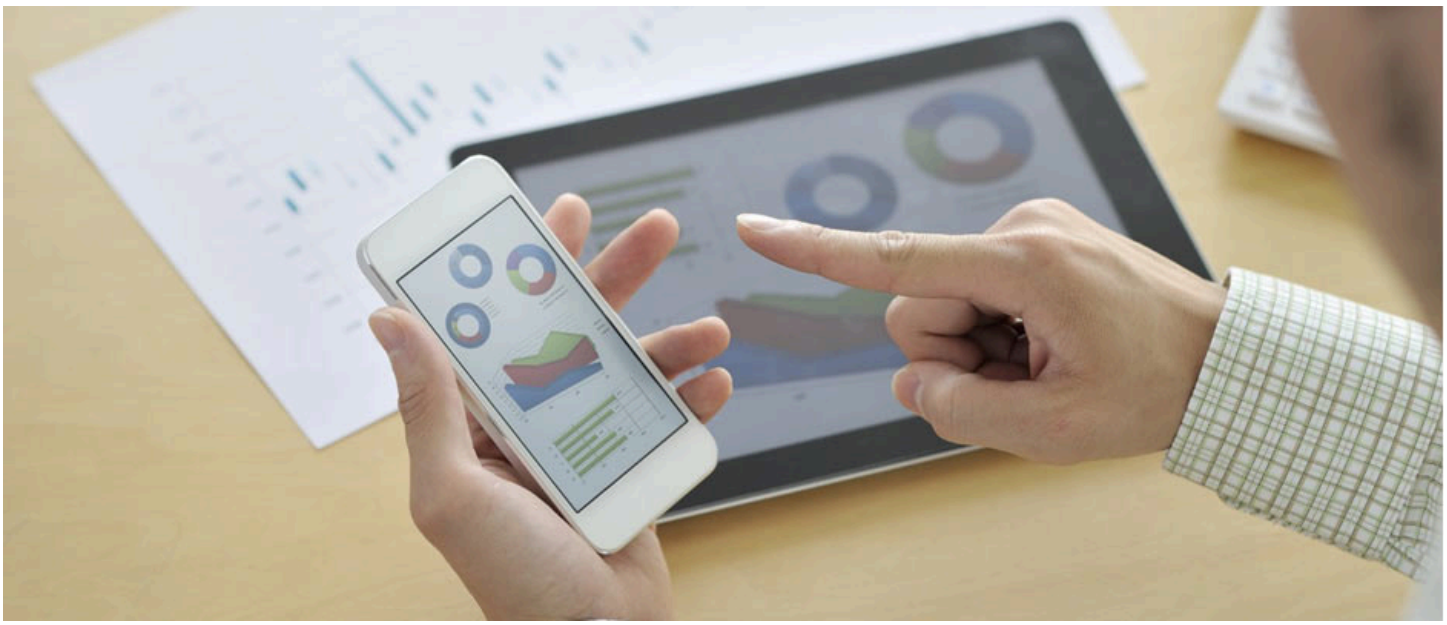
scientiamobile

*"The first step in a great mobile experience"*



# API DOCUMENTATION

## 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

# API Documentation

The `wurfl.h` interface gives you access to the main WURFL functionalities. Listed below is a quick overview of the functions, along with a few example on how to correctly use them.

> *Note: As of WURFL 1.5 the functions* `get_capability` *and* `get_virtual_capability` *return a zero-terminated string.*

# Typedef Documentation:

**typedef struct wurfl_device_capability_enumerator_tag* wurfl_device_capability_enumerator_handle**

**Description**:

WURFL Device capability enumerator handle, used to enumerate capabilities in a specific WURFL device.

**typedef struct wurfl_device_tag* wurfl_device_handle**

WURFL Device handle, used to represent an instance of WURFL device.

**typedef struct wurfl_tag* wurfl_handle**

**Description**:

WURFL handle, used to represent an instance of WURFL database.

# Enumeration Type Documentation:

### enum wurfl_cache_provider

**Enumerator**:

`WURFL_CACHE_PROVIDER_NONE` : do not use caching mechanisms.

`WURFL_CACHE_PROVIDER_LRU` : use the least-recently-used cache mechanism.

### enum wurfl_error

**Enumerator**:

`WURFL_OK` : no error.

