WebFaction API Documentation

Swarma Limited - WebFaction is a service of Swarma Limited
## CONTENTS

1 Introduction 3

2 Tutorial 5
   2.1 Getting Started ......................................................... 5
   2.2 Creating an Email Address ........................................... 5
   2.3 Installing an Application ............................................ 6
   2.4 Packaging the Install Script for the Control Panel ............... 7
   2.5 Additional Resources ............................................... 8

3 API Reference 9
   3.1 General ................................................................. 9
   3.2 Email ........................................................................... 10
   3.3 Websites and Domains .................................................. 14
   3.4 Applications ................................................................ 18
   3.5 Cron ............................................................................ 19
   3.6 DNS ............................................................................ 19
   3.7 Databases ..................................................................... 21
   3.8 Files ............................................................................ 24
   3.9 Shell Users .................................................................... 25
   3.10 Servers ........................................................................ 26
   3.11 Miscellaneous ......................................................... 26

4 Application Types 29

Index 33
Contents:
INTRODUCTION

The WebFaction API (Application Programming Interface) is a powerful XML-RPC interface for managing many control panel and account tasks. With the WebFaction API, you can automate application installation, email address configuration, and more.

Like other XML-RPC APIs, the WebFaction API works by sending a short piece of XML over HTTP. Luckily, many languages have XML-RPC libraries to make requests quick and painless.

For example, you can send an XML-RPC request using Python's xmlrpclib module:

```python
>>> import xmlrpclib
>>> server = xmlrpclib.ServerProxy('https://api.webfaction.com/')
>>> session_id, account = server.login('widgetsco', 'widgetsrock')
```

Or with Ruby's xmlrpc package:

```ruby
>> require 'xmlrpc/client'
=> true
>> require 'pp'
=> true
>> server = XMLRPC::Client.new2("https://api.webfaction.com/")
#<XMLRPC::Client:0x5b1698 @cookie=nil, @create=nil, @port=443>
>> pp server.call("login", "widgetsco", "widgetsrock")
["ca4c008c24c0de9c9c8",
 {"mail_server"=>"Mail5",
  "web_server"=>"Web55",
  "username"=>"widgetsco",
  "id"=>687,
  "home"=>"/home"}]
=> nil
```

To learn more about XML-RPC and find an implementation in your favorite language, please visit XMLRPC.com.
The WebFaction API allows you to write scripts to automate certain tasks that you would normally accomplish with
the control panel or an SSH session.

For instance, you could use the API to write a script to configure lots of email addresses, instead of creating them one
by one in the control panel.

You can also use the API to automate the installation of any application that you like, and you can share the resulting
install script to allow other users to use it in one click.

2.1 Getting Started

The API is a set of methods available via XML-RPC calls at the URL https://api.webfaction.com/. In
this documentation we will use the Python programming language to talk to the API, but you can use any language
that you want, provided that it has an XML-RPC library.

First, connect to the server and login:

```python
>>> import xmlrpclib
>>> server = xmlrpclib.ServerProxy('https://api.webfaction.com/')
>>> session_id, account = server.login('test5', 'password')
>>> account
{'username': 'test5', 'home': '/home2', 'id': 237}
```

The username and password passed to the login method are those used to login to the control panel.

As you can see, the login method returns a tuple. The first element is a string containing a session ID that you
will need to pass to all subsequent methods. The second element is a dictionary containing various data about your
account, including the base home directory (usually /home or /home2).

2.2 Creating an Email Address

Let’s create a new email address for our account:

```python
>>> server.create_email(session_id, 'user@mydomain.example', 'test5')
{'autoresponder_from': '',
 'autoresponder_message': '',
 'autoresponder_on': 0,
 'autoresponder_subject': '',
 'email_address': 'user@mydomain.example',
 'id': 2037,
 'script_machine': ''
}
```
The `create_email` method takes the session ID, the email address and a string of comma separated target mailboxes. It returns a dictionary containing various data about the newly created email address.

This call is equivalent to creating the email address from the control panel, but the advantage is that you can script it.

### 2.3 Installing an Application

Now let’s see which methods we can use to install an application. For this tutorial, let’s install a Joomla application.

First, create a Static/CGI/PHP application:

```python
>>> server.create_app(session_id, 'my_joomla_app', 'static_php56', False, '', False)
{'autostart': False, 'extra_info': '', 'id': 892545, 'machine': 'Web31', 'name': 'my_joomla_app', 'open_port': False, 'port': 0, 'type': 'static_php56'}
```

The `create_app` method uses the following parameters:

- `session_id` – session ID returned by `login`
- `name` (string) – name of the application
- `type` (string) – type of the application
- `autostart` (boolean) – whether the app should get restarted with an `autostart.cgi` script
- `extra_info` (string) – additional information required by the application (for example, a file path). If `extra_info` is not required by the application, it is ignored.
- `open_port` (boolean) – for applications that listen on a port, whether the port should be open on shared and dedicated IP addresses

Calling `create_app` is equivalent to creating the application through the control panel: it creates the directory and configures everything that’s needed for this application.

Next, you need to download the Joomla archive and extract it in the app directory. To do so we’ll use the `system` method of the API, which allows us to execute some command on our server as if we were logged in with SSH. Since the default Static/CGI/PHP app comes with an `index.html` file which would shadow the `index.php` file provided by Joomla, we will also delete that file:

```python
>>> cmd = "rm -f index.html;"
>>> cmd += "wget https://wiki.webfaction.com/attachment/wiki/JoomlaFiles/Joomla_1.5.0-Beta-Full_Package.tar.gz?format=raw"
>>> cmd += "tar xzvf Joomla_1.5.0-Beta-Full_Package.tar.gz?format=raw"
>>> server.system(session_id, cmd)
''
```

Because we previously installed an application, the `system` method automatically runs in the new application’s directory. The `system` method returns whatever the command printed to standard output. If the command prints something to standard error, `system` raises an error with that text.

