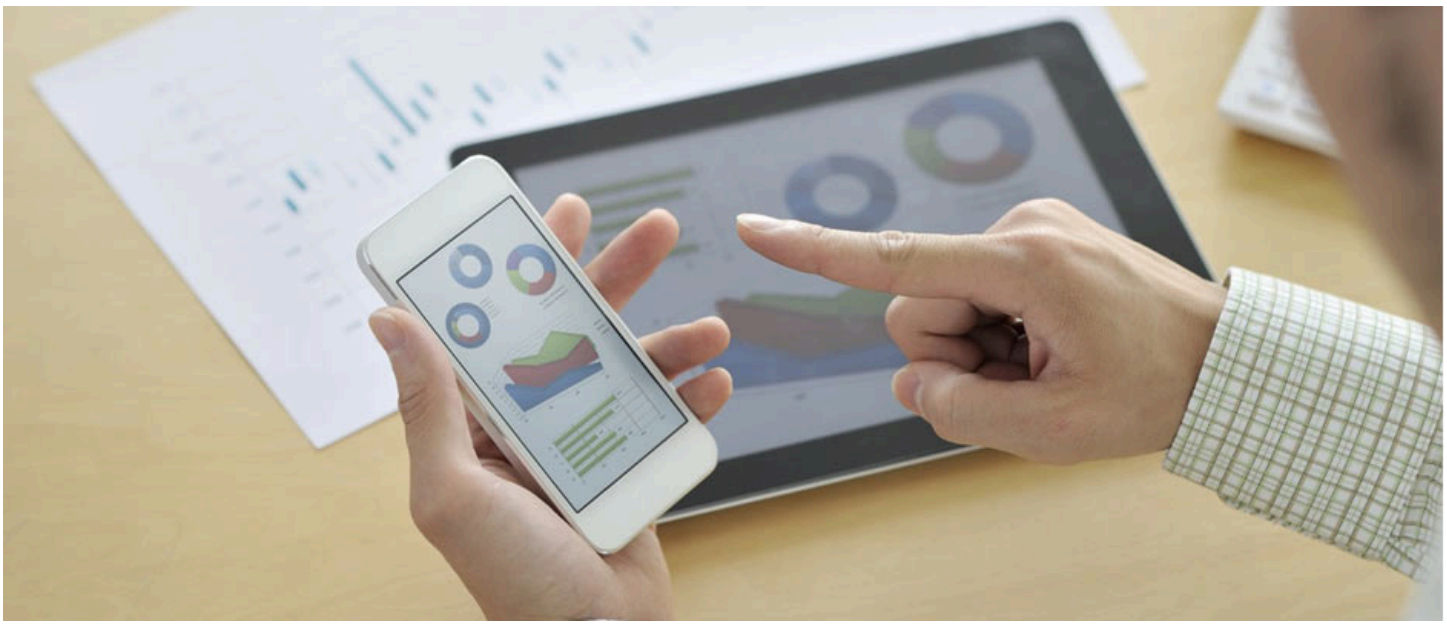
scientiamobile

"The first step in a great mobile experience"



WURFL MICROSERVICE CLIENT API FOR PYTHON

Support

The ScientiaMobile Enterprise Support Portal is open to all WURFL users, both commercial license holders and evaluation users. It represents the combined knowledge base for the WURFL community. Commercial licensees are invited to post questions in the forum using the account to which their licenses are associated. This may mean faster handling of those posts by ScientiaMobile's personnel.

For commercial license holders, there are tiered support levels to address a variety of business support needs. After logging into your account, commercial licensees with support options can access the Enterprise Support portal to post tickets. These tickets will receive expedited attention.

To inquire about support plans, use our License Inquiry or our General Inquiry form.

