



# INFUZE GOLANG MODULE USER GUIDE

## Support

The [ScientiaMobile Support Forum](#) 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

# WURFL InFuze Module for golang : User Guide

WURFL InFuze for golang is a GO language module wrapping the WURFL C API and encapsulating it in two golang types to provide a fast and intuitive interface. It is compatible on linux/macOS platforms for golang 1.7 or 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 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](#).

## Installation on Linux/MacOS X

InFuze for golang is available as a binary package. To install it select the package for the correct go version and untar the contents in your GOPATH folder. For example, on go-1.8:

```
# cd $GOPATH
# tar xvzf golang-1.8-wurfl-1.9.4.0-binary.tar.gz
```

## Usage

Here is an example to get started:

```
package main
```

```
import (
    "fmt"
    "html"
    "net/http"
    "scientiamobile/wurfl"
)
```

```
func main() {
```

```
    var wengine *wurfl.Wurfl
    var device *wurfl.Device
```

```
    wengine, err := wurfl.Create("/usr/share/wurfl/wurfl.zip", nil, nil, -1, wurfl.WurflCacheProviderLru, "100000")
```

```
    if err != nil {
        fmt.Println(err)
    }
```

```
    ua := "Dalvik/1.6.0 (Linux; U; Android 4.3; SM-N900T Build/JSS15J)"
```

```
    device, err = wengine.LookupUserAgent(ua)
```

```

deviceid, err := device.GetDeviceID()

fmt.Println(deviceid)

fmt.Println(device.GetCapability("device_os"))
fmt.Println(device.GetVirtualCapability("is_android"))

device.Destroy()
wengine.Destroy()
}

```

Create the WURFL Engine once, then lookup UserAgent (or a request directly using `wengine.LookupRequest()`) 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`, with `xxxxx` replaced with your personal access token, located in your license account page):

```

uerr := wengine.SetUpdaterDataURL(Url)
if uerr != nil {
    fmt.Printf("SetUpdaterDataURL returned : %s\n", uerr.Error())
}

```

Specify the periodicity you would like for update checks:

```

_ = wengine.SetUpdaterDataFrequency(wurfl.WurflUpdaterFrequencyDaily)

```

Then start the updater:

```

uerr = wengine.UpdaterStart()
if uerr != nil {
    fmt.Printf("UpdaterStart returned : %s\n", uerr.Error())
}

```

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.

## API Reference

### Index

- [Constants](#)
- [type Device](#)
  - [func \(d \\*Device\) Destroy\(\)](#)
  - [func \(d \\*Device\) GetCapabilities\(caps \[\]string\) \[\]string](#)
  - [func \(d \\*Device\) GetCapability\(cap string\) string](#)
  - [func \(d \\*Device\) GetDeviceID\(\) \(string, error\)](#)
  - [func \(d \\*Device\) GetMatchType\(\) int](#)
  - [func \(d \\*Device\) GetNormalizedUserAgent\(\) \(string, error\)](#)

- [func \(d \\*Device\) GetOriginalUserAgent\(\) \(string, error\)](#)
- [func \(d \\*Device\) GetRootID\(\) string](#)
- [func \(d \\*Device\) GetUserAgent\(\) \(string, error\)](#)
- [func \(d \\*Device\) GetVirtualCapability\(vcap string\) string](#)
- [func \(d \\*Device\) IsRoot\(\) bool](#)
- [type Wurfl](#)
  - [func Create\(Wurflxml string, Patches \[\]string, CapFilter \[\]string, EngineTarget int, CacheProvider int, CacheExtraConfig string\) \(\\*Wurfl, error\)](#)
  - [func \(w \\*Wurfl\) Destroy\(\)](#)
  - [func \(w \\*Wurfl\) GetAPIVersion\(\) string](#)
  - [func \(w \\*Wurfl\) GetAllCaps\(\) \[\]string](#)
  - [func \(w \\*Wurfl\) GetAllVCaps\(\) \[\]string](#)
  - [func \(w \\*Wurfl\) GetEngineTarget\(\) string](#)
  - [func \(w \\*Wurfl\) GetInfo\(\) string](#)
  - [func \(w \\*Wurfl\) GetLastLoadTime\(\) string](#)
  - [func \(w \\*Wurfl\) GetUserAgentPriority\(\) string](#)
  - [func \(w \\*Wurfl\) HasCapability\(cap string\) bool](#)
  - [func \(w \\*Wurfl\) HasVirtualCapability\(vcap string\) bool](#)
  - [func \(w \\*Wurfl\) LookupDeviceID\(DeviceID string\) \(\\*Device, error\)](#)
  - [func \(w \\*Wurfl\) LookupDeviceIDWithRequest\(DeviceID string, r \\*http.Request\) \(\\*Device, error\)](#)
  - [func \(w \\*Wurfl\) LookupRequest\(r \\*http.Request\) \(\\*Device, error\)](#)
  - [func \(w \\*Wurfl\) LookupUserAgent\(ua string\) \(\\*Device, error\)](#)
  - [func \(w \\*Wurfl\) SetUpdaterDataFrequency\(Frequency int\) error](#)
  - [func \(w \\*Wurfl\) SetUpdaterDataURL\(DataURL string\) error](#)
  - [func \(w \\*Wurfl\) SetUpdaterLogPath\(LogFile string\) error](#)
  - [func \(w \\*Wurfl\) SetUserAgentPriority\(prio int\)](#)
  - [func \(w \\*Wurfl\) UpdaterRunonce\(\) error](#)
  - [func \(w \\*Wurfl\) UpdaterStart\(\) error](#)
  - [func \(w \\*Wurfl\) UpdaterStop\(\) error](#)

