajenti.ui.UIElement method), 32

- context (ajenti.api.BasePlugin attribute), 26
  - context (ajenti.plugins.PluginManager attribute), 30
  - CRASH (ajenti.plugins.tasks.api.TaskResult attribute), 44
  - create() (ajenti.ui.UI method), 31
  - create\_classconfig() (ajenti.api.BasePlugin method), 26
  - create\_config() (ajenti.plugins.dashboard.api.ConfigurableWidget method), 41
- ## D
- DashboardWidget (class in ajenti.plugins.dashboard.api), 41
  - debug (in module ajenti), 25
  - default\_classconfig (ajenti.api.BasePlugin attribute), 26
  - default\_classconfig (ajenti.users.UserManager attribute), 38
  - delete() (ajenti.ui.UIElement method), 32
  - delete\_item (ajenti.ui.binder.CollectionElement attribute), 37
  - Dependency (class in ajenti.plugins), 29
  - Dependency.Unsatisfied, 29
  - describe() (ajenti.plugins.Dependency.Unsatisfied method), 29
  - describe() (ajenti.plugins.PluginCrashed method), 29
  - describe() (ajenti.plugins.PluginFormatError method), 29
  - description (ajenti.plugins.BinaryDependency attribute), 30
  - description (ajenti.plugins.FileDependency attribute), 30
  - description (ajenti.plugins.ModuleDependency attribute), 29
  - description (ajenti.plugins.PluginDependency attribute), 30
  - DictAutoBinding (class in ajenti.ui.binder), 35
  - dispatch\_event() (ajenti.ui.UI method), 31
  - dispatch\_event() (ajenti.ui.UIElement method), 32
  - do() (ajenti.plugins.packages.api.PackageManager method), 42
- ## E
- empty() (ajenti.ui.UIElement method), 33
  - ERROR (ajenti.plugins.tasks.api.TaskResult attribute), 44
  - event() (ajenti.ui.UIElement method), 33
  - exit() (in module ajenti), 25
  - extra\_location (ajenti.plugins.PluginManager attribute), 30
  - extract() (ajenti.ui.binder.Binding class method), 35
  - extract\_context() (in module ajenti.api), 27
- ## F
- FileDependency (class in ajenti.plugins), 30
  - FileDependency.Unsatisfied, 30
  - filter (ajenti.ui.binder.BasicCollectionElement attribute), 37
  - find() (ajenti.api.sensors.Sensor static method), 28
  - find() (ajenti.ui.UI method), 31
  - find() (ajenti.ui.UIElement method), 33
  - find\_type() (ajenti.ui.UIElement method), 33
  - find\_uid() (ajenti.ui.UI method), 31
  - find\_uid() (ajenti.ui.UIElement method), 33
- ## G
- get() (ajenti.ui.binder.Binding method), 35
  - get\_all() (ajenti.plugins.PluginManager method), 30
  - get\_all() (ajenti.plugins.services.api.ServiceMultiplexor method), 43
  - get\_class() (ajenti.plugins.tasks.api.TaskDefinition method), 43
  - get\_implementations() (ajenti.plugins.PluginManager method), 30
  - get\_instance() (ajenti.plugins.PluginContext method), 30
  - get\_instances() (ajenti.plugins.PluginContext method), 30
  - get\_lists() (ajenti.plugins.packages.api.PackageManager method), 42
  - get\_name() (ajenti.ipc.IPCHandler method), 28
  - get\_name() (ajenti.users.PermissionProvider method), 37
  - get\_one() (ajenti.plugins.services.api.ServiceManager method), 43
  - get\_one() (ajenti.plugins.services.api.ServiceMultiplexor method), 43
  - get\_order() (ajenti.plugins.PluginManager method), 30
  - get\_permissions() (ajenti.users.PermissionProvider method), 37
  - get\_plugins\_root() (ajenti.plugins.PluginManager method), 30
  - get\_profiles() (in module ajenti.profiler), 31
  - get\_progress() (ajenti.plugins.tasks.api.Task method), 43