Update Notifications

If you would like to be notified of our API updates, major data updates, and other technical changes, please subscribe to our ScientiaMobile Announcements list.

scientiamobile

www.scientiamobile.com

Tel +1.703.310.6650

E-mail: sales@scientiamobile.com

Copyright © 2026 ScientiaMobile, all rights reserved. WURFL Cloud, WURFL OnSite, WURFL and, InFuze WURFL InSight and respective logos are trademarks of ScientiaMobile. Apache is the trademark of the Apache Software Foundation. NGINX is the trademark of Nginx Software Inc. Varnish is the trademark of Varnish Software AB

WURFL Microservice Client API for Python

> ***Note:*** *This guide assumes that you have access to a running instance of the WURFL Microservice HTTP Server, either through ScientiaMobile's **Docker Registry**, the **AWS Marketplace**, the **Azure Marketplace**, or other distribution channels. The guide also assumes familiarity with **Device Detection (WURFL)** and **WURFL Capabilities**.*

WURFL Capabilities

WURFL Capabilities: WURFL capabilities are device properties represented in WURFL. While users of the WURFL Microservice for Docker have access to the list of licensed capabilities, customers who obtained the product through one of the Marketplace will have access to the **WURFL capability list, predefined for the product they have licensed**. Please note that there are two types of capabilities supported in WURFL: Static and Virtual.

While the difference between the two is mostly immaterial for you as a user as far as their practical usage goes, you still need to use two separate methods to use one or the other.

WURFL Microservice Client API (`wmclient`): Given an HTTP Request and a capability, the WURFL Client will return the property value.

> ***Note:*** *While the `wmclient` is an API, it requires interaction with the WURFL Microservice server to work. This introduces some latency (hugely mitigated by a built-in caching layer). For this reason, ScientiaMobile does not refer to the WURFL Microservice Client (`wmclient`) as a "WURFL API". That name is reserved for the **WURFL OnSite APIs**.*

Obtaining, Installing and Running the `wmclient`

In order to use the `wmclient` please follow these instructions:

1. Download example.py from **WURFL repository**.
2. Edit `example.py` (in the package) by modifying the `create()` call to include the public IP address of your AMI instance, Azure VM instance or Docker container service (i.e. replacing localhost with the IP address of your running HTTP server, be it an AMI instance, Azure VM or Docker container service).

> ***Note:*** *There are multiple ways in which a service deployed through Docker can be exposed to the world around it. Please refer to your DevOps if in doubt of the actual address at which the service is running.**

3. Get dependencies (`pip install pycurl` and `pip install requests`) and run the included `example.py` file. (`python example.py`) Please note that if you clone the whole wmclient repository, you won't need to download the wmclient dependency from Pypi, while, if you download only the example.py file, you'll have to do `pip install wmclient`

Here's the code for your reference:

```
from wmclient import *

try:
    client = WmClient.create("http", "localhost", 8080, "")

    info = client.get_info()
    print("Printing WM server information")
    print("WURFL API version: " + info.wurfl_api_version)
    print("WM server version:  " + info.wm_version)
    print("Wurfl file info: " + info.wurfl_info)

    ua = "Mozilla/5.0 (Linux; Android 7.1.1; ONEPLUS A5000 Build/NMF26X)
AppleWebKit/537.36 (KHTML, like Gecko) " \
         "Chrome/56.0.2924.87 Mobile Safari/537.36 "

    client.set_requested_static_capabilities(["brand_name", "model_name"])
    client.set_requested_virtual_capabilities(["is_smartphone", "form_factor"])
    print()
    print("Detecting device for user-agent: " + ua);

    # Perform a device detection calling WM server API
    device = client.lookup_useragent(ua)

    if device.error is not None and len(device.error) > 0:
        print("An error occurred: " + device.error)
    else:
        # Let's get the device capabilities and print some of them
        capabilities = device.capabilities
        print("Detected device WURFL ID: " + capabilities["wurfl_id"])
        print("Device brand & model: " + capabilities["brand_name"] + " " +
capabilities["model_name"])
        print("Detected device form factor: " + capabilities["form_factor"])
        if capabilities["is_smartphone"] == "true":
            print("This is a smartphone")
            # Iterate over all the device capabilities and print them
            print("All received capabilities");
            for k in capabilities:
                print(k + ": " + capabilities[k])

            # Get all the device manufacturers, and print the first twenty
            print()
            limit = 20
            deviceMakes = client.get_all_device_makes()
            print("Print the first {} Brand of {} retrieved from
server\n".format(limit, len(deviceMakes)))

            # Sort the device manufacturer names
            list.sort(deviceMakes)
            for i in range(limit):
```

```
                print(" - {}\n".format(deviceMakes[i]))

            # Now call the WM server to get all device model and marketing names
produced by Apple
            print("Print all Model for the Apple Brand")
            devNames = client.get_all_devices_for_make("Apple")

            for model_mkt_name in devNames:
                print(" - {} {}\n".format(model_mkt_name.brand_name,
model_mkt_name.model_name))

            # Now call the WM server to get all operative system names
            print("Print the list of OSes")
            oses = client.get_all_OSes()
            # Sort and print all OS names
            list.sort(oses)
            for os in oses:
                print(" - {}\n".format(os))

            # Let's call the WM server to get all version of the Android OS
            print("Print all versions for the Android OS")
            osVersions = client.get_all_versions_for_OS("Android")
            # Sort all Android version numbers and print them.
            list.sort(osVersions)
            for ver in osVersions:
                print(" - {}\n".format(ver))

except WmClientError as wme:
    # problems such as network errors  or internal server problems
    print("An error has occurred: " + wme.message)
```

