WebFaction API Documentation

Paragon Internet Group Limited - WebFaction is a service of Parag

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

The WebFaction API (Application Programming Interface) is an 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 help you make such requests.

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

```
>>> import xmlrpclib
>>> server = xmlrpclib.ServerProxy('https://api.webfaction.com/')
>>> session_id, account = server.login('widgetsco', 'widgetsrock', 'Web500', 2)
```

Or with Ruby's `xmlrpc` package:

```
>> 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", "Web55", 2)
["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.

For a more in-depth tour of the API, see *Tutorial*. For a complete reference of calls, parameters, and return values, see *API Reference*.

CHAPTER

TWO

TUTORIAL

The WebFaction API is an XML-RPC API available at `https://api.webfaction.com/`. The API lets you automate many tasks that you would ordinarily do with the control panel or an SSH session, like modifying an email address, creating a file, or installing an application. You can also use the WebFaction API to create scripts that install applications, which you can share with other WebFaction users.

In this tutorial, you'll learn how to

- log in to the API
- complete a few common control panel tasks with the API
- use your new API skills to create a sharable application install script

2.1 Before you begin

In this tutorial, we'll demonstrate some ways to use the API using the Python programming language and the Python standard library's xmlrpclib module. You can use other languages to work with the API and to create install scripts, but to follow along with this tutorial you'll need to use the interactive Python interpreter.

On most systems you can run `python2.7` in a terminal session to start an interactive Python interpreter (including on your WebFaction server in an *SSH session*). You'll see a prompt that looks something like this:

```
$ python2.7
Python 2.7.12 (default, Oct 11 2016, 05:24:00)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Next, create create an `xmlrpclib.ServerProxy` instance called `server` that we can use to communicate with the API throughout the rest of the tutorial:

```
>>> import xmlrpclib
>>> server = xmlrpclib.ServerProxy('https://api.webfaction.com/')
```

2.2 Logging in

Before we can make any useful requests to the API, we need to log in to the API to receive a *session ID*. The *session ID* is a secret string that you provide to all subsequent API calls, which associates each of your requests to your WebFaction account.

To log in to the API, we'll need to call the `login` method with four arguments:

- your account name (that you use to log in to the WebFaction control panel)
- your password (that you use to log in to the WebFaction control panel)
- your *server name*, capitalized (for example, `Web999`)
- the API version (typically the latest version, currently version `2`)

Note: To preserve backwards compatibility, the WebFaction API is *versioned*. Which version number you pass to `login` will determine what arguments API methods will accept and what values they will return. For new projects or projects that don't need to worry about backwards compatibility, choosing the latest version is recommended.

For more information, see *API Versions*.

Continuing from the session we started in the previous section, here's how to log in with Python:

```
>>> session_id, account = server.login('test5', 'password', 'Web500', 2)
>>> session_id
'6zlwc5j1ne8kr'
>>> account
{'username': 'demo', 'web_server': 'Web500', 'version': 2, 'home': '/home',
'mail_server': 'Mailbox3', 'id': 1}
```

The XML-RPC API returns an array containing a string and a struct. Python represents the data as Python types. So the array becomes a Python list, the string becomes an ordinary Python string, and the struct becomes a Python dictionary. In this case, the first list element is the session ID (`'6zlwc5j1ne8kr'`). The second list element is a `dict` containing details about your account.

Now that we have the `session_id`, we can make some more interesting API calls.

2.3 Calling API methods

Once you have a session ID, you can call the other methods provided by the API. You can find a complete list of API methods in the *API Reference*.

Every API method except `login` requires a session ID. For example, the method `list_emails` can tell us what email addresses have been created for the account The method takes one argument: a session ID. Calling it looks like this:

```
>>> server.list_app_types(session_id)
[{'autoresponder_subject': '', 'autoresponder_message': '', 'targets': '',
'autoresponder_on': False, 'email_address': 'demo@example.com', 'id': 482804,
'autoresponder_from': ''}]
```

As another example, the method `change_mailbox_password` takes exactly three arguments:

1. a session ID
2. a mailbox name
3. a strong password

Some methods can accept several arguments where some of those arguments are optional. If a method has optional arguments, then you don't need to supply the optional arguments if those arguments appear *after* any arguments that you want to provide. For example, the method `create_email` can accept up to nine arguments, but several of those arguments are optional (for setting up autoresponders or scripts to process incoming mail). If you don't want to use autoresponders or scripts, then only the first three arguments are required:

```
>>> server.create_email(session_id, 'demo@example.com', 'demo')
{'autoresponder_subject': '', 'autoresponder_message': '', 'id': 482807,
'targets': 'demo', 'autoresponder_on': 0, 'script_path': '',
'email_address': 'demo@example.com', 'script_machine': '',
'autoresponder_from': ''}
```