`WURFL_ERROR_INVALID_HANDLE` : handle passed to the function is invalid.

`WURFL_ERROR_ALREADY_LOAD` : wurfl_load has already been invoked on the specific wurfl_handle.

`WURFL_ERROR_FILE_NOT_FOUND` : file not found during wurfl_load or remote data file update.

`WURFL_ERROR_UNEXPECTED_END_OF_FILE` : unexpected end of file or parsing error during wurfl_load.

`WURFL_ERROR_INPUT_OUTPUT_FAILURE` : error reading stream during wurfl_load or updater accessing local updated data file.

`WURFL_ERROR_DEVICE_NOT_FOUND` : specified device is missing.

`WURFL_ERROR_CAPABILITY_NOT_FOUND` : specified capability is missing.

`WURFL_ERROR_INVALID_CAPABILITY_VALUE` : invalid capability value.

`WURFL_ERROR_VIRTUAL_CAPABILITY_NOT_FOUND` : specified virtual capability is missing.

`WURFL_ERROR_CANT_LOAD_CAPABILITY_NOT_FOUND` : specified capability is missing.

`WURFL_ERROR_CANT_LOAD_VIRTUAL_CAPABILITY_NOT_FOUND` : specified virtual capability is missing.

`WURFL_ERROR_EMPTY_ID` : missing id in searching device.

`WURFL_ERROR_CAPABILITY_GROUP_NOT_FOUND` : specified capability is missing in its group.

`WURFL_ERROR_CAPABILITY_GROUP_MISMATCH` : specified capability mismatch in its group.

`WURFL_ERROR_DEVICE_ALREADY_DEFINED` : specified device is already defined.

`WURFL_ERROR_USERAGENT_ALREADY_DEFINED` : specified user agent is already defined.

`WURFL_ERROR_DEVICE_HIERARCHY_CIRCULAR_REFERENCE` : circular reference in device hierarchy.

`WURFL_ERROR_UNKNOWN` : unknown error.

`WURFL_ERROR_INVALID_USERAGENT_PRIORITY` : specified override sideloaded browser user agent configuration not valid.

`WURFL_ERROR_INVALID_PARAMETER` : invalid parameter.

`WURFL_ERROR_INVALID_CACHE_SIZE` : specified an invalid cache size, 0 or a negative value.

`WURFL_ERROR_XML_CONSISTENCY` : WURFL data file is out of date or wrong - some needed device_id/capability is missing.

`WURFL_ERROR_INTERNAL` : internal error. If this is an updater issue, please enable and check updater log using wurfl_updater_set_log_path().

`WURFL_ERROR_VIRTUAL_CAPABILITY_NOT_AVAILABLE` : the requested virtual capability has not been licensed.

`WURFL_ERROR_MISSING_USERAGENT` : an XML device definition without mandatory UA has been detected.

`WURFL_ERROR_XML_PARSE` : the XML data file is malformed.

`WURFL_ERROR_UPDATER_INVALID_DATA_URL` : updater data URL is missing or invalid (note: only .zip and .gz formats allowed)

`WURFL_ERROR_UPDATER_INVALID_LICENSE` : client license is invalid, expired etc.

`WURFL_ERROR_UPDATER_NETWORK_ERROR` : updater request returned an HTTP response != 200, or SSL error, etc. Please enable and check updater log using wurfl_updater_set_log_path().

`WURFL_ERROR_ENGINE_NOT_INITIALIZED` : prerequisite for executing an update is that the engine has been initialized (i.e., wurfl_load() has been called)

`WURFL_ERROR_UPDATER_ALREADY_RUNNING` : wurfl_updater_start() can be called just once, when the updater is not running.

`WURFL_ERROR_UPDATER_NOT_RUNNING` : wurfl_updater_stop() can be called just once, when the updater is running.

`WURFL_ERROR_UPDATER_TOO_MANY_REQUESTS` : Updater encountered HTTP 429 error.

`WURFL_ERROR_UPDATER_CMDLINE_DOWNLOADER_UNAVAILABLE` : Curl executable not found. Please check path, etc

`WURFL_ERROR_UPDATER_TIMEDOUT` : Curl operation timed out.