## Constants

```
const (
    WurflUserAgentPriorityOverrideSideloadedBrowserUserAgent = C.WURFL_USERAGENT_PRIORITY_OVERRIDE_SIDELOADED_BROWSER_USERAGENT
    WurflUserAgentPriorityUsePlainUserAgent                = C.WURFL_USERAGENT_PRIORITY_USE_PLAIN_USERAGENT
)
```

UserAgent priority possible values

```
const (
    WurflCacheProviderNone    = C.WURFL_CACHE_PROVIDER_NONE
    WurflCacheProviderLru     = C.WURFL_CACHE_PROVIDER_LRU
    // DOUBLE LRU is deprecated
    WurflCacheProviderDoubleLru = C.WURFL_CACHE_PROVIDER_DOUBLE_LRU
)
```

Cache Provider possible values

```
const (
    WurflMatchTypeExact      = C.WURFL_MATCH_TYPE_EXACT
    WurflMatchTypeConclusive = C.WURFL_MATCH_TYPE_CONCLUSIVE
    WurflMatchTypeRecovery   = C.WURFL_MATCH_TYPE_RECOVERY
    WurflMatchTypeCatchall   = C.WURFL_MATCH_TYPE_CATCHALL
    WurflMatchTypeHighPerformance = C.WURFL_MATCH_TYPE_HIGHPERFORMANCE
    WurflMatchTypeNone       = C.WURFL_MATCH_TYPE_NONE
    WurflMatchTypeCached     = C.WURFL_MATCH_TYPE_CACHED
)
```

Match type

```
const (
    WurflUpdaterFrequencyDaily = C.WURFL_UPDATER_FREQ_DAILY
    WurflUpdaterFrequencyWeekly = C.WURFL_UPDATER_FREQ_WEEKLY
)
```

Wurfl updater frequency

## type Device

```
type Device struct {
    Device C.wurfl_device_handle
    Wurfl  C.wurfl_handle
}
```

Device represents internal matched device handle

## func (\*Device) Destroy

```
func (d *Device) Destroy()
```

Destroy device handle, should be called when device attributes are no longer needed

## func (\*Device) GetCapabilities

```
func (d *Device) GetCapabilities(caps []string) []string
```

Get a list of Capabilities

## func (\*Device) GetCapability

```
func (d *Device) GetCapability(cap string) string
```

Get a single Capability

## func (\*Device) GetDeviceID

```
func (d *Device) GetDeviceID() (string, error)
```

Get wurfl\_id string from device handle

## func (\*Device) GetMatchType

```
func (d *Device) GetMatchType() int
```

Get type of Match occurred in lookup

## func (\*Device) GetNormalizedUserAgent

```
func (d *Device) GetNormalizedUserAgent() (string, error)
```

Get the Normalized (processed by wurfl api) userAgent

## func (\*Device) GetOriginalUserAgent

```
func (d *Device) GetOriginalUserAgent() (string, error)
```

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

## func (\*Device) GetRootID

```
func (d *Device) GetRootID() string
```

Retrieve the root device ID of this device.

## func (\*Device) GetUserAgent

```
func (d *Device) GetUserAgent() (string, error)
```

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

## func (\*Device) GetVirtualCapability

```
func (d *Device) GetVirtualCapability(vcap string) string
```

Get Virtual Capability

## func (\*Device) IsRoot

```
func (d *Device) IsRoot() bool
```

True if device is device root

## type Wurfl

```
type Wurfl struct {  
    Wurfl C.wurfl_handle  
}
```

Wurfl represents internal wurfl infuze handle

## func Create

```
func Create(Wurflxml string, Patches []string, CapFilter []string, CacheProvider int, CacheExtraConfig string) (*Wurfl, error)
```

Create the wurfl engine.

