scientiamobile

"The first step in a great mobile experience"

INFUZE PYTHON MODULE USER GUIDE

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

WURFL InFuze Module for Python (PyWURFL)

IMPORTANT NOTE: As of December 31st 2023, Python 2.7 will no longer be supported.

PyWURFL is a Python module wrapping the InFuze WURFL C API and encapsulating it in an object-oriented manner, to provide a fast, intuitive interface. Only Python3 builds are supported.

Installing libwurfl

In order for the Module to work it is **ESSENTIAL** that the libwurfl library is installed on your system. libwurfl is provided in your Customer Vault/FileX.

If you have not already installed libwurfl, instructions can be found here. Release notes for each API can be found here.

Compatibility

Python 2.7 is end-of-life and as of December 31st 2023, PyWURFL has dropped support for Python 2.7. This document assumes you are using Python 3.

Installation

PyWURFL is distributed as a Python wheel package, and should be installed using pip.

DO NOT use pip install pywurfl to install the WURFL InFuze for Python module.

Once you have installed libwurfl, download the **WURFL InFuze for Python** package from my.scientiamobile.com, it will be a ZIP file like infuze_python-1.12.0.0-linux-universal.zip, follow these steps:

```
# Go to your Python project or any other directory if you're installing PyWURFL globally
# Make a subdirectory for the PyWURFL contents
mkdir pywurfl
cd pywurfl

# Download the package here and unzip it
unzip infuze_python-*.zip

# You will see a tar file, which needs to be unpacked as well
tar xvf python-mod_wurfl-*.tar

# Now you will see the wheel (.whl) files
pywurfl-3.0-py3-none-any.whl

# and the legacy .egg files

pywurfl-3.0-py3.4.egg
pywurfl-3.0-py3.5.egg
pywurfl-3.0-py3.6.egg
pywurfl-3.0-py3.7.egg
pywurfl-3.0-py3.8.egg
pywurfl-3.0-py3.9.egg
pywurfl-3.0-py2.7.egg

# Starting from version 3.0, the new pywurfl packages are
```

Next, you must determine which Python environment you want to install the module into, for example:

```
# Global Python 3
$ python3 --version
Python 3.7.5

# Global Python 3.8
$ python3.8 --version
Python 3.8.0
```

```
# venv Python 3.8
$ ./my-project/bin/python --version
Python 3.8.0
```

> *Note: It's important that you install PyWURFL in the right environment for your project - you can install the module in any or all of them, but your project won't be able to use PyWURFL unless you have installed it that environment. If you're installing it globally, your "environment" is the global environment, and you can simply use python3, for example.*

Once you've determined which Python environment to use, you can install the module. In this example, we'll use the Python virtual environment in our test project my-project:

> *Attention: **DO NOT** use pip install pywurfl to install the WURFL InFuze for Python module.*

```
# Check the version of Python
$ ./my-project/bin/python --version
Python 3.8.0

# This is Python 3.8, so we install the `py3` package:
$ ./my-project/bin/python -m pip install ./pywurfl-3.0-py3-none-any.whl
```

> *If you get an error like No module named pip, you must first install pip. This may be available from your package manager (usually python3-pip), or you can install it manually.*

Python wheel packages have a specific naming convention, in the example above, we installed pywurfl-3.0-py3-none-any.whl.

To determine the right package for you, find the major version of your Python interpreter (with python3 --version, for example), and take the first digit, (ex: Python 3.8.1 => 3). This corresponds to the python tag, which starts with py, for example Python 3.6.5 would be py3. As mentioned above, starting from version 3.0, pywurfl packages are universal, the â€˜noneâ€™ and â€˜anyâ€™ tokens in the package name mean that it is not OS-specific and that it is suitable for all architectures.

Sample Usage

On Windows OS, if you have customized the libwurfl library installation path, you will have to set the environment variable WURFL_SHARED_LIBRARY_PATH with the full path of the libwurfl.dll file with the value you have chosen when installing libwurfl from the .msi installer.