`WURFL_ERROR_ROOT_NOT_SET` : set_root() must be called before any load() / reload() and update attempt.

`WURFL_ERROR_WRONG_ENGINE_TARGET` : set_engine_target() was called with a wrong/unrecognized parameter

### enum wurfl_updater_frequency

**Enumerator**:

`WURFL_UPDATER_FREQ_DAILY` : check for any updated wurfl.zip every day.

`WURFL_UPDATER_FREQ_WEEKLY` : check for any updated wurfl.zip once a week.

# Configuration Methods:

### wurfl_error wurfl_add_patch(wurfl_handle hwurfl, const char* patch)

**Availability**:

1.4

**Description**:

Adds a WURFL patch to the patch collection.

**Parameters**:

`hwurfl`: The WURFL instance to add the patch to.

`patch`: The patch filename as a zero terminated ASCII string.

**Returns**:

`WURFL_OK`: If the function terminated correctly.

`wurfl_error`: If the function failed.

### wurfl_error wurfl_add_requested_capability(wurfl_handle hwurfl, const char* requested_capability)

**Availability**:

1.4

**Description**:

Adds a capability to the requested capabilities collection. If this function is never called on a `wurfl_handle`, the `wurfl_load()` function will automatically load all the features in the WURFL repository. If one or more of the capabilities are added to the requested capabilities collection, only the specified ones (if available) will be loaded from the repository, resulting in a reduced memory footprint.

**Parameters**:

`hwurfl`: The WURFL instance.

`requested_capability`: The capability name as a zero terminated ASCII string.

**Returns**:

`WURFL_OK`: If the function terminated correctly, or a different.

`wurfl_error`: If the function failed.

### wurfl_error wurfl_set_cache_provider(wurfl_handle hwurfl, wurfl_cache_provider cache_provider, const char* config)

**Availability**:

1.4

**Description**:

Sets the cache provider. If not called, a default `WURFL_CACHE_PROVIDER_LRU` of size 30000 is used.

**Parameters**:

`hwurfl`: The WURFL instance.

`cache_provider`: The cache provider.

`config`: The cache configuration as a zero-terminated ASCII string representing the desired size (like "100000"), or a null pointer or empty string if the specified cache doesn't need any specific configuration.

**Returns**:

`WURFL_OK`: If the funciton terminated correctly, or another.

`wurfl_error`: If the function failed.

### wurfl_error wurfl_set_root(wurfl_handle hwurfl, const char* root)

**Availability**:

1.4

**Description**:

Sets the WURFL root.

**Parameters**:

`hwurfl`: The WURFL instance to set the root to.

`root`: The root filename as a zero terminated ASCII string

**Returns**

`WURFL_OK`: If the function terminated correctly.

`wurfl_error`: If the function failed.

### wurfl_error wurfl_load(wurfl_handle hwurfl)

**Availability**:

1.4

**Description**:

Performs WURFL root file and patch loading. Must be called after specifying root filename by calling `wurfl_set_root` to set the root file name, and optionally `wurfl_add_patch` and/or `wurfl_add_requested_capability` to respectively specify patches and requested capabilities (if no capability is requested, all capabilities from WURFL root file and patches are loaded).

**Parameters**:

`hwurfl`: The WURFL instance.

**Returns**:

`WURFL_OK`: If the function terminated correctly.

`wurfl_error`: If the function failed.

### const char* wurfl_get_last_load_time_as_string(wurfl_handle hwurfl)

**Availability**:

1.5.1

**Description**:

Returns a zero terminated ASCII string of the most recently successful WURFL load phase. `0` will be returned if WURFL has not been successfully initialized. The returned string has the following format: `Www Mmm dd hh:mm:ss yyyy` where `Www` is the weekday, `Mmm` the month (in letters), dd the day of the month, `hh:mm:ss` the time, and `yyyy` the year.

**Parameters**:

`hwurfl`: The WURFL instance to use.

**Returns**:

### wurfl_handle wurfl_create()

**Availability**:

1.4

**Description**:

Creates a new WURFL instance. All configuration settings must be set prior to calling this method.

**Returns**:

a valid `wurfl_handle` or a null pointer if the function failed.

### const char* wurfl_get_api_version()

**Availability**:

1.5.1

**Description**:

Returns a string representing the currently used Libwurfl API version.