Next, create a MySQL database, since Joomla requires one:
WebFaction API Documentation

```python
>>> server.create_db(session_id, 'test5_joomla_db', 'mysql', 'db_password')
{'type': 'mysql',
'id': '1161011151605309510611111091080970951000980451',
'name': 'test5_joomla_db'}
```

This creates a MySQL database called `test5_joomla_db` and a user of the same name, with the password `db_password`.

Next, copy the Joomla configuration file from `configuration.php-dist` to `configuration.php` and then edit it to specify the database connection settings:

```python
>>> server.system(session_id, "cp configuration.php-dist configuration.php;")
```

```python
>>> server.replace_in_file(session_id, 'configuration.php',
    ("var $user = '';", "var $user = 'test5_joomla_db';"),
    ("var $password = '';", "var $password = 'db_password';"),
    ("var $db = '';", "var $db = 'test5_joomla_db';"))
```

The handy `replace_in_file` method lets us find and replace in a file. It takes a session ID, the name of the file, and any number of tuples containing a string to replace and the replacement string.

There are a few other steps needed to install a Joomla application: we need to edit the Joomla SQL file and run it. We won’t detail them here— you can look at the actual script for details.

## 2.4 Packaging the Install Script for the Control Panel

Previously, we installed a Joomla application by manually running a bunch of commands. We can package these commands in an install script for the control panel to run for us. The advantage is that we can then run this install script over and over again directly from the control panel or share it with others.

### 2.4.1 How the Install Script is Run

To use an install script in the control panel:

1. Log in to the control panel.
2. Click **Domains / websites → Applications**. The list of applications appears.
3. Click the **Add new application** button. The **Create a new application** form appears.
4. In the **Name** field, enter a name for the application.
5. In the **App Category** menu, click to select **Custom**. The **Script URL** field appears.
6. In the **Script URL** field, enter the install script’s URL.
7. Click the **Fetch URL** button.
8. If applicable, in the **Machine** menu, select a web server.
9. Click the **Save** button.

When you click the **Create** button, the script is run with the following parameters:

```
install_script create|delete username password app_name autostart extra_info
```

- `username` – control panel username
• `password` – user’s hashed password
• `app_name` – application name from Name field.
• `autostart` (boolean) – whether the user selected the Autostart checkbox
• `extra_info` – contents of the Extra info field.

When the user creates the app, the control panel will call the script with `create` as the first parameter. If the user deletes the app later on, the control panel will call the script with `delete` as the first parameter.

When it calls the script with `create`, the control panel expects the script to print the application ID to standard output and nothing else. If the script prints anything else to standard output, or prints anything to standard error, the control panel will display it as an error message on the add application page.

When it calls the script with `delete`, the control panel expects your script to not print anything to standard output or standard error. If anything gets printed the control panel will display it as an error message.

Additionally, if your script is written in Python and you include a docstring (PEP 257) at the beginning of the script, the control panel will use it as the application documentation.

To see two examples of install scripts, take a look at the Joomla install script or the MoinMoin install script.

### 2.4.2 Making a Script Available in One Click

Now that you have your install script, one way to share it would be to ask others copy and paste it in the control panel. That’s not convenient but fortunately, there is a better way.

On the add application page in the control panel, there is a field *Install script url* that appears when you select *Custom install script*. If you enter a URL and click *Fetch URL*, the control panel will look for an install script at that URL.

For the control panel to find the script on the page, your script must be enclosed between these two magic tags:

```
-----BEGIN WEBFACTION INSTALL SCRIPT-----
-----END WEBFACTION INSTALL SCRIPT-----
```

If you make your script available on the web, all you have to do to let anyone use it is give them a special URL in the form `https://my.webfaction.com/app/new-application?script_url=location` where `location` is the escaped URL to your script.

You can see examples at [InstallScripts](https://my.webfaction.com/app/new-application?script_url=location), where Joomla, MoinMoin, and other applications can be installed using that approach.

### 2.5 Additional Resources

For more information about the WebFaction API, please consult the [API Reference](https://my.webfaction.com/app/new-application?script_url=location).
The WebFaction XML-RPC API provides methods to handle many account tasks. This documentation is a complete reference to all of the possible API methods.

Please note that XML-RPC parameters are positional (order matters), and many parameters are required. Parameters may only be omitted if omitted parameters have default values and follow all other parameters to which you have supplied a value.

### 3.1 General

#### login

**Parameters**

- **username** (string) – a valid WebFaction control panel username
- **password** (string) – a valid WebFaction control panel user’s password
- **machine** (string) – the case-sensitive machine name (for example, `Web55`); optional for accounts with only one machine

Log in a user and return credentials required to make requests for that user. The method returns a session ID string and a struct containing following key-value pairs:

- **id** account ID
- **username** username
- **home** home directory path
- **web_server** Web server associated with the logged in account (for example, `Web55`)
- **mail_server** mail server associated with the logged in account (for example, `Mailbox2`)

**Note:** The session ID is required for all subsequent API calls.

#### list_disk_usage

**Parameters**

- **session_id** – session ID returned by `login`

List disk space usage statistics about your account (similar to usage statistics shown on the control panel). The method returns a struct containing the following members:

- **home_directories** A list of structs with details for each home directory associated with the account. Each struct contains the following members:
  - **last_reading** The date and time of the last recording of the home directory’s size
WebFaction API Documentation

machine The server name (for example, Web300)
name The username
size The disk usage in kilobytes

mailboxes A list of structs with details for each mailbox associated with the account. Each struct contains the following members:
last_reading The date and time of the last recording of the mailbox’s size
name The mailbox name
size The disk usage in kilobytes

mysql_databases A list of structs with details for each MySQL database associated with the account. Each struct contains the following members:
last_reading The date and time of the last recording of the database’s size
name The database name
size The disk usage in kilobytes

day_databases A list of structs with details for each PostgreSQL database associated with the account. Each struct contains the following members:
last_reading The date and time of the last recording of the database’s size
name The database name
size The disk usage in kilobytes

total The account’s total disk usage in kilobytes
quota The account’s total disk allotment in kilobytes
percentage The account’s total disk usage as a percentage of the quota (for example, an account using 3.1 GB of 100 GB would use 3.1 percent of its quota)

3.2 Email

3.2.1 Mailboxes

change_mailbox_password

Parameters

- session_id – session ID returned by login
- mailbox (string) – a valid mailbox name
- password (string) – the new mailbox password

Change a mailbox password.