  - get\_sync\_provider() (ajenti.users.UserManager method), 38
  - get\_template() (ajenti.ui.binder.CollectionAutoBinding method), 36
  - get\_variants() (ajenti.api.sensors.Sensor method), 28
- ## H
- handle() (ajenti.ipc.IPCHandler method), 29
  - has\_permission() (ajenti.users.UserManager method), 38
  - has\_updates() (ajenti.ui.UI method), 31
  - has\_updates() (ajenti.ui.UIElement method), 33
  - hash\_password() (ajenti.users.UserManager method), 38
  - hash\_passwords() (ajenti.users.UserManager method), 38
  - hidden (ajenti.plugins.dashboard.api.DashboardWidget attribute), 41
  - hidden (ajenti.plugins.tasks.api.Task attribute), 43
- ## I
- icon (ajenti.plugins.dashboard.api.DashboardWidget attribute), 41
  - icon (ajenti.plugins.services.api.Service attribute), 42
  - id (ajenti.api.sensors.Sensor attribute), 28
  - id (ajenti.ui.UIElement attribute), 33

- index (ajenti.plugins.dashboard.api.DashboardWidget attribute), 41
- inflate() (ajenti.ui.UI method), 32
- init() (ajenti.api.BasePlugin method), 26
- init() (ajenti.api.sensors.Sensor method), 28
- init() (ajenti.plugins.dashboard.api.ConfigurableWidget method), 41
- init() (ajenti.plugins.packages.api.PackageManager method), 42
- init() (ajenti.plugins.services.api.ServiceMultiplexor method), 43
- init() (ajenti.plugins.tasks.api.Task method), 43
- init() (ajenti.ui.UI method), 32
- init() (ajenti.ui.UIElement method), 33
- init() (in module ajenti), 25
- installation\_uid (in module ajenti), 25
- instantiate() (ajenti.plugins.PluginContext method), 30
- interface() (in module ajenti.api), 27
- invalidate() (ajenti.ui.UIElement method), 33
- IPCHandler (class in ajenti.ipc), 28
- IPCServer (class in ajenti.ipc), 29
- is\_satisfied() (ajenti.plugins.BinaryDependency method), 30
- is\_satisfied() (ajenti.plugins.FileDependency method), 30
- is\_satisfied() (ajenti.plugins.ModuleDependency method), 29
- is\_satisfied() (ajenti.plugins.PluginDependency method), 30
- ## J
- JobDefinition (class in ajenti.plugins.tasks.api), 43
- ## L
- launch() (ajenti.api.AppContext method), 26
- ListAutoBinding (class in ajenti.ui.binder), 35
- ListElement (class in ajenti.ui.binder), 37
- load() (ajenti.plugins.PluginManager method), 30
- load\_all() (ajenti.plugins.PluginManager method), 30
- load\_classconfig() (ajenti.api.BasePlugin method), 26
- load\_recursive() (ajenti.plugins.PluginManager method), 31
- ## M
- make\_report() (in module ajenti.util), 39
- measure() (ajenti.api.sensors.Sensor method), 28
- ModuleDependency (class in ajenti.plugins), 29
- ModuleDependency.Unsatisfied, 29
- ## N
- name (ajenti.plugins.dashboard.api.DashboardWidget attribute), 42
- name (ajenti.plugins.tasks.api.Task attribute), 43
- nearest() (ajenti.ui.UIElement method), 33
- new\_item (ajenti.ui.binder.CollectionElement attribute), 37
- NoImplementationsError, 27
- notify() (ajenti.api.AppContext method), 26
- notrack() (in module ajenti.api), 27
- notrack\_this() (in module ajenti.api), 27
- ## O
- on() (ajenti.ui.UIElement method), 33
- on() (in module ajenti.ui), 34
- on\_add() (ajenti.ui.binder.CollectionAutoBinding method), 36
- on\_config() (ajenti.plugins.dashboard.api.ConfigurableWidget method), 41
- on\_config\_save() (ajenti.plugins.dashboard.api.ConfigurableWidget method), 41
- on\_config\_start() (ajenti.plugins.dashboard.api.ConfigurableWidget method), 41
- on\_delete() (ajenti.ui.binder.CollectionAutoBinding method), 36
