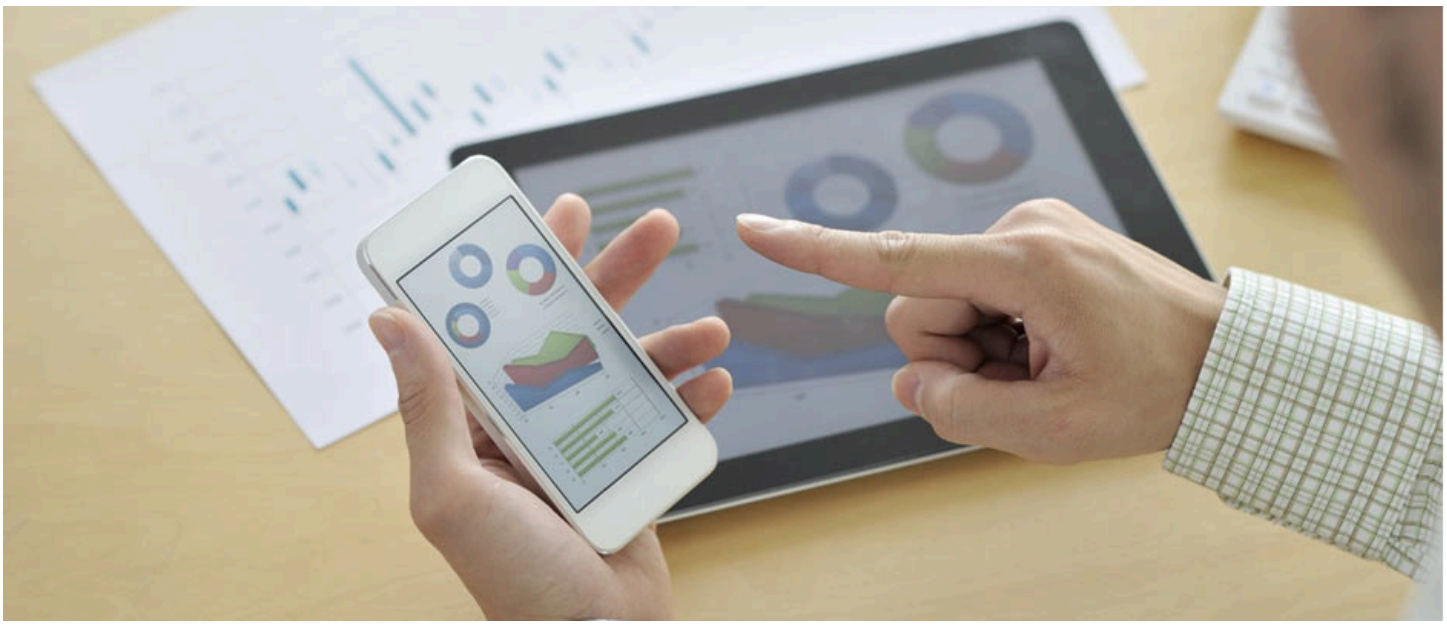
scientiamobile

"The first step in a great mobile experience"



WURFL INFUZE MODULE FOR RUST : 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

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 InFuze Module for rust : User Guide

WURFL InFuze for rust is a Rust language module wrapping the WURFL C API and encapsulating it in two rust types to provide a fast and intuitive interface. It is compatible on `linux` platforms for Rust `edition 2021`, version 1.85 and higher.

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 the **Downloads** section of your license in the **Customer Vault**.

If you have not already installed libwurfl, instructions can be found **here**. Changes in each release are listed in the **WURFL InFuze API Changelog**.

Installation on Linux

InFuze for Rust is available as a source code package. In order to use it, you have to install Rust and Cargo (the rust build and execution tool).

You can install Rust and Cargo with this command: `curl https://sh.rustup.rs -sSf | sh`

If everything goes fine, this message will appear: "Rust is installed now. Great!"

Unpack the contents of the rust-wurfl package in a directory of choice. For example: `tar xvzf rust-mod_wurfl-W.X.Y.Z.tar.gz`

You can build the `wurfl` to use it as a library with the command `cargo build --lib` from the rust-wurfl project root.

WURFL Data Snapshot

To perform lookups, you will need a copy of your WURFL data snapshot (also referred to as the `wurfl.xml`). While there is one included in the release package, it is intended to be a sample and will not contain all of your licensed capabilities. Your licensed WURFL data snapshot can be accessed by **following these directions**.