But if we wanted to create an email address such that a script will receive each incoming message but not use an autoresponder, then we must provide all of the arguments to the `create_email` method, since the script arguments come *after* the autoresponder arguments:

```
>>> server.create_email(session_id, 'demo2@example.com', 'demo', False,
                        '', '', '', 'Web500', '/home/demo/mymailscript.py')
{'autoresponder_subject': '', 'autoresponder_message': '', 'id': 482808,
'targets': 'demo', 'autoresponder_on': 0,
'script_path': '/home/demo/mymailscript.py',
'email_address': 'demo2@example.com', 'script_machine': 'Web500',
'autoresponder_from': ''}
```

Now that you know how to call API methods, you can combine several methods to complete more complicated tasks.

2.4 Creating an application with the API

Suppose we wanted to start developing a site with a database and a PHP script. One way to do this would be to log in to the control panel, create a *Static/CGI/PHP* application, create a MySQL database user and database, and then open an SSH session to create an `index.php` file. But we can also do all of that with the API. Let's try it out.

First, let's create the *Static/CGI/PHP* application with the `create_app` method. The method takes three required arguments:

1. a session ID
2. an app name
3. an *application type string* (in this case, `static_php70`)

```
>>> server.create_app(session_id, 'development_app', 'static_php70')
{'name': 'development_app', 'port': 0, 'machine': 'Web500',
'autostart': False, 'open_port': False, 'type': 'static_php70',
'id': 1033807, 'extra_info': ''}
```

Next, let's create a MySQL database and user with the `create_db` method. In addition to a session ID, it requires three arguments: a database name, a database type (in our case, `mysql`), and a password for the default user (automatically named after the database).

```
>>> server.create_db(session_id, 'development', 'mysql', 'mysecret')
{'machine': 'Web500', 'db_type': 'mysql', 'db_user': 'development',
'name': 'development'}
```

Next, let's remove the `index.html` that's created in each new *Static/CGI/PHP* app. We can do that with the `system` method, which requires two arguments: a session ID and a shell command.

```
>>> server.system(session_id, "rm /home/demo/webapps/development_app/index.html")
''
```

Now we're free to start development by creating an `index.php` file. To do that, we can use the `write_file` method:

```
>>> server.write_file(session_id,
                      '/home/demo/webapps/development_app/index.php',
                      """<?php
    $appname = 'Development App!';
    echo "Welcome to " . $appname;
    ?>
    """)
''
```

Now that you've had an introduction to the API, you can explore the other *API methods* available and use the API to automate many parts of your account.

2.5 Creating an install script for the control panel

In the previous sections, we worked with the API by manually running a bunch of commands. While we could turn those commands into a script that we could run locally, a more flexible option is to create an install script for the control panel. In this section, you'll learn how to create a Python install script for the control panel that handles creating and deleting an application and that you can share with others.

2.5.1 Running an install script

To run a custom install script with the control panel:

1. *Log in to the WebFaction 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*.
6. In the *App type* menu, click to select *Custom install script*.
7. If applicable, in the *Machine* menu, click to select a web server.
8. In the *Script URL* field, enter the URL for the install script.
9. Click the *Fetch script* button.
10. Click the *Save* button.

When you click the control panel's *Save* button, the script is called as if you ran *install_script* from the command line with these arguments:

```
install_script create username password machine app_name autostart extra_info
```

where:

- *username* is the control panel username
- *password* is the user's hashed password
- *machine* is the machine name from the control panel *Machine* menu
- *app_name* is the application name from the control panel's *Name* field
- *autostart* is a boolean value for whether the user selected the control panel's *Autostart* checkbox

- *extra_info* is the contents of the control panel's *Extra info* field

Later, if you click the control panel's *Delete* button for the installed application, then the control panel will run *install_script* with the following arguments:

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

The control panel expects that all install scripts will be Python 2.7 scripts enclosed in these tags:

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

The control panel expects install scripts run with the `create` argument to have the following behavior:

- The script will call `create_app` (for example, to create a *Static/CGI/PHP* or *Custom app (listening on port)* application)
- If the script runs successfully, then the script prints only the application ID to standard output and prints nothing to standard error.
- If the script fails, then the script prints error messages to standard output or standard error.