See also:

See Strengthening Passwords for important information about choosing passwords.

create_mailbox

Parameters
• session_id – session ID returned by login
• mailbox (string) – mailbox name
• enable_spam_protection (boolean) – whether spam protection is enabled for the mailbox (optional, default: true)
• discard_spam (boolean) – whether spam messages received by the new mailbox are discarded (optional, default: false)
• spam_redirect_folder (string) – name of the IMAP folder where messages identified as spam are stored (optional, default: an empty string)
• use_manual_procmailrc (boolean) – whether to use manual procmailrc rules as specified by the manual_procmailrc parameter (optional, default: false)
• manual_procmailrc (string) – the procmailrc rules for the mailbox (optional, default: an empty string)

Warning: If discard_spam is true, messages misidentified as spam—false positives—may be lost permanently.

Create a mailbox and return a struct containing the following key-value pairs:
- id mailbox ID
- name mailbox name
- enable_spam_protection name of the folder where messages identified as spam are stored
- password a randomly generated password
- discard_spam a boolean indicating whether spam emails are be discarded
- spam_redirect_folder name of the IMAP folder where messages identified as spam are stored
- use_manual_procmailrc a boolean indicating whether manual procmailrc rules are enabled
- manual_procmailrc a string containing manual procmailrc rules

See also:
- update_mailbox

delete_mailbox

Parameters
- session_id – session ID returned by login
- mailbox (string) – mailbox name

Delete a mailbox.

list_mailboxes

Parameters session_id – session ID returned by login

Get information about the account’s mailboxes. The method returns an array of structs with the following key-value pairs:
- id mailbox ID
- name mailbox name
- enable_spam_protection name of the folder where messages identified as spam are stored
password    a randomly generated password

discard_spam    a boolean indicating whether spam emails are be discarded

spam_redirect_folder    name of the IMAP folder where messages identified as spam are stored

use_manual_procmailrc    a boolean indicating whether manual procmailrc rules are enabled

manual_procmailrc    a string containing manual procmailrc rules

update_mailbox

Parameters

• session_id – session ID returned by login

• mailbox (string) – mailbox name

• enable_spam_protection (boolean) – whether spam protection is enabled for the mailbox (optional, default: true)

• discard_spam (boolean) – whether spam messages received by the new mailbox are discarded (optional, default: false)

• spam_redirect_folder (string) – name of the IMAP folder where messages identified as spam are stored (optional, default: an empty string)

• use_manual_procmailrc (boolean) – whether to use manual procmailrc rules as specified by the manual_procmailrc parameter (optional, default: false)

• manual_procmailrc (string) – the procmailrc rules for the mailbox (optional, default: an empty string)

Warning: If discard_spam is true, messages misidentified as spam—false positives—may be lost permanently.

Change the details of an existing mailbox. The mailbox must exist before calling the method. The method returns a struct containing the following key-value pairs:

id    mailbox ID

name    mailbox name

enable_spam_protection    name of the folder where messages identified as spam are stored

password    a randomly generated password

discard_spam    a boolean indicating whether spam emails are be discarded

spam_redirect_folder    name of the IMAP folder where messages identified as spam are stored

use_manual_procmailrc    a boolean indicating whether manual procmailrc rules are enabled

manual_procmailrc    a string containing manual procmailrc rules

See also:

create_mailbox

3.2.2 Addresses

create_email

Parameters
Create an email address which delivers to the specified mailboxes. If autoresponder_on is true, then an autoresponder subject, message, and from address may be specified.

See also: 
update_email

Delete an email address.

list_emails

Parameters session_id – session ID returned by login

email_address (string) – an email address (for example, name@example.com)

targets (string) – names of destination mailboxes or addresses, separated by commas

autoresponder_on (boolean) – whether an autoresponder is enabled for the address (optional, default: false)

autoresponder_subject (string) – subject line of the autoresponder message (optional, default: an empty string)

autoresponder_message (string) – body of the autoresponder message (optional, default: an empty string)

autoresponder_from (string) – originating address of the autoresponder message (optional, default: an empty string)

script_machine (string) – a machine name for specifying a path to a script (optional, default: an empty string)

script_path (string) – an absolute path to a script; see Sending Mail to a Script for details (optional, default: an empty string)

Get information about the account’s email addresses. The method returns an array of structs with the following key-value pairs:

id email ID

email_address email address

targets mailboxes or email addresses to which the address is set to deliver

autoresponder_on a boolean indicating whether an autoresponder is enabled for the address

autoresponder_subject the autoresponder subject line (if applicable)

autoresponder_message the autoresponder message body (if applicable)

autoresponder_from the autoresponder from address (if applicable)

update_email

Parameters

• session_id – session ID returned by login
• **email_address** *(string)* – an email address (for example, `name@example.com`)
• **targets** *(array)* – names of destination mailboxes or addresses
• **autoresponder_on** *(boolean)* – whether an autoresponder is enabled for the address (optional, default: false)
• **autoresponder_subject** *(string)* – subject line of the autoresponder message (optional, default: an empty string)
• **autoresponder_message** *(string)* – body of the autoresponder message (optional, default: an empty string)
• **autoresponder_from** *(string)* – originating address of the autoresponder message (optional, default: an empty string)
• **script_machine** *(string)* – a machine name for specifying a path to a script (optional, default: an empty string)
• **script_path** *(string)* – an absolute path to a script; see *Sending Mail to a Script* for details (optional, default: an empty string)