Wurflxml - path to the wurfl.xml/zip file Patches - slice of paths of patches files to load CapFilter - list of capabilities used; allow to init engine without loading all 500+ caps CacheProvider - WurflCacheProviderLru CacheExtraConfig - size of single lru cache in the form "100000"

## func (\*Wurfl) Destroy

```
func (w *Wurfl) Destroy()
```

Destroy the wurfl engine

## **func (\*Wurfl) GetAPIVersion**

func (w \*Wurfl) GetAPIVersion() string

Returns version of internal InFuze API

## **func (\*Wurfl) GetAllCaps**

func (w \*Wurfl) GetAllCaps() []string

Return all capabilities names

## **func (\*Wurfl) GetAllVCaps**

func (w \*Wurfl) GetAllVCaps() []string

Return all virtual capabilities names

## **func (\*Wurfl) GetInfo**

func (w \*Wurfl) GetInfo() string

Get wurfl.xml info

## **func (\*Wurfl) GetLastLoadTime**

func (w \*Wurfl) GetLastLoadTime() string

Get last wurfl.xml load time

## **func (\*Wurfl) GetUserAgentPriority**

func (w \*Wurfl) GetUserAgentPriority() string

Tells if WURFL is using the plain user agent or the sideloaded browser user agent for device detection

## **func (\*Wurfl) HasCapability**

func (w \*Wurfl) HasCapability(cap string) bool

Return true if the capability exists

## **func (\*Wurfl) HasVirtualCapability**

func (w \*Wurfl) HasVirtualCapability(vcap string) bool

Return true if the virtual capability exists

## **func (\*Wurfl) LookupDeviceID**

func (w \*Wurfl) LookupDeviceID(DeviceID string) (\*Device, error)

Lookup by wurfl\_ID and return Device handle

## **func (\*Wurfl) LookupDeviceIDWithRequest**

func (w \*Wurfl) LookupDeviceIDWithRequest(DeviceID string, r \*http.Request) (\*Device, error)

Lookup by wurfl\_ID and request headers and return Device handle

## func (\*Wurfl) LookupRequest

```
func (w *Wurfl) LookupRequest(r *http.Request) (*Device, error)
```

Lookup Request and return Device handle

## func (\*Wurfl) LookupUserAgent

```
func (w *Wurfl) LookupUserAgent(ua string) (*Device, error)
```

Lookup up useragent and return Device handle

## func (\*Wurfl) SetUpdaterDataFrequency

```
func (w *Wurfl) SetUpdaterDataFrequency(Frequency int) error
```

Set interval of update checks

## func (\*Wurfl) SetUpdaterDataURL

```
func (w *Wurfl) SetUpdaterDataURL(DataURL string) error
```

Set your scientiamobile vault https updater url

## func (\*Wurfl) SetUpdaterLogPath

```
func (w *Wurfl) SetUpdaterLogPath(LogFile string) error
```

Set path of updater log file

## func (\*Wurfl) SetUserAgentPriority

```
func (w *Wurfl) SetUserAgentPriority(prio int)
```

Sets which UA wurfl is using (plain or sideloaded)

## func (\*Wurfl) UpdaterRunonce

```
func (w *Wurfl) UpdaterRunonce() error
```

Start updater process once and wait for termination

## func (\*Wurfl) UpdaterStart

```
func (w *Wurfl) UpdaterStart() error
```

Start the updater thread

## func (\*Wurfl) UpdaterStop

```
func (w *Wurfl) UpdaterStop() error
```

Stop the updater thread

**IMPORTANT - Decommissioning of Engine Target options** Prior to version 1.9 of the API, users could choose between `WURFL_ENGINE_TARGET_HIGH_PERFORMANCE` and `WURFL_ENGINE_TARGET_HIGH_ACCURACY` engine optimization options. These options had been



*introduced years ago to manage the behavior of certain web browsers and their tendency to present "always different" User-Agent strings that would baffle strategies to cache similar WURFL queries in memory.*

*As the problem has been solved by browser vendors, the need to adopt this strategy has diminished and ultimately disappeared (i.e. there was no longer much to be gained with the performance mode in most circumstances) and ScientiaMobile elected to "remove" this option to simplify configuration and go in the direction of uniform API behavior in different contexts.*

**© 2018 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.