### const char* wurfl_get_wurfl_info(wurfl_handle hwurfl)

**Availability**:

1.5.1

**Description**:

Returns a zero terminated ASCII string containing infos on WURFL root and the loaded patches, or 0 if WURFL has not been initialized yet.

**Parameters**:

`hwurfl`: The WURFL instance to use.

# WURFL Query Methods:

**wurfl_device_handle wurfl_lookup(wurfl_handle hwurfl, wurfl_header_retrieve_callback header_retrieve_callback, const void *header_retrieve_callback_data)**

**Availability**:

1.5.0

**Description**:

WURFL Device lookup function. You must provide a header retrieve function which returns a header value given a header name. If there are no headers with that name the header retrieve callback function should return 0. Otherwise, it should return the header value associated to that name.

You must also provide the header container in which all the headers are stored using the `header_retrieve_callback_data` parameter, so the header retrieve callback function will look into it when searching for a header. The `header_retrieve_callback` function will be called at least one time during the lookup phase, but it can be called multiple times to check for additional headers.

**Parameters**:

`wurfl_handle`: The WURFL instance.

`wurfl_header_retrieve_callback`: A callback for function each time a header is looked up.

`header_retrieve_callback_data`: A header container in which all headers are stored.

**Returns**:

A new WURFL device instance (`wurfl_device_handle`).

### wurfl_device_handle wurfl_lookup_useragent(wurfl_handle hwurfl, const char* useragent)

**Availability**:

1.5.0

**Description**:

User Agent lookup function.

**Parameters**:

`wurfl_handle`: The WURFL instance.

`useragent`: The User Agent to lookup as a zero terminated ASCIIZ string.

**Returns**:

A new WURFL device instance, or a null pointer if the function failed.

**Example:**

```
wurfl_device_handle hdevice = wurfl_lookup_useragent(hwurfl, "USERAGENT");

if (hdevice) {
        cout << wurfl_device_get_id(hdevice) << endl;
}
```

### const char* wurfl_device_get_id(wurfl_device_handle hwurfldevice)

**Availability**:

1.4

**Description**:

Returns the device id of this device

**Parameters**:

`hwurfldevice`: The WURFL device instance.

**Returns**:

The WURFL device id as a zero terminated ASCIIZ string, or a null pointer if the function failed.

### const char* wurfl_device_get_root_id(wurfl_device_handle hwurfldevice)

**Description**:

Retrieve the root device id of this device.

**Parameters**:

`hwurfldevice`: The WURFL device instance.

**Returns**:

The WURFL device root id as a zero terminated ASCIIZ string, or a null pointer if the function failed.

### const char* wurfl_device_get_useragent(wurfl_device_handle hwurfldevice)

**Availability**:

1.4

**Description**:

Retrieve the User Agent for a device (as found in the WURFL repository).

**Parameters**:

`hwurfldevice`: The WURFL device instance

**Returns**:

The matched User Agent (as contained in the WURFL repository) as a zero terminated ASCIIZ string, or a null pointer if the function failed.

### const char* wurfl_device_get_original_useragent(wurfl_device_handle hwurfldevice)

**Availability**:

1.5.1

**Description**:

Returns the original useragent of this device.

**Parameters**:

`wurfl_device_handle`: The WURFL device instance

**Returns**:

The original User Agent as a zero terminated ASCIIZ string, or a null pointer if the function failed.

### const char* wurfl_device_get_normalized_useragent(wurfl_device_handle hwurfldevice)

**Availability**:

1.5.0

**Description**:

Returns the normalized useragent of this device as a zero terminated ASCIIZ string, or a null pointer if the function failed.

**Parameters**:

`hwurfldevice`: The WURFL device instance.

**Returns**:

Returns the normalized useragent of this device as a zero terminated ASCIIZ string, or a null pointer if the function failed

### wurfl_device_handle wurfl_get_device(wurfl_handle hwurfl, const char* deviceid)

**Availability**:

1.4

**Description**:

Retrieve a device by device id.

**Parameters**:

`wurfl_handle`: The WURFL instance.

`deviceid`: The wurfl device id to lookup as a zero terminated ASCIIZ string.

**Returns**:

A new WURFL device instance, or a null pointer if the function failed.