Change the details of an existing email address. The email address must exist before calling the method. The method returns a struct with the following key-value pairs:

- **id** _email ID_
- **email_address** _email address_
- **targets** _mailboxes or email addresses to which the address is set to deliver_

See also: *create_email*

### 3.3 Websites and Domains

**create_domain**

*Parameters*

- **session_id** – session ID returned by *login*
- **domain** *(string)* – a domain name in the form of `example.com`
- **subdomain** *(string)* – each additional parameter provided after _domain_: a subdomain name of _domain_

Create a domain entry. If _domain_ has already been created, you may supply additional parameters to add subdomains. For example, if `example.com` already exists, *create_domain* may be called with four parameters—a session ID, `example.com`, `www`, `private`—to create `www.example.com` and `private.example.com`.

**Example:** Create a domain entry for `widgetcompany.example` using Python:

```python
>>> import xmlrpclib
>>> server = xmlrpclib.ServerProxy('https://api.webfaction.com/')
>>> session_id, account = server.login('widgetsco', 'widgetsrock')
>>> server.create_domain(session_id, 'widgetcompany.example', 'www', 'design')
{'domain': 'widgetcompany.example',
```
`create_website`  

**Parameters**

- `session_id` – session ID returned by `login`
- `website_name` (*string*) – the name of the new website entry
- `ip` (*string*) – *IP address* of the server where the entry resides
- `https` (*boolean*) – whether the website entry should use a secure connection
- `subdomains` (*array*) – an array of strings of (sub)domains to be associated with the website entry
- `site_apps` (*array*) – each additional parameter provided after `subdomains`: an array containing a valid application name (a string) and a URL path (a string)

Create a new website entry. Applications may be added to the website entry with additional parameters supplied after `subdomains`. The additional parameters must be arrays containing two elements: a valid application name and a path (for example, `htdocs` and `/`).

**Example:** Create a website entry for `widgetcompany.example`’s new Django project over HTTPS using Python:

```python  
>>> import xmlrpclib  
>>> server = xmlrpclib.ServerProxy('https://api.webfaction.com/')  
>>> session_id, account = server.login('widgetsco', 'widgetsrock')  
>>> server.create_website(session_id,  
... 'widgets_on_the_web',  
... '174.133.82.194',  
... True,  
... ['widgetcompany.example', 'www.widgetcompany.example'],  
... ['django', '/'])  
{'https': True,  
'id': 67074,  
'ip': '174.133.82.194',  
'name': 'widgets_on_the_web',  
'site_apps': [['django', '/']],  
'subdomains': ['widgetcompany.example', 'www.widgetcompany.example']}  
```

`delete_domain`  

**Parameters**

- `session_id` – session ID returned by `login`
- `domain` (*string*) – name of the domain to be deleted or the parent domain of the subdomains to be deleted
- `subdomains` (*string*) – each additional parameter provided after `domain`: subdomains of `domain` to be deleted

Delete a domain record or subdomain records. Subdomains of a domain may be deleted by supplying additional parameters after `domain`. If any subdomains are provided, only subdomains are deleted and the parent domain remains.

`delete_website`  

**Parameters**
WebFaction API Documentation

- **session_id** – session ID returned by login
- **website_name** *(string)* – name of website to be deleted
- **ip** *(string)* – IP address of the server where the website resides
- **https** *(boolean)* – whether the website uses a secure connection (optional, default: false)

Delete a website entry.

**list_bandwidth_usage**

**Parameters** session_id – session ID returned by login

List bandwidth usage statistics for your websites (similar to usage statistics shown on the control panel). The method returns a struct containing two members:

**daily**: A struct containing members named for the dates for the past two weeks (for example, 2015-01-05, 2015-01-04, 2015-01-03 and so on). The value of each dated member is a struct containing members named for each domain associated with the account (for example, example.com, www.example.com, somedomain.example, www.somedomain.example and so on). The value of each domain name member is the bandwidth usage for that domain during that day in kilobytes.

**monthly**: A struct containing members named for the months for the past year (for example, 2015-01, 2014-12, 2014-11 and so on). The value of each month member is a struct containing members named for each domain associated with the account (for example, example.com, www.example.com, somedomain.example, www.somedomain.example and so on). The value of each domain name member is the bandwidth usage for that domain during that month in kilobytes.

Overall, the struct resembles this outline:

- **daily**
  - today
    * www.example.com: 1024
    * example.com: 512
    ...
  - yesterday
  ...
  - two weeks ago

- **monthly**
  - this month
    * www.example.com: 2048
    * example.com: 1024
    ...
  - last month
  ...
  - a year ago

**list_domains**

**Parameters** session_id – session ID returned by login
Get information about the account’s domains. The method returns an array of structs with the following key-value pairs:

- **id**: domain ID
- **domain**: domain (for example, `example.com`)
- **subdomains**: array of subdomains for the domain

### list_websites

**Parameters**

- **session_id**: session ID returned by [login](#)

Get information about the account’s websites. The method returns an array of structs with the following key-value pairs:

- **id**: website ID
- **name**: website name
- **ip**: website IP address
- **https**: whether the website is served over HTTPS
- **subdomains**: array of website’s subdomains
- **website_apps**: array of the website’s apps and their URL paths; each item in the array is a two-item array, containing an application name and URL path

### update_website

**Parameters**

- **session_id**: session ID returned by [login](#)
- **website_name**: (string) – the name of the website entry
- **ip**: (string) – IP address of the server where the entry resides
- **https**: (boolean) – whether the website entry should use a secure connection
- **subdomains**: (array) – an array of strings of (sub)domains to be associated with the website entry
- **site_apps**: (array) – each additional parameter provided after subdomains: an array containing a valid application name (a string) and a URL path (a string)

Update a website entry. Applications may be added to the website entry with additional parameters supplied after subdomains. The additional parameters must be arrays containing two elements: a valid application name and a path (for example, `htdocs` and `'/}`).