Usage

Here is an example to get started:

```rust
use wurfl::*;
use std::collections::HashMap;

fn main() {
    println!("Starting WURFL wrapper usage sample!");

    // download WURFL file from the ScientiaMobile Snapshot generator (replace the sample URL with your own specific customer URL to avoid error 402)
    // we use the current directory as destination
    let download_res = wurfl_download("https://data.scientiamobile.com/xxxxx/wurfl.zip", ".");
    match download_res {
        Ok(_) => println!("WURFL file downloaded successfully"),
        Err(_) => println!("WURFL file downloaded FAILED: keeping default version"),
    }

    // create the engine using the WURFL file downloaded in the current directory
    let wurfl_path = "wurfl.zip";
    let wurfl_res = Wurfl::new(wurfl_path, None, None, WurflCacheProvider::LRU, Some("100000"));
    let engine = match wurfl_res {
        Ok(engine) => engine,
        Err(error) => panic!("Problem initializing wurfl: {:?}", error),
    };
    println!("WURFL API Version created: {}", engine.get_api_version());

    let ua = "Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.0.0 Mobile Safari/537.36";
    // Create headers
    let mut headers = HashMap::new();
    headers.insert("User-Agent", ua);
    headers.insert("Sec-Ch-Ua", "\"Chromium\";v=\"94\", \"HuaweiBrowser\";v=\"99\", \";Not A Brand\";v=\"99\"");
    headers.insert("Sec-Ch-Ua-Full-Version", "99.123.456");
    headers.insert("Sec-Ch-Ua-Platform", "Android");
    headers.insert("Sec-Ch-Ua-Platform-Version", "12");
    headers.insert("Sec-Ch-Ua-Model", "Pixel 6");

    // lookup the device with the headers and Client Hints
    let device_res = engine.lookup_with_headers(headers);
    match device_res {
        Ok(d) => {
            println!("Device ID: {}", d.get_device_id());

            match d.get_virtual_cap("complete_device_name") {
                Ok(Some(device_name)) => println!("Complete device name: {}", device_name),
                Ok(None) => println!("Complete device name not available."),
                Err(e) => eprintln!("Error retrieving 'complete_device_name': {}",
```

```
e),
            }

            match d.get_virtual_cap("is_smartphone") {
                Ok(Some(is_smartphone)) => println!("Device is smartphone?: {}",
is_smartphone),
                Ok(None) => println!("Information about whether the device is a
smartphone is not available."),
                Err(e) => eprintln!("Error retrieving 'is_smartphone': {}", e),
            }
        }
        Err(_) => panic!("Lookup user agent failed"),
    };
}
```

You can also compile and run the example application with the command:

`cargo run --package wurfl --example example`

This code snippet creates the WURFL Engine once, then performs a lookup with a UserAgent string (you can also use any struct that implements the `IntoIterator` trait and contains the request headers directly using `engine.lookup_with_headers`) and get the static and virtual capabilities needed in your implementation (**NOTE:** virtual capabilities are calculated at runtime).

WURFL Updater

If you want to keep your `wurfl.zip` up-to-date with the ScientiaMobile data release schedule, then you might want to use the Updater features.

After creating your WURFL Engine, set your personal WURFL Snapshot URL (in the form "https://data.scientiamobile.com/xxxxx/wurfl.zip", available on the **Snapshot generator** page of your license in the Customer Vault):

```
let we = engine.set_updater_data_url(wurfl_download_url);
    let no_error = we.is_none();
    // handle error or Ok cases ...
```

Specify the periodicity you would like for update checks (for example, the Daily update frequency):

```
let we =
engine.set_updater_data_frequency(WurflUpdaterFrequency::WurflUpdaterFrequencyDaily);
```

Do note that a wurfl.zip file must already be present in a writable path in order for the updater to check the file and determine whether or not it needs to update the file.

Start the updater:

```
let we_h = engine.updater_start();
```

Updater will run a daily 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.

Updater methods usually return an `Option<WurflError>`

API Reference

Index

- Constants
- type Device
  - fn get_capabilities
  - fn get_virtual_capabilities
  - fn get_static_cap
  - fn get_device_id
  - fn get_match_type
  - fn (*Device) get_normalized_user_agent
  - fn get_original_user_agent
  - fn get_root_id
  - fn get_user_agent
  - fn get_virtual_cap
  - fn is_root
- fn wurfl_download
- type Wurfl
  - fn new
  - fn get_api_version
  - fn get_all_caps
  - fn get_all_vcaps
  - fn get_info
  - fn get_last_load_time
  - fn has_capability
  - fn has_virtual_capability
  - fn lookup_device_id
  - fn lookup_device_id_with_headers
  - fn lookup_with_headers

- fn lookup_useragent
- fn set_updater_data_frequency
- fn set_updater_data_url
- fn set_updater_data_url_timeout
- fn updater_runonce
- fn updater_start
- fn updater_stop

Constants

Cache Provider possible values

```
pub enum WurflCacheProvider {
    NoCache,
    LRU,
}
```

Match type

```
pub enum Type {
    WurflMatchTypeExact,
    WurflMatchTypeConclusive,
    WurflMatchTypeRecovery,
    WurflMatchTypeCatchall,
    WurflMatchTypeNone,
    WurflMatchTypeCached,
}
```

Wurfl updater frequency

```
pub enum WurflUpdaterFrequency {
    WurflUpdaterFrequencyDaily,
    WurflUpdaterFrequencyWeekly,
}
```

type Device

Device provides access to device static and virtual capabilities

fn get_capabilities

```
pub fn get_capabilities(&self,cap_names: &[&str]) -> Result<HashMap<String, String>,
WurflError>]
```

Returns a map -> key:cap_name,value:cap_value for the calling device

fn get_virtual_capabilities

```
pub fn get_virtual_capabilities(&self,vcap_names: &[&str]) -> Result<HashMap<String,
String>, WurflError>]
```