The control panel expects install scripts run with the `delete` argument to have the following behavior:

- If the script runs successfully, then the script prints nothing to standard output or standard error.
- If the script fails, then the script prints error messages to standard output or standard error.

Whenever a script fails (when run with the `create` or `delete` arguments), whatever was printed to standard output or standard error are displayed as error messages in the control panel.

Additionally, if your script is written in Python and you include a docstring (**PEP 257**) at the beginning of the script, then the control panel will display the docstring as help text in the control panel.

2.5.2 Example install script

To see how an install script is typically put together, here's how the application we created with the API in the previous section would look as an install script:

```
-----BEGIN WEBFACTION INSTALL SCRIPT-----
#!/usr/bin/env python2.7

"""A sample WebFaction control panel install script."""

import random
import sys
import string
import xmlrpclib

server = xmlrpclib.ServerProxy('https://api.webfaction.com/')

def randompassword():
    """Generate a password."""
    chars = string.ascii_uppercase + string.ascii_lowercase + string.digits
    size = random.randint(16, 24)
    return ''.join(random.choice(chars) for x in xrange(size))


def create(session_id, app_name, home):
    """Create the application."""
```

```
    # prepare database details
    db_name = "{}_db".format(app_name)
    db_user = db_name
    db_password = randompassword()

    # create database
    server.create_db(session_id, db_name, 'mysql', db_password)

    # create static app
    app_info = server.create_app(session_id, app_name, 'static_php70')

    # remove existing index.html
    server.system(session_id,
                  "rm {}/webapps/{}/index.html".format(home, app_name))

    # create PHP file
    server.write_file(session_id,
                      '{}/webapps/{}/index.php'.format(home, app_name),
                      """<?php
$db_user = {db_user}
$db_pass = {db_password}
$appname = '{app_name}';

echo "Welcome to " . $appname;
?>
""".format(db_user=db_user, db_password=db_password, app_name=app_name))

    # when finished, always print app id
    print app_info['id']


def delete(session_id, app_name):
    """Delete the application and its database and database user."""
    db_name = "{}_db".format(app_name)
    db_user = db_name

    server.delete_app(session_id, app_name)
    server.delete_db(session_id, db_name, 'mysql')
    server.delete_db_user(session_id, db_user, 'mysql')


def main():
    username, password, machine, app_name, autostart, extra_info = sys.argv[2:]
    session_id, account_info = server.login(username, password, machine, 2)
    home = "{}/{}".format(account_info['home'], account_info['username'])

    if sys.argv[1] == 'create':
        create(session_id, app_name, home)
    else:
        delete(session_id, app_name)


if __name__ == '__main__':
    main()

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

2.5.3 Sharing your scripts

Once you've made a script, you can share it with others by making the file available at public URL. You can automatically open the installer script in the WebFaction control panel by using a special URL in the format of `https://my.webfaction.com/new-application?script_url=script` where *script* is the URL for for your installer script.

For example, you can use `the example script` from the previous section by opening https://my.webfaction.com/new-application?script_url=https://docs.webfaction.com/xmlrpc-api/_downloads/example.txt in a browser.

2.6 Additional resources

For more about the API, please see the *API Reference*. For help with the API, join us in the WebFaction Q&A community.

CHAPTER

THREE

API REFERENCE

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 that you supply a value for.

3.1 API Versions

The WebFaction API is versioned. You can choose which version of the API to use when you call the `login` method. Subsequent API calls using the session ID returned by `login` will use the same API version as the call to `login`.

For backwards compatibility, version `1` is the default version. Unless otherwise noted, methods, parameters, and return values are the same between versions.

Version `2` provides support for managing SSL/TLS certificates. See *Certificates*, `create_website`, `list_websites`, and `update_website` for details.

3.2 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
- **version** (*integer*) – the API version number (`1` or `2`); optional for version `1`

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. The session expires one hour after the last API call using the session ID. The session ID will also determine which *API version* is used for 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

`machine` The *server name*

`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

`postgresql_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)

3.3 Email

3.3.1 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`

3.3.2 Addresses

`create_email`

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)

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_email`

Parameters

- **session_id** – session ID returned by `login`
- **email_address** (*string*) – an email address (for example, `name@example.com`)

Delete an email address.

`list_emails`

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

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`

3.4 Websites, Domains, and Certificates

3.4.1 Certificates

`create_certificate`

Parameters

- **session_id** – session ID returned by `login`
- **name** (*string*) – a name for the certificate, such as the name of the website or domain you plan to use the certificate with
- **certificate** (*string*) – the text of your PEM-format certificate, including the `BEGIN` and `END` lines
- **private_key** (*string*) – the text of your PEM-format private key, including the `BEGIN` and `END` lines
- **intermediates** (*string*) – (optional) the text of a PEM-format intermediate certificate or bundle, including the `BEGIN` and `END` lines

---

Note: *API version 2* only.

Create an SSL/TLS certificate. To associate the certificate with an HTTPS website, see `create_website` or `update_website`.

`delete_certificate`

Parameters

- **session_id** – session ID returned by `login`
- **name** (*string*) – a certificate name

Note: *API version 2* only.

Delete an SSL/TLS certificate.

`list_certificates`

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

Note: *API version 2* only.

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

name **(string)** the name of the certificate record

id **(integer)** a unique number for the certificate

certificate **(string)** the text of your PEM-format certificate

private_key **(string)** the text of your PEM-format private key

intermediates **(string)** the text of a PEM-format intermediate certificate or bundle (or an empty string, if one wasn't provided)

expiry_date **(string)** the date the certificate expires in *YYYY-MM-DD* format

domains **(string)** a comma-separated list of domains that the certificate is issued for

`update_certificate`

Parameters

- **session_id** – session ID returned by `login`
- **name** (*string*) – a certificate name
- **certificate** (*string*) – the text of your PEM-format certificate, including the `BEGIN` and `END` lines
- **private_key** (*string*) – the text of your PEM-format private key, including the `BEGIN` and `END` lines
- **intermediates** (*string*) – (optional) the text of a PEM-format intermediate certificate or bundle, including the `BEGIN` and `END` lines

---

Note: *API version 2* only.

Update an existing SSL/TLS certificate.

3.4.2 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:

```
>>> 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',
 'id': 47255,
 'subdomains': ['www', 'design']}
```

`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.

`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

3.4.3 Websites

`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)

Note: *API version 2* uses the following 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

•**certificate** (*string*) - the name of a certificate or an empty string to specify no certificate

•**site_apps** (*array*) - each additional parameter provided after `certificate`: an array containing a valid application name (a string) and a URL path (a string)

For more about creating SSL/TLS certificates, see *Certificates*.

Create a new website entry. Applications may be added to the website entry with additional parameters supplied after `subdomains` (*API version 1*) or `certificate` (*API version 2*). 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:

```
>>> 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_website`

Parameters

- **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 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_websites`

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

Get information about the account's websites.

API version 1 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

API version 2 returns an additional key-value pair:

certificate the name of the certificate for the website

`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)

Note: *API version 2* uses the following 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
- **certificate** (*string*) - the name of a certificate or an empty string to specify no certificate
- **site_apps** (*array*) - each additional parameter provided after `certificate`: an array containing a valid application name (a string) and a URL path (a string)

For more about creating SSL/TLS certificates, see *Certificates*.

Update a website entry. For each parameter after after `subdomains` (*API version 1*) or `certificate` (*API version 2*), you may supply an array of two elements:

1. a valid application name (such as `'htdocs'`) and
2. a URL path for the application (such as `'/'` or `'/blog'`)

Together, the `site_apps` parameters replace the website's existing apps. For example, if a website has three apps and `update_website` is called with two `site_apps` array parameters, the original three apps are removed and the two apps are added.

Example: Update a website entry for `widgetcompany.example`'s new Django project over HTTPS using 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,
 'ip': '174.133.82.195',
 'name': 'widgets_on_the_web',
 'site_apps': [['django', '/'], ['wordpress', '/blog']],
 'subdomains': ['widgetcompany.example', 'dev.widgetcompany.example']}
```

See also:

`create_website`

3.5 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`

3.6 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.

3.7 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)
- **cname** (*string*) – CNAME record (optional, default: an empty string)
- **mx_name** (*string*) – Mail exchanger record hostname (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`).

`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 hostname (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`).

`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 hostname

mx_priority Mail exchanger record priority

spf_record TXT record

srv_record Service record

3.8 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

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** (strings `mysql` or `postgresql`) – 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

3.9 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 users'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.
```

`replace_in_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().

3.10 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`

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

3.11 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)

id numeric identifier (but not the IP address itself)

`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`)

id numeric machine identifier

3.12 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.