**Example:** Update a website entry for `widgetcompany.example`’s new Django project over HTTPS using Python:

```python
>>> import xmlrpclib
>>> server = xmlrpclib.ServerProxy('https://api.webfaction.com/')
>>> session_id, account = server.login('widgetsco', 'widgetsrock')
>>> server.update_website(session_id,
...     'widgets_on_the_web',
...     '174.133.82.195',
...     True,
...     ['widgetcompany.example', 'dev.widgetcompany.example'],
...     [('django', '/'), ('wordpress', '/blog')])
{'https': True,
 'id': 67074,
...}
```
3.4 Applications

**create_app**

Parameters

- **session_id** – session ID returned by `login`
- **name** *(string)* – name of the application
- **type** *(string)* – type of the application
- **autostart** *(boolean)* – whether the app should restart with an `autostart.cgi` script (optional, default: false)
- **extra_info** *(string)* – additional information required by the application; if `extra_info` is not required or used by the application, it is ignored (optional, default: an empty string)
- **open_port** *(boolean)* – for applications that listen on a port, whether the port should be open on shared and dedicated IP addresses (optional, default: false)

Create a new application.

**See also:**

For a complete list of application types, see *Application Types*.

**delete_app**

Parameters

- **session_id** – session ID returned by `login`
- **name** *(string)* – name of the application

Delete an application.

**list_apps**

Parameters **session_id** – session ID returned by `login`

Get information about the account’s applications. The method returns an array of structs with the following key-value pairs:

- **id** app ID
- **name** app name
- **type** app type
- **autostart** whether the app uses autostart
- **port** port number if the app listens on a port, otherwise is 0
- **open_port** for applications that listen on a port, whether the port is open on shared and dedicated IP addresses (True for open ports, False for closed ports, or for applications that do not listen to a port)
- **extra_info** extra info for the app if any
machine name of the machine where the app resides

list_app_types

Parameters

session_id – session ID returned by login

Get information about available app types. The method returns an array of structs with the following key-value pairs:

name an identifier for the application type (for use as the create_app method’s type parameter)

label a short description of the application type

description a longer description of the application type

autostart applicable or an empty string, indicating whether the application uses an autostart.cgi file

extra_info description of any additional information required by the application installer’s extra_info field

open_port a boolean value indicating whether the application may use an open port

See also:

create_app

3.5 Cron

create_cronjob

Parameters

• session_id – session ID returned by login

• line (string) – crontab line to be added

Create a new cron job.

See also:

For more information about the cron syntax, please see the Wikipedia entry on cron.

delete_cronjob

Parameters

• session_id – session ID returned by login

• line (string) – crontab line to be removed

Remove an existing cron job.

3.6 DNS

create_dns_override

Parameters

• session_id – session ID returned by login

• domain (string) – domain name to be overridden (for example, sub.example.com)

• a_ip (string) – A IP address (optional, default: an empty string)
WebFaction API Documentation

- **cname** *(string)* – CNAME record (optional, default: an empty string)
- **mx_name** *(string)* – Mail exchanger record host name (optional, default: an empty string)
- **mx_priority** *(string)* – Mail exchanger record priority (optional, default: an empty string)
- **spf_record** *(string)* – TXT record (optional, default: an empty string)
- **aaaa_ip** *(string)* – An IPv6 address (optional, default: an empty string)
- **srv_record** *(string)* – A service locator (optional, default: an empty string)

Create DNS records and return an array of the new records (as in the form of `list_dns_overrides`).

```typescript
delete_dns_override
```

**Parameters**

- **session_id** – session ID returned by `login`
- **domain** *(string)* – domain name to be overridden (for example, `sub.example.com`)
- **a_ip** *(string)* – A IP address (optional, default: an empty string)
- **cname** *(string)* – CNAME record (optional, default: an empty string)
- **mx_name** *(string)* – Mail exchanger record host name (optional, default: an empty string)
- **mx_priority** *(string)* – Mail exchanger record priority (optional, default: an empty string)
- **spf_record** *(string)* – TXT record (optional, default: an empty string)
- **aaaa_ip** *(string)* – An IPv6 address (optional, default: an empty string)
- **srv_record** *(string)* – A service locator (optional, default: an empty string)

Delete DNS records and return an array of the deleted records (as in the form of `list_dns_overrides`).

```typescript
list_dns_overrides
```

**Parameters** **session_id** – session ID returned by `login`

Get information about the account’s DNS overrides. The method returns an array of structs with the following key-value pairs:

- **id** domain ID
- **domain** domain name to be overridden (for example, `sub.example.com`)
- **a_ip** A IP address
- **aaaa_ip** AAAA IP address (for IPv6)
- **cname** CNAME record
- **mx_name** Mail exchanger record host name
- **mx_priority** Mail exchanger record priority
- **spf_record** TXT record
- **srv_record** Service record
3.7 Databases

**change_db_user_password**

Parameters
- **session_id** – session ID returned by `login`
- **username** *(string)* – a database user’s username
- **password** *(string)* – the new password
- **db_type** *(string)* – the database type, either `mysql` or `postgresql`

Change a database user’s password. The method returns a struct containing the following key-value pairs:
- **username** database username
- **machine** database machine name
- **db_type** database type (MySQL or PostgreSQL)
- **database** database name

See also:
See *Strengthening Passwords* for important information about choosing passwords.