- on\_prepare() (ajenti.plugins.dashboard.api.ConfigurableWidget method), 41
- on\_start() (ajenti.plugins.dashboard.api.ConfigurableWidget method), 41
- open\_content() (ajenti.api.BasePlugin method), 26
- ## P
- p() (in module ajenti.ui), 34
- PackageInfo (class in ajenti.plugins.packages.api), 42
- Packager (class in ajenti.plugins.packages.api), 42
- pagesize (ajenti.ui.binder.CollectionElement attribute), 37
- path\_to() (ajenti.ui.UIElement method), 33
- permission (ajenti.users.SecurityError attribute), 38
- PermissionProvider (class in ajenti.users), 37
- persistent() (in module ajenti.api), 27
- platform (in module ajenti), 25
- platform\_select() (in module ajenti.util), 38
- platform\_string (in module ajenti), 25
- platform\_unmapped (in module ajenti), 25
- plugin() (in module ajenti.api), 26
- PluginContext (class in ajenti.plugins), 30
- PluginCrashed, 29
- PluginDependency (class in ajenti.plugins), 30
- PluginDependency.Unsatisfied, 30
- PluginFormatError, 29
- PluginInfo (class in ajenti.api), 25
- PluginLoadError, 29
- PluginManager (class in ajenti.plugins), 30
- populate() (ajenti.ui.binder.Binder method), 36
- populate() (ajenti.ui.binder.Binding method), 35
- populate() (ajenti.ui.binder.CollectionAutoBinding method), 36

- populate() (ajenti.ui.binder.DictAutoBinding method), 35
  - populate() (ajenti.ui.binder.ListAutoBinding method), 35
  - populate() (ajenti.ui.binder.PropertyBinding method), 35
  - post\_bind (ajenti.ui.binder.BasicCollectionElement attribute), 37
  - post\_clone() (ajenti.ui.UIElement method), 33
  - post\_item\_bind (ajenti.ui.binder.BasicCollectionElement attribute), 37
  - post\_item\_update (ajenti.ui.binder.BasicCollectionElement attribute), 37
  - profile\_end() (in module ajenti.profiler), 31
  - profile\_start() (in module ajenti.profiler), 31
  - profiled() (in module ajenti.profiler), 31
  - property\_definitions (ajenti.ui.UIElement attribute), 33
  - PropertyBinding (class in ajenti.ui.binder), 35
  - public() (in module ajenti.util), 38
- ## R
- reason() (ajenti.plugins.BinaryDependency.Unsatisfied method), 30
  - reason() (ajenti.plugins.Dependency.Unsatisfied method), 29
  - reason() (ajenti.plugins.FileDependency.Unsatisfied method), 30
  - reason() (ajenti.plugins.ModuleDependency.Unsatisfied method), 29
  - reason() (ajenti.plugins.PluginDependency.Unsatisfied method), 30
  - refresh() (ajenti.plugins.packages.api.PackageManager method), 42
  - register\_implementation() (ajenti.plugins.PluginManager method), 31
  - register\_interface() (ajenti.plugins.PluginManager method), 31
  - remove() (ajenti.ui.UIElement method), 34
  - render() (ajenti.ui.UI method), 32
  - render() (ajenti.ui.UIElement method), 34
  - require\_permission() (ajenti.users.UserManager method), 38
  - reset() (ajenti.ui.binder.Binder method), 36
  - resolve\_path() (ajenti.plugins.PluginManager method), 31
  - restart() (ajenti.plugins.services.api.Service method), 42
  - restart() (in module ajenti), 25
  - restrict() (in module ajenti.users), 37
  - reverse\_event() (ajenti.ui.UIElement method), 34
  - rootcontext() (in module ajenti.api), 27
  - run() (ajenti.ipc.IPCServer method), 29
  - run() (ajenti.plugins.tasks.api.Task method), 43
- ## S
- satisfied() (ajenti.plugins.Dependency method), 29
  - save() (ajenti.plugins.tasks.api.JobDefinition method), 43