If you are using device detection from code running as part of an HTTP server, it is highly recommended that you pass the complete HTTP request to the `lookup_request` method as follows:

```
device = client.lookup_request(req)
```

where `req` is assumed to be a request from the `requests` python package.

For a complete reference of all static and virtual capabilities, **you can refer to this document**. Please note that WURFL Microservice for the AWS Marketplace comes with a **predefined set of capabilities**.

Complete reference of the wmclient API

```
PACKAGE DOCUMENTATION

def create(scheme, host, port, baseURI)
Creates a new WURFL Microservice client. The client has a default timeout of 10
seconds (set as 10000 milliseconds). If not scheme is give, client assumes a 'http'
value.

def destroy(self)
Closes and deallocates all resources used to connect to server. Function calls made
after this one will cause error

def lookup_useragent(self, useragent)
Performs a device detection from the given User-Agent. If the User-Agent is None or
empty a generic device is returned

def lookup_request(     self,req)
Performs a device detection from the headers carried by the give HTTP request. The
request object is assumed to be the one used in requests python framework. If the
User-Agent header is None or empty a generic device is returned

def lookup_device_id(self, wurflId)
Retrieves a device with the given WURFL ID. If the WURFL ID is None, empty or wring a
WMClientError is raised

def get_all_OSes(self)
return a list of all device OS

def get_all_device_makes(self)
returns a list of all device makers

def get_all_devices_for_make(self, make)
returns a list of all device models for the given maker

def get_all_versions_for_OS(self, osName)
returns a list of all the known versions for the given device OS

def get_api_version(self)
returns the version of this WM Client

def get_info(self)
Returns info about WURFL microservice server. It raises a WmClientError in case of
connection errors/timeouts

def has_static_capability(self, capName)
returns True if the given static capability is handled by this client, False
otherwise

def has_virtual_capability(self, capName)
returns True if the given virtual capability is handled by this client, False
```

```
otherwise

def set_http_timeout(self, timeout)
Sets HTTP connection timeout in milliseconds

def set_requested_capabilities( self, capsList)
Sets the list of virtual and static capabilities handled by this client


def set_requested_static_capabilities(  self, capsList)
Sets the list of static capabilities handled by this client


def set_requested_virtual_capabilities( self, capsList)
Sets the list of virtual capabilities handled by this client
```

More docs can be found in the **WURFL Microservice client for python repository**

Error in case of Missing Capabilities

As mentioned above, depending on the configuration of your WURFL Microservice, one of the WURFL capabilities may not be available. In this case, the JSONDeviceData.capabilities dictionary will not contain an entry for that capability and caller will get either an error or the zero value for it (None), depending on how the user accesses the capability dictionary.

Presence of a capability can be safely checked with:

```
value = JSONDeviceData.capabilities.get("has_cellular_radio")
```

And check if it has a value:

```
if value is not None:
    # do something...
```

Additional capabilities can always be obtained by upgrading to greater version of the AMIs in the AWS Marketplace or by contacting ScientiaMobile to license additional capabilities for the Docker Image or other products.

© 2026 ScientiaMobile Inc.
All Rights Reserved.

NOTICE: All information contained herein is, and remains the property of ScientiaMobile Incorporated and its suppliers, if any. The intellectual and technical concepts contained herein are proprietary to ScientiaMobile Incorporated and its suppliers and may be covered by U.S. and Foreign Patents, patents in process, and are protected by trade secret or copyright law. Dissemination of this information or reproduction of this material is strictly forbidden unless prior written permission is obtained from ScientiaMobile Incorporated.