Kwapi Documentation

Release

OpenStack, LLC

May 29, 2014

Contents

Kwapi is a framework designed for acquiring energy consumption metrics. It allows to upload metrics from various wattmeters to Ceilometer.

Its architecture is based on a layer of drivers, which retrieve measurements from wattmeters, and a layer of plugins that collect and process them. The communication between these two layers goes through a bus. In the case of a distributed architecture, a plugin can listen to several drivers at remote locations.

Drivers and plugins are easily extensible to support other types of wattmeters, and provide other services.

CHAPTER 1

What is the purpose of the project and vision for it?

Kwapi could be used to do:

- Energy monitoring of data centers
- Usage-based billing
- Efficient scheduling

It aims at supporting various wattmeters, being scalable and easily extensible.

This documentation offers information on how Kwapi works and how to contribute to the project.

CHAPTER 2

Table of contents

2.1 Installing

2.1.1 Installing Kwapi

1. Clone the Kwapi git repository to the management server:

```
$ git clone https://github.com/stackforge/kwapi.git
```

2. As a user with `root` permissions or `sudo` privileges, run the Kwapi installer and copy the configuration files:

```
$ pip install kwapi
$ cp -r kwapi/etc/kwapi /etc/
```

2.1.2 Running Kwapi services

Start the drivers on all the machines that can access wattmeters:

```
$ kwapi-drivers
```

Start the forwarder on a remote machine (optional):

```
$ kwapi-forwarder
```

Start the API plugin if you want to use Ceilometer:

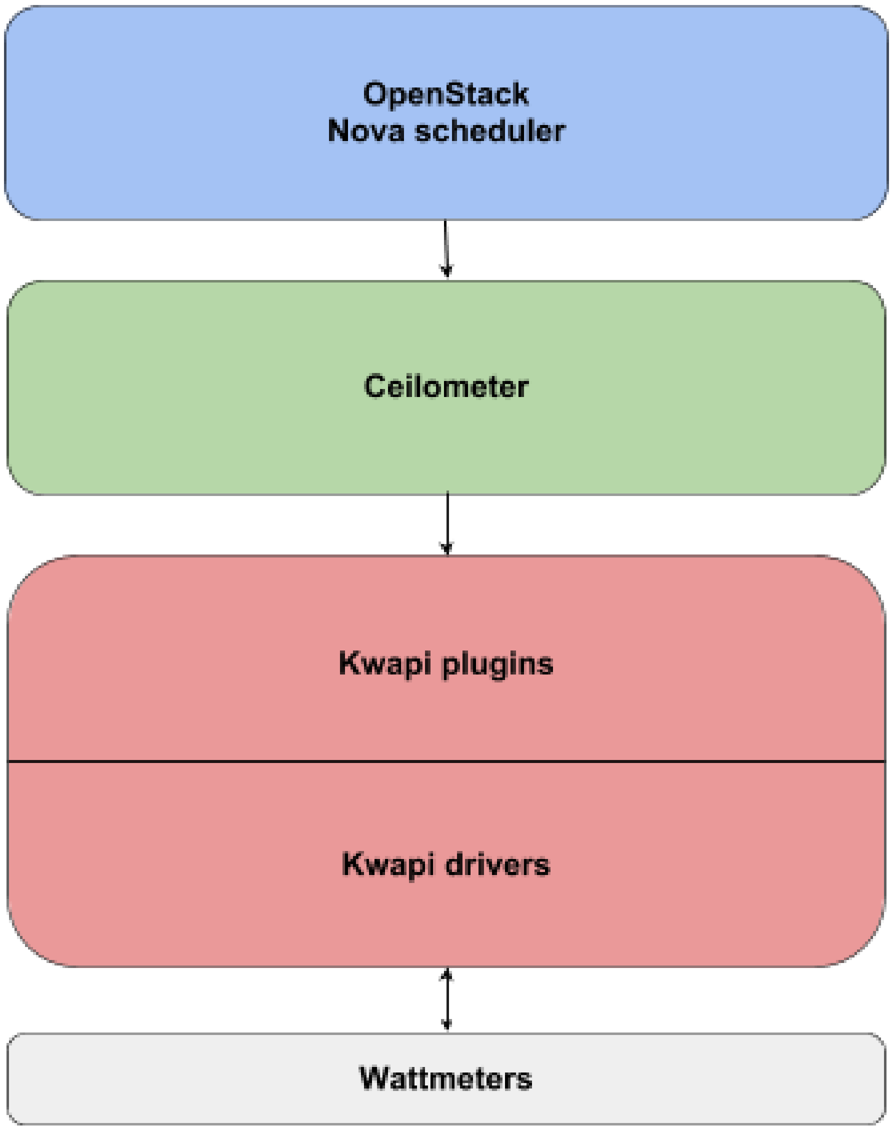
```
$ kwapi-api
```

Start the RRD plugin if you want to display graphs in a web browser:

```
$ kwapi-rrd
```

2.2 System Architecture

Overview of the global layered architecture:



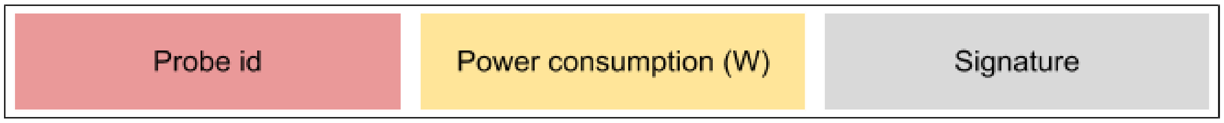
2.2.1 Kwapi drivers

Kwapi supports different kinds of wattmeters (IPMI, Eaton PDU, Wattsup, etc). Wattmeters communicate via IP networks or serial links. Each wattmeter has one or more sensors (probes). Wattmeters send their values quite often (each second), and they are listen by wattmeter drivers. Wattmeter drivers are derived from a Driver superclass, itself derived from Thread. So drivers are threads. At least one driver thread is instantiated for each wattmeter. Their constructors takes as arguments a list of probe IDs, and kwargs (specific arguments).

Driver threads roles are:

1. Setting up wattmeter.
2. Listening and decoding received data.
3. Calling a driver superclass method with measurements as argument. This method appends signature to the measurements, and publishes them on the bus.

Message format:



Driver manager

The driver manager is the loader and the checker of driver threads. It loads all drivers according the configuration file, and checks regularly that driver threads are alive. In case of crash, the event is logged and the driver thread is reloaded. We can imagine that a driver will crash if a technician unplug a wattmeter, for example.

Bus

Currently, the internal Kwapi bus is ZeroMQ. Publishers are driver threads, and subscribers are plugins.

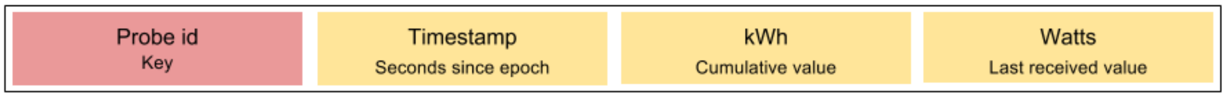
2.2.2 Kwapi plugins

Kwapi API plugin

API plugin allows Ceilometer pollster to get consumption data through a REST API. This plugin contains a collector that computes kWh, and an API based on Flask.

Collector

The collector stores these values for each probe:



Fields:

- Probe id: could be the hostname of the monitored machine. But it is a bit more complicated because a probe can monitor several machines (PDU).
- Timestamp: is updated when a new value is received.
- KWh: is computed by taking into account the new watt value, and the elapsed time since the previous update. It allows Ceilometer to compute average consumption for a given duration (knowing the kWh consumed and the time elapsed since its last check).
- Watts: offers the possibility to know instantaneous consumption of a device, without having to query two times a probe in a small interval to deduce it. This could be especially useful if a probe has a large refresh interval: there is no need to wait its next value.

No history is kept because Ceilometer already has a storage architecture. The collector is cleaned periodically to prevent a deleted probe from being stored indefinitely in the collector. So when a probe has not been updated for a long time, it is deleted.

API

Verb	URL	Parameters	Expected result
GET	/v1/		Returns detailed information about this specific version of the API.
GET	/v1/probe-ids/		Returns all known probe IDs.
GET	/v1/probes/		Returns all information about all known probes.
GET	/v1/probes/<probe>/	probe id	Returns all information about this probe (id, timestamp, kWh, W).
GET	/v1/probes/<probe>/<meter>/	probe id, meter { timestamp, kwh, w }	Returns the probe meter value.