  - save() (ajenti.plugins.tasks.api.TaskDefinition method), 43
  - save\_classconfig() (ajenti.api.BasePlugin method), 26
  - save\_config() (ajenti.plugins.dashboard.api.DashboardWidget method), 42
  - search() (ajenti.plugins.packages.api.PackageManager method), 42
  - SecurityError, 38
  - Sensor (class in ajenti.api.sensors), 28
  - server (in module ajenti), 25
  - Service (class in ajenti.plugins.services.api), 42
  - ServiceManager (class in ajenti.plugins.services.api), 43
  - ServiceMultiplexor (class in ajenti.plugins.services.api), 43
  - session (ajenti.api.AppContext attribute), 26
  - set() (ajenti.ui.binder.Binding method), 35
  - set\_page() (ajenti.ui.binder.CollectionAutoBinding method), 36
  - set\_password() (ajenti.users.UserManager method), 38
  - set\_progress() (ajenti.plugins.tasks.api.Task method), 43
  - set\_sync\_provider() (ajenti.users.UserManager method), 38
  - setup() (ajenti.ui.binder.Binder method), 36
  - sorting (ajenti.ui.binder.CollectionElement attribute), 37
  - source (ajenti.plugins.services.api.Service attribute), 42
  - start() (ajenti.ipc.IPCServer method), 29
  - start() (ajenti.plugins.services.api.Service method), 42
  - start() (ajenti.plugins.tasks.api.Task method), 43
  - stop() (ajenti.plugins.services.api.Service method), 42
  - str\_fsize() (in module ajenti.util), 38
  - str\_timedelta() (in module ajenti.util), 38
  - style (ajenti.ui.UIElement attribute), 34
  - SUCCESS (ajenti.plugins.tasks.api.TaskResult attribute), 44
- ## T
- Task (class in ajenti.plugins.tasks.api), 43
  - TaskDefinition (class in ajenti.plugins.tasks.api), 43
  - TaskError, 43
  - TaskResult (class in ajenti.plugins.tasks.api), 44
  - timeout (ajenti.api.sensors.Sensor attribute), 28
  - title (ajenti.plugins.configurator.api.ClassConfigEditor attribute), 42
  - track() (in module ajenti.api), 27
  - typeid (ajenti.plugins.configurator.api.ClassConfigEditor attribute), 42
  - typeid (ajenti.plugins.dashboard.api.DashboardWidget attribute), 42
  - typeid (ajenti.ui.binder.CollectionElement attribute), 37
  - typeid (ajenti.ui.binder.ListElement attribute), 37
  - typeid (ajenti.ui.UIElement attribute), 34
- ## U
- ui (ajenti.plugins.tasks.api.Task attribute), 43



UI (class in ajenti.ui), 31  
 UIElement (class in ajenti.ui), 32  
 unpopulate() (ajenti.ui.binder.Binder method), 36  
 unpopulate() (ajenti.ui.binder.Binding method), 35  
 unpopulate() (ajenti.ui.binder.CollectionAutoBinding  
 method), 36  
 unpopulate() (ajenti.ui.binder.DictAutoBinding method),  
 35  
 unpopulate() (ajenti.ui.binder.ListAutoBinding method),  
 35  
 update() (ajenti.ui.binder.Binder method), 36  
 update() (ajenti.ui.binder.Binding method), 35  
 update() (ajenti.ui.binder.CollectionAutoBinding  
 method), 36  
 update() (ajenti.ui.binder.DictAutoBinding method), 35  
 update() (ajenti.ui.binder.ListAutoBinding method), 35  
 update() (ajenti.ui.binder.PropertyBinding method), 35  
 user (ajenti.api.AppContext attribute), 26  
 UserManager (class in ajenti.users), 38

## V

vacuum\_instances() (ajenti.plugins.PluginContext  
 method), 30  
 value (ajenti.plugins.Dependency attribute), 29  
 value() (ajenti.api.sensors.Sensor method), 28  
 values (ajenti.ui.binder.BasicCollectionElement at-  
 tribute), 37  
 version (in module ajenti), 25  
 visible (ajenti.ui.UIElement attribute), 34