Returns a map -> key:vcap_name,value:vcap_value for the calling device

fn get_static_cap

```
pub fn get_static_cap(&self, capability_name: &str) -> Result<Option<&str>,
WurflError>
```

Returns the device value of the static capability with the given name

fn get_device_id

```
pub fn get_device_id(&self) -> &str
```

Get wurfl_id string from device handle

fn get_match_type

```
 pub fn get_match_type(&self) -> MatchType
```

Get type of Match occurred in lookup

fn (*Device) get_normalized_user_agent

```
pub fn get_normalized_user_agent(&self) -> &str {
```

Get the Normalized (processed by wurfl api) userAgent

fn get_original_user_agent

```
pub fn get_original_user_agent(&self) -> &str
```

Get the original UserAgent of matched device (the one passed to lookup)

fn get_root_id

```
pub fn get_root_id(&self) -> &str
```

Returns the unique ID of this device ancestor in the WURFL hierarchy.

fn get_user_agent

```
pub fn get_user_agent(&self) -> &str
```

Get default UserAgent of matched device (might be different from UA passed to lookup)

fn get_virtual_cap

```
pub fn get_virtual_cap(&self, virtual_capability_name: &str) -> Result<Option<&str>,
WurflError>
```

Get Virtual Capability

fn is_root

```
pub fn is_root(&self) -> bool
```

Returns true if this device is a root device in the WURFL hierarchy

fn wurfl_download

```
pub fn wurfl_download(url: &str, dest: &str) -> Result<(), WurflError>
```

Downloads a file from the given Snapshot Generator url to the given destination. The destination must exist and be writable.

Returns an error if the file could not be downloaded (connection error, write error, etc).

type Wurfl

WURFL holds data and exposes methods used to perform device detection and access device capabilities.

fn new

```
 pub fn new(wurfl_xml: &str, patches: Option<&[&str]>, cap_filter: Option<&[&str]>,
              cache_provider: WurflCacheProvider, cache_extra_config: Option<&str>)
-> Result<Wurfl, WurflError>
```

Create the wurfl engine. Parameters :

```
- wurfl_xml : path to the wurfl.xml/zip file
- patches : vector of paths of patches files to load
- cap_filter : don't use it (pass None as argument)
- cache_provider : WurflCacheProviderLru or NoCache
- cache_extra_config : size of lru cache in the form "100000"
```

fn get_api_version

```
pub fn get_api_version(&self) -> &str
```

Returns version of internal InFuze API

fn get_all_caps

```
pub fn get_all_caps(&self) -> Vec<String>
```

Return all capabilities names

fn get_all_vcaps

```
pub fn get_all_vcaps(&self) -> Vec<String>
```

Returns all virtual capabilities names

fn get_info

```
pub fn get_info(&self) -> &str
```

Get wurfl.xml info

fn get_last_load_time

```
pub fn get_last_load_time(&self) -> &str
```

Get last wurfl.xml load time

fn has_capability

```
 pub fn has_capability(&self, cap_name: &str) -> bool
```

Return true if the capability exists

fn has_virtual_capability

```
 pub fn has_virtual_capability(&self, vcap_name: &str) -> bool
```

Return true if the virtual capability exists

fn lookup_device_id

```
pub fn lookup_device_id(&self, device_id: &str) -> Result<Device, WurflError>
```

Lookup by wurfl_ID and return Device handle

fn lookup_device_id_with_headers

```
pub fn lookup_device_id_with_headers<U, V, T: IntoIterator<Item=(U, V)>>(&self,
device_id: &str, headers: T) -> Result<Device, WurflError>
```

Lookup by wurfl_ID and request headers and return Device handle

fn lookup_with_headers

```
pub fn lookup_with_headers<U, V, T: IntoIterator<Item=(U, V)>>(&self, headers: T) ->
Result<Device, WurflError> where U: ToString, V: AsRef<[u8]>
```

Lookup Request and return Device handle

fn lookup_useragent

```
pub fn lookup_useragent(&self, user_agent: &str) -> Result<Device, WurflError>
```

Lookup up useragent and return Device handle

fn set_updater_data_frequency

```
pub fn set_updater_data_frequency(&self, freq: WurflUpdaterFrequency) ->
Option<WurflError>
```

Set interval of update checks

fn set_updater_data_url

```
pub fn set_updater_data_url(&self, data_url: &str) -> Option<WurflError>
```

Set your HTTPS updater URL, as shown on your **Snapshot generator** page in the Customer Vault

fn set_updater_data_url_timeout

```
pub fn set_updater_data_url_timeout(&self, conn_timeout: i32, data_transfer_timeout:
i32) -> Option<WurflError>
```

Set path of updater log file

fn updater_runonce

```
pub fn updater_runonce(&self) -> Option<WurflError>
```

Start updater process once and wait for termination

fn updater_start

```
pub fn updater_start(&self) -> Option<WurflError>
```

Start the updater thread

fn updater_stop

```
pub fn updater_stop(&self) -> Option<WurflError>
```

Stop the updater thread

© 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.