Authentication

The pollster provides a token (X-Auth-Token). The API plugin checks the token (Keystone request), and if the token is valid, requested data are sent. Responses are not signed because Ceilometer trusts Kwapi plugin.

Ceilometer pollster

The API plugin is queried by a Ceilometer pollster. The Ceilometer pollster is started periodically by Ceilometer central agent. It knows the Kwapi URL by doing a Keystone request (endpoint-get). It queries probe values through Kwapi API, using the GET /v1/probes/ call, so that it gets all detailed informations about all probes in just one query. For each probe, it creates a counter object and publishes it on the Ceilometer bus.

Published counters:

- Energy (cumulative type): represents kWh.
- Power (gauge type): represents watts.

Counter timestamps are Kwapi timestamps, so that Ceilometer doesn't store wrong data if a probe is not updated. Ceilometer handles correctly the case where a probe value is reset (kWh decrease), because of its cumulative type.

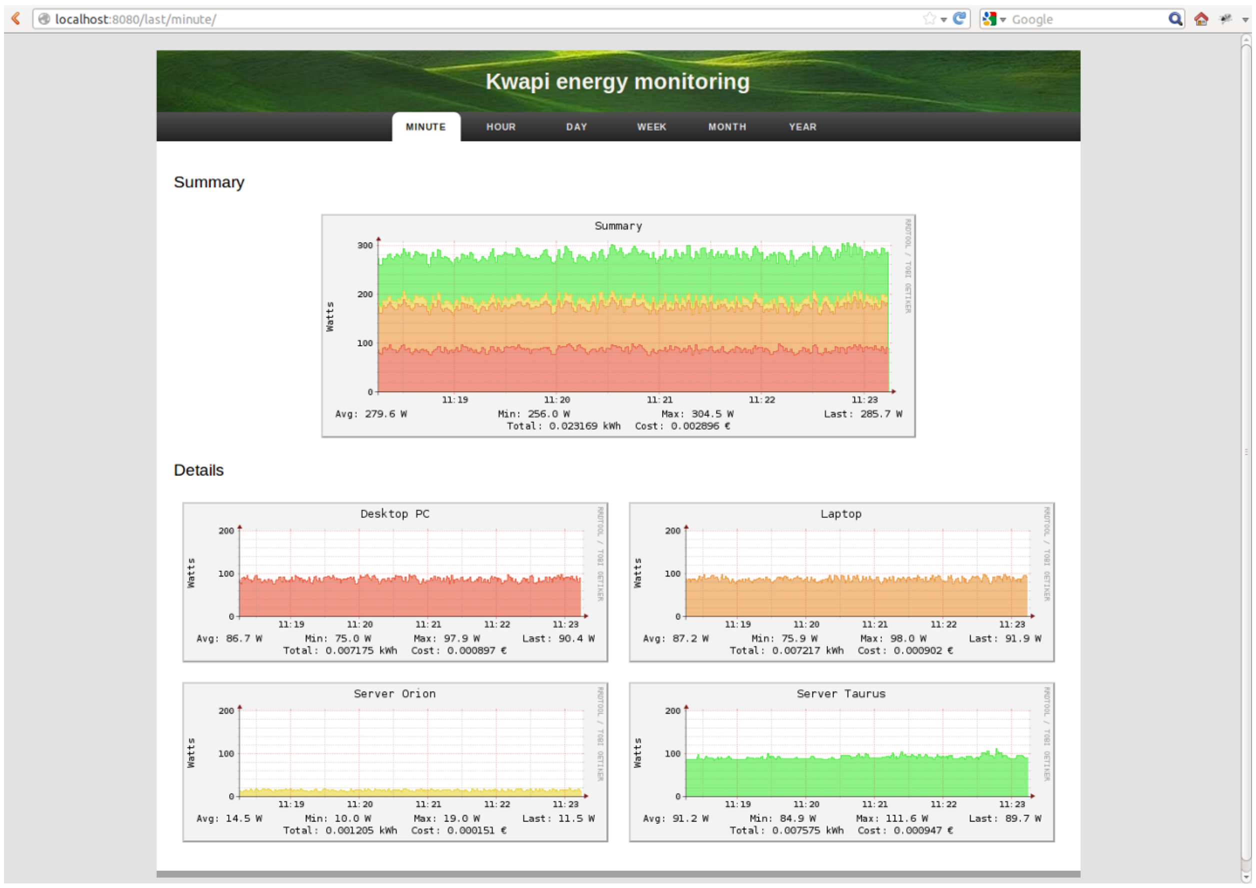
Kwapi RRD plugin

Web interface

The visualization plugin provides a web interface with power consumption graphs. It is based on Flask and RRDtool.