**create_db**

Parameters
- **session_id** – session ID returned by `login`
- **name** *(string)* – database name
- **db_type** *(string)* – the database type, either `mysql` or `postgresql`
- **password** *(string)* – password for the default database user
- **db_user** *(string)* – an existing database user (optional, default: create a new user)

Create a database. Optionally, you may assign ownership of the database to an existing user. To assign ownership to an existing user, provide an empty string as the `password` parameter and the username as the `db_user` parameter.

**Note:** MySQL database names may not exceed 16 characters.

See also:
See *Strengthening Passwords* for important information about choosing passwords.

**create_db_user**

Parameters
- **session_id** – session ID returned by `login`
- **username** *(string)* – the new database user’s username
- **password** *(string)* – the new database user’s password
- **db_type** *(string)* – the database type, either `mysql` or `postgresql`

Create a database user. The method returns a struct with the following key-value pairs:
- **machine** machine name
WebFaction API Documentation

_username_ database username

_db_type_ database type (MySQL or PostgreSQL)

See also:

See *Strengthening Passwords* for important information about choosing passwords.

```delete_db```

Parameters

- **session_id** – session ID returned by `login`
- **name** (*string*) – database name
- **db_type** (*string*) – the database type, either `mysql` or `postgresql`

Delete a database.

```delete_db_user```

Parameters

- **session_id** – session ID returned by `login`
- **username** (*string*) – the database user’s username
- **db_type** (*string*) – the database type

Delete a database user. The method returns a struct with the following key-value pairs:

- **machine** machine name
- **username** database username
- **db_type** database type (MySQL or PostgreSQL)

```enable_addon```

Parameters

- **session_id** – session ID returned by `login`
- **database** (*string*) – a database name
- **db_type** (*string*) – the database type (always use `postgresql`)
- **addon** (*string*) – the addon to enable (`tsearch` or `postgis`)

Enable a database addon. The method returns a struct with the following key-value pairs:

- **machine** machine name
- **db_type** database type (always PostgreSQL)
- **addon** addon enabled
- **db_type** database type (MySQL or PostgreSQL)

**Note:** This method applies to PostgreSQL databases only.

```grant_db_permissions```

Parameters

- **session_id** – session ID returned by `login`
- **username** (*string*) – a database user’s username
• **database** *(string)* – a database name
• **db_type** *(string)* – the database type *(mysql or postgresql)*

Grant full database permissions to a user with respect to a database. The method returns a struct with the following key-value pairs:

- **machine** machine name
- **username** database username
- **db_type** database type (MySQL or PostgreSQL)
- **database** database name

```
list_dbs
```

**Parameters**
- **session_id** – session ID returned by `login`

Get information about the account’s databases. The method returns an array of structs with the following key-value pairs:

- **name** database name
- **db_type** database type (MySQL or PostgreSQL)
- **users** an array of arrays each containing the name of a user with access to the database and that user’s permissions to the database
- **machine** machine name
- **encoding** character encoding (such as UTF-8)
- **addons** installed PostgreSQL addons, such as PostGIS

```
list_db_users
```

**Parameters**
- **session_id** – session ID returned by `login`

Get information about the account’s database users. The method returns an array of structs with the following key-value pairs:

- **machine** machine name
- **username** database username
- **db_type** database type (MySQL or PostgreSQL)

```
make_user_owner_of_db
```

**Parameters**
- **session_id** – session ID returned by `login`
- **username** *(string)* – a database user’s username
- **database** *(string)* – a database name
- **db_type** *(string)* – the database type *(mysql or postgresql)*

Assign ownership of a database to a user. The method returns a struct with the following key-value pairs:

- **machine** machine name
- **username** database username
- **db_type** database type (MySQL or PostgreSQL)
- **database** database name
revoke_db_permissions

Parameters

- **session_id** – session ID returned by [login](#)
- **username** *(string)* – a database user’s username
- **database** *(string)* – a database name
- **db_type** *(string)* – the database type (*mysql* or *postgresql*)

Revoke a user’s permissions with respect to a database. The method returns a struct with the following key-value pairs:

- **machine** machine name
- **username** database username
- **db_type** database type (MySQL or PostgreSQL)
- **database** database name

### 3.8 Files

replace_in_file

Parameters

- **session_id** – session ID returned by [login](#)
- **filename** *(string)* – path to file from the user’s home directory
- **changes** *(array)* – each additional parameter provided after *filename* : an array of two strings, where the first is the text to be replaced and the second is the replacement text

Find and replace strings in a file in the user’s home directory tree.

Any parameters after *filename* must be arrays containing a pair of strings, where the first is the string to be replaced and the second is the replacement text.

**Example:** Find all appearances of the word “eggs” in a file in the user’s home directory and replace them with the word “spam” using Python:

```
$ cat myfile.txt
eggs, spam, spam, and spam.
spam, spam, spam, and eggs.

>>> import xmlrpclib
>>> server = xmlrpclib.ServerProxy('https://api.webfaction.com/')
>>> session_id, account = server.login('widgetsco', 'widgetsrock')
>>> server.replace_in_file(session_id, 'myfile.txt', ('eggs', 'spam'))
''
>>> exit()

$ cat myfile.txt
spam, spam, spam, and spam.
spam, spam, spam, and spam.
```

write_file

**write_file**
Parameters

- **session_id** – session ID returned by login
- **filename** (*string*) – path to file to be written from the user’s home directory
- **str** (*string*) – string to be written
- **mode** (*string*) – write mode (optional, default: wb)

Write a string to a file in the user’s home directory tree.

**Note:** Commonly, the write mode is w for plain text and wb for binaries. a and ab can be used to append to files.

See also:
For more information about write modes, please see open().

### 3.9 Shell Users

#### change_user_password

Parameters

- **session_id** – session ID returned by login
- **username** (*string*) – username
- **password** (*string*) – a new password

Change a shell user’s password.