CHAPTER

FOUR

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.

awstats-7.6 AWStats (7.6)

cherrypy-13.1.0_python-2.7 CherryPy 13.1.0 (Python 2.7)

cherrypy-13.1.0_python-3.5 CherryPy 13.1.0 (Python 3.5)

custom_app_with_port Custom app (listening on port)

custom_install_script Custom install script

custom_websockets_app_with_port Custom websockets app (listening on port)

django-1.10.8_mod_wsgi-4.5.18_python-2.7 Django 1.10.8 (mod_wsgi 4.5.18/Python 2.7)

django-1.10.8_mod_wsgi-4.5.18_python-3.5 Django 1.10.8 (mod_wsgi 4.5.18/Python 3.5)

django-1.11.7_mod_wsgi-4.5.20_python-2.7 Django 1.11.7 (mod_wsgi 4.5.20/Python 2.7)

django-1.11.7_mod_wsgi-4.5.20_python-3.5 Django 1.11.7 (mod_wsgi 4.5.20/Python 3.5)

django-1.11.7_mod_wsgi-4.5.20_python-3.6 Django 1.11.7 (mod_wsgi 4.5.20/Python 3.6)

django-1.8.18_mod_wsgi-4.5.15_python-2.7 Django 1.8.18 (mod_wsgi 4.5.15/Python 2.7)

django-1.8.18_mod_wsgi-4.5.15_python-3.5 Django 1.8.18 (mod_wsgi 4.5.15/Python 3.5)

django-1.9.13_mod_wsgi-4.5.15_python-2.7 Django 1.9.13 (mod_wsgi 4.5.15/Python 2.7)

django-1.9.13_mod_wsgi-4.5.15_python-3.5 Django 1.9.13 (mod_wsgi 4.5.15/Python 3.5)

django-2.0_mod_wsgi-4.5.20_python-3.5 Django 2.0 (mod_wsgi 4.5.20/Python 3.5)

django-2.0_mod_wsgi-4.5.20_python-3.6 Django 2.0 (mod_wsgi 4.5.20/Python 3.6)

drupal-7.56 Drupal (7.56)

drupal-8.4.3 Drupal (8.4.3)

ghost-1.19.0 Ghost 1.19.0

git Git

joomla-3.8.3 Joomla (3.8.3)

mod_wsgi-4.5.24_python-2.7 mod_wsgi 4.5.24/Python 2.7

mod_wsgi-4.5.24_python-3.5 mod_wsgi 4.5.24/Python 3.5

mod_wsgi-4.5.24_python-3.6 mod_wsgi 4.5.24/Python 3.6

mysql MySQL private instance

node-0.10.48 Node.js 0.10.48

node-0.12.18 Node.js 0.12.18

node-4.8.7 Node.js 4.8.7

node-6.12.2 Node.js 6.12.2

node-8.9.1 Node.js 8.9.1

node-8.9.3 Node.js 8.9.3

passenger-5.1.12_ruby-2.2 Passenger 5.1.12 (nginx 1.12.1/Ruby 2.2)

passenger-5.1.12_ruby-2.3 Passenger 5.1.12 (nginx 1.12.1/Ruby 2.3)

passenger-5.1.12_ruby-2.4 Passenger 5.1.12 (nginx 1.12.1/Ruby 2.4)

postgresql PostgreSQL private instance

pyramid-1.9.1_python-2.7 Pyramid 1.9.1/Python 2.7

rails-4.2.10_passenger-5.1.10_ruby-2.2 Rails 4.2.10 (Passenger 5.1.10/Ruby 2.2)

rails-5.1.4_passenger-5.1.10_ruby-2.2 Rails 5.1.4 (Passenger 5.1.10/Ruby 2.2)

redmine-3.4.3_passenger-5.1.10_ruby-2.4 Redmine 3.4.3 (Passenger 5.1.10/Ruby 2.4)

static_only Static only (no .htaccess)

static_php54 Static/CGI/PHP-5.4

static_php55 Static/CGI/PHP-5.5

static_php56 Static/CGI/PHP-5.6

static_php70 Static/CGI/PHP-7.0

static_php71 Static/CGI/PHP-7.1

subversion Subversion

symlink53 Symbolic link to static/cgi/php53 app

symlink54 Symbolic link to static/cgi/php54 app

symlink55 Symbolic link to static/cgi/php55 app

symlink56 Symbolic link to static/cgi/php56 app

symlink70 Symbolic link to static/cgi/php70 app

symlink71 Symbolic link to static/cgi/php71 app

symlink_static_only Symbolic link to static-only app

trac-1.2.2_git Trac (1.2.2) - Git

trac-1.2.2_svn Trac (1.2.2) - Subversion

webdav WebDav

webdav_symlink WebDav Symlink

webstat Webalizer

wordpress-4.9.1 WordPress 4.9.1

- *genindex*