Verb	URL	Parameters	Expected result
GET	/last/<period>/	period { minute, hour, day, week, month, year }	Returns a webpage with a summary graph and all probe graphs.
GET	/probe/<probe>/	probe id	Returns a webpage with all graphs about this probe (all periods).
GET	/graph/<period>/	period { minute, hour, day, week, month, year }	Returns a summary graph about this period.
GET	/graph/<period>/<probe>/	period { minute, hour, day, week, month, year }, probe id	Returns a graph about this probe.

Webpage with a summary graph and all probe graphs:



```
Summary
Avg: 279.6 W    Min: 256.0 W    Max: 304.5 W    Last: 285.7 W
        Total: 0.023169 kWh   Cost: 0.002896 €

Desktop PC
Avg: 86.7 W    Min: 75.0 W    Max: 97.9 W    Last: 90.4 W
        Total: 0.007175 kWh   Cost: 0.000897 €

Laptop
Avg: 87.2 W    Min: 75.9 W    Max: 98.0 W    Last: 91.9 W
        Total: 0.007217 kWh   Cost: 0.000902 €

Server Orion
Avg: 14.5 W    Min: 10.0 W    Max: 19.0 W    Last: 11.5 W
        Total: 0.001205 kWh   Cost: 0.000151 €

Server Taurus
Avg: 91.2 W    Min: 84.9 W    Max: 111.6 W    Last: 89.7 W
        Total: 0.007575 kWh   Cost: 0.000947 €
```

In the menu bar, you can choose the period for which you want to display graphs (last minutes, hour, day, week, month or year). By clicking on a probe, you can display all graphs available for this probe, with different resolutions.

Graphs

The summary graph shows the total power consumption (sum of all the probes). Each colour corresponds to a probe.

The legend contains:

- Minimum, maximum, average and last power consumption.
- Energy consumed (kWh).
- Cost.

File sizes:

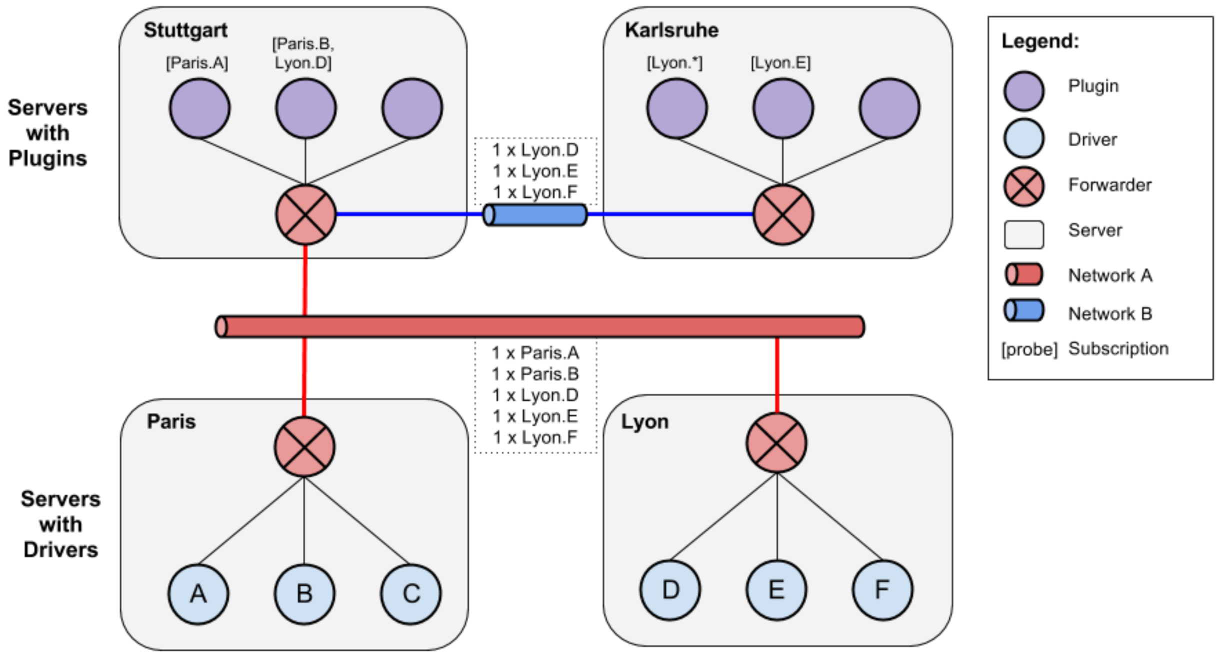
- RRD file: 10 Ko.
- Probe graph: 12 Ko.
- Summary graph: 24 Ko.