Here is an example to get started using PyWURFL:

```
# Import WURFL InFuze (PyWURFL) module
from pywurfl.wurfl import Wurfl

# Create a WURFL Engine. Please note that the installed wurfl.zip path may change.
# for example, on OS X systems, it will be in `/usr/local/share/wurfl/wurfl.zip`
# on Linux systems, it will be in `/usr/share/wurfl/wurfl.zip`.
wurfl = Wurfl('/usr/share/wurfl/wurfl.zip')

# Lookup an HTTP request
http_request = {
    "accept-encoding": "gzip, deflate, br",
    "accept-language": "en-US,en;q=0.9",
    "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp",
    "user-agent": " Mozilla/5.0 (Linux; Android 10; SM-G981U1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80
.0.3987.132 Mobile Safari/537.36",
}

dev = wurfl.parse_headers(http_request)

# You can also lookup a device with just the user-agent string
# dev = wurfl.parse_useragent(user_agent)
```

```
# retrieve some properties and capabilities values

# WURFL device ID:
print("device id =", dev.id)

# Some static capabilities:
static_capabilities = ["model_name", "brand_name", "device_os"]

# Retrieve the value of a single static capability:
print("get_capability('model_name') =",
      dev.get_capability(static_capabilities[0]))

# Retrieve the value of many static capabilities at once:
print("get_capabilities(static_capabilities) =",
      dev.get_capabilities(static_capabilities))

# Some virtual capabilities:
virtual_capabilities = ["complete_device_name", "form_factor"]

# Retrieve the value of a single virtual capability:
print("get_virtual_capability('complete_device_name') =",
      dev.get_virtual_capability(virtual_capabilities[0]))

# Retrieve the value of many virtual capabilities at once:
print("get_virtual_capabilities(virtual_capabilities) =",
      dev.get_virtual_capabilities(virtual_capabilities))

# Make sure you release the device when you are finished
dev.release()
```

Here we create the Wurfl engine, then obtain the device object through lookup on a user agent string. Then we get device properties and static and virtual capability values as needed. Please note that virtual capabilities are calculated at runtime, so they might be significantly slower than static capabilities.

By running the above code, you should get an output like:

```
device id = samsung_sm_g981u_ver1_subuau1
get_capability('model_name') = SM-G981U1
get_capabilities(static_capabilities) = {'model_name': 'SM-G981U1', 'brand_name': 'Samsung', 'device_os': 'Androi
d'}
get_virtual_capability('complete_device_name') = Samsung SM-G981U1 (Galaxy S20 5G)
get_virtual_capabilities(virtual_capabilities) = {'complete_device_name': 'Samsung SM-G981U1 (Galaxy S20 5G)',
'form_factor': 'Smartphone'}
```

WURFL Updater

If you want to keep your wurfl.zip up-to-date with ScientiaMobile's data release schedule, please consider using the Updater features, available in WURFL InFuze for Python as follows:

After creating your WURFL engine, set your personal WURFL Snapshot URL (in the form "https://data.scientiamobile.com/xxxxx/wurfl.zip", available from your license in my.scientiamobile.com):

```
wurfl = Wurfl('wurfl.zip')
try:
    wurfl.set_updater_data_url("https://data.scientiamobile.com/xxxxx/wurfl.zip")
except Exception as exception:
    print("Error while setting updater data URL: ")
    print(exception)
```

> *Note: you must use the same file type (zip or gz) in the updater URL that you use in the initial Wurfl() construction.*

For long-running scripts, you can specify the frequency you want to check for updates: (DAILY or WEEKLY, default is DAILY):

```
wurfl.set_updater_data_frequency(wurfl.UPDATER_FREQUENCIES["DAILY"])
```

Note: the wurfl path should be writable, and a wurfl.zip file must already be present in order for the Updater to determine whether or not an update is required.

Then start the updater:

```
try:
    wurfl.updater_start()
except Exception as exception:
    print("Error while starting the updater: ")
    print(exception)
```

Updater will run a periodic check for the latest release of the wurfl.zip file, download it, and update the running engine to the latest version - all during normal application operations.

The internal updater also supports simple file logging, useful in debugging network problems and the like:

```
wurfl.set_updater_log_path("updater.log")
```

Please note that:

- The WURFL data file and the path where it resides, specified in the WURFL engine construction, **MUST** have write/rename access. The old data file will be replaced (i.e. a rename operation will be performed) with the updated version upon successful update operation completion, and the directory will be used for temp file creation, etc.
- ScientiaMobile does not distribute uncompressed XML data files via the updater. This means that, if you plan to use the updater, you **MUST** use the compressed (i.e. a ZIP or a XML.GZ) data file in the engine construction call.

set_updater_data_frequency() sets how often the updater **checks** for an updated data file, not how often the engine data file is actually updated.

The WURFL InFuze Updater functionality relies on availability and features of the well-known and widely available curl command-line utility. A check for curl availability is done in the set_updater_data_url() call.