

Provisioning API

In this article

- [Section overview](#)
- [Execution Logs tab](#)

Section overview

The following section allows **configuring and monitoring of hooks for Provisioning API**.

Provisioning API provides a mechanism for real-time integration with 3rd party systems, including softswitches, gateways, and CRM systems. It calls pre-defined handlers on an occurrence of specific events in the system. The handlers are allowed to modify data, forbid or allow the action or simply process given event.

For detailed information about Provisioning API functionality, go to [APIs](#) section of our User Guide.

To configure handlers and check their call log, go to the **Integration > Provisioning API**.



Tip

The full list of **Provisioning API** parameters matches with [CoreAPI](#) and they are available upon an individual request of your current clients.

Screenshot: Provisioning section

Provisioning API					
Execution Logs					
New Handler					
Rows 1 – 2 of 2					
20					
Page 1 of 1					
ID	Priority	Name	Event	Handler	
1	1	Block client on the switch	Clients Balance Became <= 0	Script /usr/local/script	
2	1	Create client in the CRM	Accounts Create	HTTP crm.net/client-add	
About 0.0575s					
© 2004-2019 JeraSoft. All Rights Reserved.					

Column Name	Description
ID	Handler's identification number
Priority	Priority of handlers execution
Name	Handler's title
Event	Description of the handler event
Handler	Category of the handler that is used and location. There are two types of handlers that can be used: • HTTP scripts, called via POST requests (used in most cases) • Local server scripts, called locally on the server (used in very specific cases)

The list of section functional **buttons/icons** is as follows:

Button/Icon	Description
	Allows creating a new handler
	Identifies a disabled status of a handler
	Identifies an enabled status of a handler
	Identifies an archived status of a handler
	Allows viewing details of a target handlers' performance Execution Logs tab for a respective handler
	Allows deleting a handler from the system

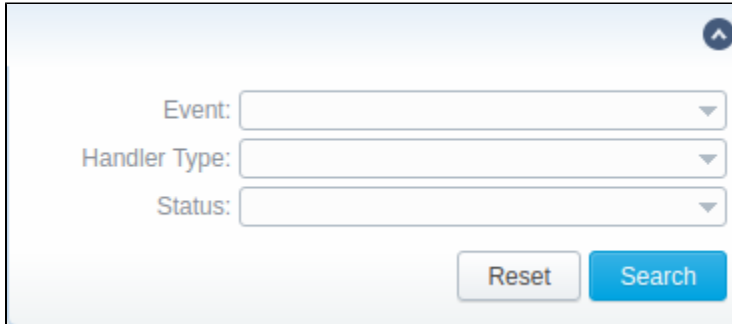
✓ Tip

For a quick switch between **enabled** and **disabled** statuses, click on a respective *status icon* in the section. However, to change **archived** status, you need to do it from a handler edit form

Advanced Search

Advanced Search drop-down menu, located in the top right corner of the section, is called to facilitate easy access to required information. By clicking on a red downward arrow  icon, the following drop-down menu is displayed:

Screenshot: Advanced Search drop-down menu

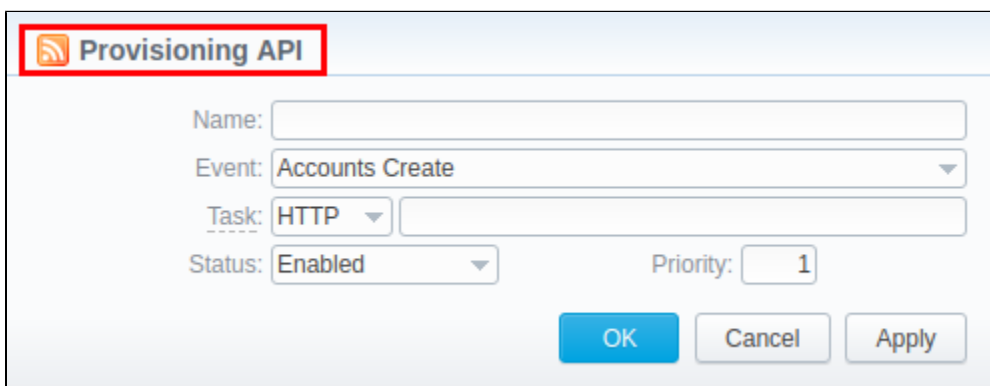


Field	Description
Event	Select from a list of all possible handler events
Handler Type	Indicate a type of handler: • script • HTTP
Status	Choose a target status: • Enabled • Disabled • Archived or leave this field blank. In this case, both enabled and disabled handlers will be displayed. This field is empty by default

Creating a New Handler

To start with provisioning, you need to create a handler manually. Click the **New Handler** button and specify respective parameters in the appeared pop-up window:

Screenshot: Provisioning section/Handler adding form



Field	Description
Name	Specify a particular title for a handler
Event	Specify a handler event from the following list:
	Clients • create • update • delete • archive • custom fields update • balance became >=0 • balance became <=0
	Accounts • create • update • delete
	Clients Packages • assign • activate • deactivate • renew • close
Task	Determine a type of handler and details: • script - here you need to specify the path where the following script is located, for example, user/local/vcs/script.py. • http:// - here specify the port and method, for example: 120.0.0.1:5000/api.
Status	Choose the state of the handler: • enabled - select it to make a handler active; • disabled - select it to unable a handler; • archived - select it to archive a handler.
Priority	Establish an order of handler performing. Note: The handler with 1 priority will precede all other handlers in order.

Attention

In **VCS 3.17.0**, to prevent performance degradation and data inconsistency, affected by external side, **Before** event type has been removed from the section.

✔ Best practice example

There is an example based on **http://handler** usage.

2. Open the **Provisioning** section and start creating a handler.

- Specify the name, type, and status.
- In the **Event** field, select **Clients Create** event from the drop-down list.
- In the **Task** field, indicate **http://** type and determine the port and method, for example, **120.0.0.1:5000/api**.
- Click **Apply**.

Find an example of the **http://handler** below:

```
from flask import Flask, request
import json
app = Flask(__name__)
@app.route("/api", methods=['GET', 'POST'])
def api():
    data = json.loads(request.data)
    return json.dumps(data)
if __name__ == "__main__":
    app.run()
```

⚠ Attention

To put a handler into action, you need to **restart** the **Cache Manager**. To do so, click on the corresponding icon in **Task Scheduler** section

Execution Logs tab

You can access the details about handler execution in this tab. For more details, check out a related article: [Execution Logs](#).

⚠ Warning

Please note, the Provisioning functionality is experimental and may be changed completely in future releases.