A cache mechanism prevents graphs from being rebuilt uselessly.

2.2.3 Kwapi forwarder

The forwarder aims at decreasing the network traffic: if multiple plugins listen the same probe, the metric is sent once on the network, and the forwarder duplicate it and sends a copy to each listeners. The forwarder can also be installed on a gateway machine, in order to connect isolated networks.

The following diagram shows these two features:



Using the forwarder is optional, and the plugins can be configured to subscribe directly to the drivers. Direct subscribing without using the forwarder is recommanded if the drivers and the plugins are running on the same machine.

2.3 Configuration Options

2.3.1 Kwapi drivers specific

The following table lists the Kwapi drivers specific options in the drivers configuration file. Please note that Kwapi uses openstack-common extensively, which requires that the other parameters are set appropriately. For information we are listing the configuration elements that we use after the Kwapi drivers specific elements.

Parameter	Default	Note
probes_endpoint	ipc:///tmp/kwapi-drivers	Endpoint where the drivers send their measurements ipc://<file> or tcp://<host>:<port>
enable_signing	true	Enable message signing between drivers and plugins
metering_secret	change this or be hacked	Secret value for signing metering messages
check_drivers_interval	60	Check drivers at the specified interval and restart them if they are crashed

The configuration file contains a section for each wattmeter.

A sample configuration file can be found in drivers.conf.

2.3.2 Kwapi plugin API specific

The following table lists the Kwapi API specific options in the API configuration file. Please note that Kwapi uses openstack-common extensively, which requires that the other parameters are set appropriately. For information we are listing the configuration elements that we use after the Kwapi API specific elements.

Parameter	Default	Note
api_port	5000	API port
probes_endpoint	ipc:///tmp/kwapi-forwarder	Endpoint where the measurements are received
signature_checking	true	Enable the verification of signed metering messages
driver_metering_secret	change this or be hacked	Secret value for verifying signed metering messages
acl_enabled	true	Check the Keystone tokens provided by the clients
policy_file	/etc/kwapi/policy.json	Policy file
cleaning_interval	300	Delete the probes that have not been updated during the specified interval

A sample configuration file can be found in api.conf.

Keystone Middleware Authentication

The following table lists the Keystone middleware authentication options which are used to get admin token. Please note that these options need to be under [keystone_authtoken] section.

Parameter	Default	Note
auth_host		The host providing the Keystone service API endpoint for validating and requesting tokens
auth_port	35357	The port used to validate tokens
auth_protocol	https	The protocol used to validate tokens
auth_uri	auth_protocol://auth_host:auth_port	The full URI used to validate tokens
admin_token		Either this or the following three options are required. If set, this is a single shared secret with the Keystone configuration used to validate tokens.
admin_user		User name for retrieving admin token
admin_password		Password for retrieving admin token
admin_tenant_name		Tenant name for retrieving admin token
signing_dir		The cache directory for signing certificate
certfile		Required if Keystone server requires client cert
keyfile		Required if Keystone server requires client cert. This can be the same as certfile if the certfile includes the private key.

2.3.3 Kwapi plugin RRD specific

The following table lists the Kwapi RRD specific options in the RRD configuration file. Please note that Kwapi uses openstack-common extensively, which requires that the other parameters are set appropriately. For information we are listing the configuration elements that we use after the Kwapi RRD specific elements.