See also:
See Strengthening Passwords for important information about choosing passwords.

#### create_user

Parameters

- **session_id** – session ID returned by login
- **username** (*string*) – username
- **shell** (*string*) – the user’s command line interpreter; one of none, bash, sh, ksh, csh, or tcsh.
- **groups** (*array*) – extra permission groups of which the new user is to be a member

Create a new shell user. If shell is none, the user has FTP access only. All users are automatically a member of their own group; do not include the user’s own group in groups. Use an empty array to specify no extra groups.

#### delete_user

Parameters

- **session_id** – session ID returned by login
- **username** (*string*) – username

Delete a user.

#### list_users

### 3.9 Shell Users
Parameters **session_id** – session ID returned by **login**

Get information about the account’s shell users. The method returns an array of structs with the following key-value pairs:

- **username** username
- **machine** the user’s server (for example, **Web100**)
- **shell** the user’s configured command line interpreter (for example, **bash** or **tcsh**)
- **groups** extra permissions groups of which the user is a member

### 3.10 Servers

**list_ips**

Parameters **session_id** – session ID returned by **login**

Get information about all of the account’s machines and their IP addresses. This method returns an array of structs with the following key-value pairs:

- **machine** machine name (for example, **Web100**)
- **ip** IP address
- **is_main** a boolean value indicating whether the IP address is the primary address for the server (true) or an extra IP address provisioned to the account (false)

**list_machines**

Parameters **session_id** – session ID returned by **login**

Get information about the account’s machines. This method returns an array of structs with the following key-value pairs:

- **name** machine name (for example, **Web100**)
- **operating_system** the machine’s operating system (for example, **Centos6-64bit**)
- **location** the machine’s location (for example, **USA**)

### 3.11 Miscellaneous

**run_php_script**

Parameters

- **session_id** – session ID returned by **login**
- **script** (string) – an absolute path to script (or path to the script starting from the user’s home directory)
- **code_before** (string) – PHP code to be executed before **script**

Run PHP code followed by a PHP script. The PHP code and script is run as the user.

**set_apache_acl**

Parameters
• **session_id** – session ID returned by [login](#)

• **paths** (*string or array of strings*) – path from home directory

• **permission** (*string*) – `r`, `w`, or `x` or a combination thereof (optional, default: `rwx`)

• **recursive** (*boolean*) – whether Apache’s access is granted recursively (optional, default: false)

Grant the machine-wide Apache instance access to files or directories.

**system**

**Parameters**

• **session_id** – session ID returned by [login](#)

• **cmd** (*string*) – a shell command to be executed

Execute a command as the user, as if through SSH. If an application was installed previously in the session, the command will be run from the directory where that application was installed.