Parameter	Default	Note
rrd_port	8080	Port used to display webpages
probes_endpoint	ipc:///tmp/kwapi-forwarder	Endpoint where the measurements are received
signature_checking	true	Enable the verification of signed metering messages
driver_metering_secret	change this or be hacked	Secret value for verifying signed metering messages
png_dir	/var/lib/kwapi/kwapi-png	The directory where are stored PNG files
rrd_dir	/var/lib/kwapi/kwapi-rrd	The directory where are stored RRD files
currency	€	The currency symbol used in graphs
kwh_price	0.125	The kWh price used in graphs
hue	100	The hue of the graphs
max_watts	200	The maximum value of the summary graph
refresh_interval	5	The webpage auto-refresh interval

A sample configuration file can be found in rrd.conf.

2.3.4 General options

The following is the list of openstack-common options that we use:

Parameter	Default	Note
log_file		Log output to a named file
verbose	true	Print more verbose output

2.3.5 Kwapi forwarder specific

The following table lists the Kwapi forwarder specific options in the forwarder configuration file. Please note that Kwapi uses openstack-common extensively, which requires that the other parameters are set appropriately. For information we are listing the configuration elements that we use after the Kwapi forwarder specific elements.

Parameter	Default	Note
forwarder_endpoint	ipc:///tmp/kwapi-forwarder	Endpoint where the measurements are forwarded and where the plugins subscriptions are received
probes_endpoint	ipc:///tmp/kwapi-drivers	Endpoint where the drivers send their measurements. ipc://<file> or tcp://<host>:<port>

The configuration file contains a section for each wattmeter.

A sample configuration file can be found in forwarder.conf.

2.4 Contributing to Kwapi

2.4.1 Joining the Project

Contributor License Agreement

In order to contribute to the Kwapi project, you need to have signed OpenStack's contributor's agreement.

See also:

- http://wiki.openstack.org/HowToContribute
- http://wiki.openstack.org/CLA

LaunchPad Project

Most of the tools used for OpenStack depend on a launchpad.net ID for authentication. After signing up for a launchpad account, join the "openstack" team to have access to the mailing list and receive notifications of important events.

See also:

- http://launchpad.net
- http://launchpad.net/kwapi
- http://launchpad.net/~openstack

2.4.2 Project Hosting Details

Bug tracker https://bugs.launchpad.net/kwapi

Mailing list http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack-dev (prefix subjects with `[energy]` for faster responses)

Code Hosting https://github.com/stackforge/kwapi

Code Review https://review.openstack.org/#/q/status:open+project:stackforge/kwapi,n,z

See also:

- *Joining the Project*

2.4.3 Areas to Contribute

Drivers

Kwapi aims at supporting various wattmeters. If you have a non-supported wattmeter, you can easily contribute by writing a new one.

Plugins

Kwapi plugins process the metrics. You can contribute by writing new plugins to bring new functionnalities.

Testing

The first version of Kwapi has not yet unit tests and has not seen much run-time in real environments. Setting up a copy of Kwapi to monitor a real OpenStack installation or to perform some load testing would be especially helpful.

2.4.4 Working with the Source

Setting up a Development Sandbox

1. Set up a server or virtual machine to run OpenStack using devstack.
2. Clone the kwapi project to the machine:

```
$ cd /opt/stack
$ git clone https://github.com/stackforge/kwapi.git
$ cd ./kwapi
```

3. Once this is done, you need to setup the review process:

   ```
   $ git remote add gerrit ssh://<username>@review.openstack.org:29418/stackforge/kwapi.git
   ```

4. If you are preparing a patch, create a topic branch and switch to it before making any changes:

   ```
   $ git checkout -b TOPIC-BRANCH
   ```

Code Reviews

Kwapi uses the OpenStack review process for all code and developer documentation contributions. Code reviews are managed through gerrit.

See also:

- http://wiki.openstack.org/GerritWorkflow
- OpenStack Gerrit instance.

2.5 Glossary

driver Software thread running querying a wattmeter and sending the results to the plugins.

forwarder Component that forwards plugins subscriptions and metrics. Used to minimize the network traffic, or to connect isolated networks through a gateway.

plugin An action triggered whenever a meter reaches a certain threshold.

probe A wattmeter sensor. A wattmeter can have only one probe (usually the IPMI cards), or multiple probes (usually the PDUs).

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*