**Note:** If `cmd` writes to standard error, the API method will return an XML-RPC fault.
# APPLICATION TYPES

The following application types may be used with API methods such as `create_app`. Each entry contains the application type's unique name paired with it's descriptive label.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>awstats74</td>
<td>AWStats (7.4)</td>
</tr>
<tr>
<td>cherrypy380_27</td>
<td>CherryPy 3.8.0 (Python 2.7)</td>
</tr>
<tr>
<td>cherrypy380_34</td>
<td>CherryPy 3.8.0 (Python 3.4)</td>
</tr>
<tr>
<td>custom_app_with_port</td>
<td>Custom app (listening on port)</td>
</tr>
<tr>
<td>custom_install_script</td>
<td>Custom install script</td>
</tr>
<tr>
<td>custom_websockets_app_with_port</td>
<td>Custom websockets app (listening on port)</td>
</tr>
<tr>
<td>django1422_mw4421_27</td>
<td>Django 1.4.22 (mod_wsgi 4.4.21/Python 2.7)</td>
</tr>
<tr>
<td>django1711_mw4421_27</td>
<td>Django 1.7.11 (mod_wsgi 4.4.21/Python 2.7)</td>
</tr>
<tr>
<td>django1711_mw4421_34</td>
<td>Django 1.7.11 (mod_wsgi 4.4.21/Python 3.4)</td>
</tr>
<tr>
<td>django187_mw4421_34</td>
<td>Django 1.8.7 (mod_wsgi 4.4.21/Python 3.4)</td>
</tr>
<tr>
<td>django188_mw4421_27</td>
<td>Django 1.8.8 (mod_wsgi 4.4.21/Python 2.7)</td>
</tr>
<tr>
<td>django188_mw4421_34</td>
<td>Django 1.8.8 (mod_wsgi 4.4.21/Python 3.4)</td>
</tr>
<tr>
<td>django191_mw4421_27</td>
<td>Django 1.9.1 (mod_wsgi 4.4.21/Python 2.7)</td>
</tr>
<tr>
<td>django191_mw4421_34</td>
<td>Django 1.9.1 (mod_wsgi 4.4.21/Python 3.4)</td>
</tr>
<tr>
<td>django191_mw4421_35</td>
<td>Django 1.9.1 (mod_wsgi 4.4.21/Python 3.5)</td>
</tr>
<tr>
<td>drupal_6_37</td>
<td>Drupal (6.37)</td>
</tr>
<tr>
<td>drupal_7_41</td>
<td>Drupal (7.41)</td>
</tr>
<tr>
<td>drupal_8_0</td>
<td>Drupal (8.0.0)</td>
</tr>
<tr>
<td>ghost-0.7.1</td>
<td>Ghost 0.7.1</td>
</tr>
<tr>
<td>git_230</td>
<td>Git 2.3.0</td>
</tr>
<tr>
<td>joomla_346</td>
<td>Joomla (3.4.6)</td>
</tr>
<tr>
<td>mod_wsgi4421-python27</td>
<td>mod_wsgi 4.4.21/Python 2.7</td>
</tr>
<tr>
<td>Application Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td><strong>mod_wsgi4421-python34</strong></td>
<td>mod_wsgi 4.4.21/Python 3.4</td>
</tr>
<tr>
<td><strong>mod_wsgi4421-python35</strong></td>
<td>mod_wsgi 4.4.21/Python 3.5</td>
</tr>
<tr>
<td><strong>mysql</strong></td>
<td>MySQL private instance</td>
</tr>
<tr>
<td><strong>node-0.10.31</strong></td>
<td>Node.js 0.10.31</td>
</tr>
<tr>
<td><strong>node-0.12.7</strong></td>
<td>Node.js 0.12.7</td>
</tr>
<tr>
<td><strong>node-4.2.2</strong></td>
<td>Node.js 4.2.2</td>
</tr>
<tr>
<td><strong>passenger-4.0.58</strong></td>
<td>Passenger 4.0.58 (nginx 1.6.2/Ruby 2.1)</td>
</tr>
<tr>
<td><strong>passenger-5.0.21</strong></td>
<td>Passenger 5.0.21 (nginx 1.8.0/Ruby 2.2)</td>
</tr>
<tr>
<td><strong>postgresql</strong></td>
<td>PostgreSQL private instance</td>
</tr>
<tr>
<td><strong>pyramid_15_27</strong></td>
<td>Pyramid 1.5/Python 2.7</td>
</tr>
<tr>
<td><strong>rails-4.1.12</strong></td>
<td>Rails 4.1.12 (nginx 1.6.2/Passenger 4.0.58/Ruby 2.1.2)</td>
</tr>
<tr>
<td><strong>rails-4.2.4</strong></td>
<td>Rails 4.2.4 (Passenger 5.0.21/Ruby 2.2)</td>
</tr>
<tr>
<td><strong>redmine-2.6.7</strong></td>
<td>Redmine 2.6.7</td>
</tr>
<tr>
<td><strong>redmine-3.0.5</strong></td>
<td>Redmine 3.0.5</td>
</tr>
<tr>
<td><strong>redmine-3.1.1</strong></td>
<td>Redmine 3.1.1</td>
</tr>
<tr>
<td><strong>static_only</strong></td>
<td>Static only (no .htaccess)</td>
</tr>
<tr>
<td><strong>static_php54</strong></td>
<td>Static/CGI/PHP-5.4</td>
</tr>
<tr>
<td><strong>static_php55</strong></td>
<td>Static/CGI/PHP-5.5</td>
</tr>
<tr>
<td><strong>static_php56</strong></td>
<td>Static/CGI/PHP-5.6</td>
</tr>
<tr>
<td><strong>static_php70</strong></td>
<td>Static/CGI/PHP-7.0</td>
</tr>
<tr>
<td><strong>subversion</strong></td>
<td>Subversion</td>
</tr>
<tr>
<td><strong>symlink53</strong></td>
<td>Symbolic link to static/cgi/php53 app</td>
</tr>
<tr>
<td><strong>symlink54</strong></td>
<td>Symbolic link to static/cgi/php54 app</td>
</tr>
<tr>
<td><strong>symlink55</strong></td>
<td>Symbolic link to static/cgi/php55 app</td>
</tr>
<tr>
<td><strong>symlink56</strong></td>
<td>Symbolic link to static/cgi/php56 app</td>
</tr>
<tr>
<td><strong>symlink70</strong></td>
<td>Symbolic link to static/cgi/php70 app</td>
</tr>
<tr>
<td><strong>symlink_static_only</strong></td>
<td>Symbolic link to static-only app</td>
</tr>
<tr>
<td><strong>trac_0127</strong></td>
<td>Trac (0.12.7) - Subversion</td>
</tr>
<tr>
<td><strong>trac_0127_git</strong></td>
<td>Trac (0.12.7) - Git</td>
</tr>
<tr>
<td><strong>trac_109_git</strong></td>
<td>Trac (1.0.9) - Git</td>
</tr>
<tr>
<td><strong>trac_109_svn</strong></td>
<td>Trac (1.0.9) - Subversion</td>
</tr>
<tr>
<td><strong>turbogears_234_27</strong></td>
<td>TurboGears (2.3.4)/Python (2.7)</td>
</tr>
</tbody>
</table>
turbo gears 2.3.4 TurboGears (2.3.4)/Python (3.4)

web d a v WebDav

web d a v s y m l i n k WebDav Symlink

web s t a t  Webalizer

w o r d p r e s s 4 3 2 WordPress 4.3.2

w o r d p r e s s 4 4 1 WordPress 4.4.1

• genindex
INDEX

C
change_db_user_password, 21
change_mailbox_password, 10
change_user_password, 25
create_app, 18
create_cronjob, 19
create_db, 21
create_db_user, 21
create_dns_override, 19
create_domain, 14
create_email, 12
create_mailbox, 10
create_user, 25
create_website, 15

D
delete_app, 18
delete_cronjob, 19
delete_db, 22
delete_db_user, 22
delete_dns_override, 20
delete_domain, 15
delete_email, 13
delete_mailbox, 11
delete_user, 25
delete_website, 15

E
enable_addon, 22

G
grant_db_permissions, 22

L
list_app_types, 19
list_apps, 18
list_bandwidth_usage, 16
list_db_users, 23
list_dbs, 23
list_disk_usage, 9
list_dns_overrides, 20
list_domains, 16
list_emails, 13
list_ips, 26
list_machines, 26
list_mailboxes, 11
list_users, 25
list_websites, 17
login, 9

M
make_user_owner_of_db, 23

P
Packaging, 7
Python Enhancement Proposals
PEP 257, 8

R
replace_in_file, 24
revoke_db_permissions, 23
run_php_script, 26

S
set_apache_acl, 26
system, 27

U
update_email, 13
update_mailbox, 12
update_website, 17

W
write_file, 24