### void wurfl_device_destroy(wurfl_device_handle hwurfldevice)

**Availability**:

1.4

**Description**:

Releases a `wurfl_device_handle`. All `wurfl_device_capability_enumerator_handle`'s relative to this device handle must be released by calling `wurfl_device_capability_enumerator_destroy` on them before calling this function.

**Parameters**:

`hwurfldevice`: The WURFL device instance to destroy.

### void wurfl_destroy(wurfl_handle hwurfl)

**Availability**:

1.4

**Description**:

Releases a `wurfl_handle`. All `wurfl_device_handles` relative to this handle must be released by calling `wurfl_device_destroy` on them before calling this function.

**Parameters**:

`wurfl_handle`: The WURFL instance to destroy.

### const char* wurfl_device_get_capability(wurfl_device_handle hwurfldevice, const char* capability)

**Description**:

Retrieves a capability value from a device.

**Parameters**:

`hwurfldevice`: The WURFL device instance.

`capability`: The capability name as a zero terminated ASCIIZ string.

**Returns**:

The capability value as a zero terminated ASCIIZ string, or a null pointer if the function failed.

### const char* wurfl_device_get_virtual_capability(wurfl_device_handle hwurfldevice, const char* capability)

**Description**:

Retrieves a virtual capability value from a device.

**Parameters**:

`hwurfldevice`: The WURFL device instance.

`capability`: The virtual capability name as a zero terminated ASCIIZ string.

**Returns**:

The virtual capability value as a zero terminated ASCIIZ string, or a null pointer if the function failed.

### int wurfl_device_get_capability_as_int(wurfl_device_handle hwurfldevice, const char *capability)

**Availability**:

1.5.0

**Description**:

Retrieves a capability value as an integer value from a device.

**Parameters**:

`wurfl_device_handle`: The WURFL device instance.

`capability`: The capability name as a zero terminated ASCIIZ string.

**Returns**:

Returns the capability value as an integer.

**Example**:

```
wurfl_device_handle hdevice = wurfl_lookup_useragent(hwurfl, "USERAGENT");

if (hdevice) {

        int resolutionWidth = wurfl_device_get_capability_as_int(hdevice,
"resolution_width");

        if (resolutionWidth == 800) {
                cout << "Do something useful for device with a 800 width
resolution" << endl;
        }
        else if (resolutionWidth == 1200) {
                cout << "Do something useful for device with a 1200 width
resolution" << endl;
        }
}
```

### int wurfl_device_get_virtual_capability_as_int(wurfl_device_handle hwurfldevice, const char *virtual_capability)

**Availability**:

1.5.0

**Description**:

Retrieves a virtual capability value as an integer value from a device.

**Parameters**:

`wurfl_device_handle`: The WURFL device instance.

`virtual_capability`: The virtual capability name as a zero terminated ASCIIZ string.

**Returns**:

Returns the virtual capability value as an integer.

**Example**:

```
wurfl_device_handle hdevice = wurfl_lookup_useragent(hwurfl, "USERAGENT");

if (hdevice) {

        int browserVersion = wurfl_device_get_virtual_capability_as_int(hdevice,
"advertised_browser_version");

        if (browserVersion == 26) {
                cout << "Use special layout" << endl;
        }
        else {
                cout << "Use common layout" << endl;
        }
}
```

### int wurfl_device_get_capability_as_bool(wurfl_device_handle hwurfldevice, const char *capability)

**Availability**:

1.5.0

**Description**:

Retrieves a capability value as a boolean value from a device.

**Parameters**:

`wurfl_device_handle`: The WURFL device instance.

`capability`: The capability name as a zero terminated ASCIIZ string.

**Returns**:

Returns the capability value as a boolean.

**Example**:

```
wurfl_device_handle hdevice = wurfl_lookup_useragent(hwurfl, "USERAGENT");

if (hdevice) {

        bool isWireless = wurfl_device_get_capability_as_bool(hdevice,
"is_wireless_device");

        if (isWireless) {
                cout << "Do something useful for a wireless device" << endl;
        }
        else {
                cout << "Do something useful for a non wireless device" << endl;
        }
}
```

### int wurfl_device_get_virtual_capability_as_bool(wurfl_device_handle hwurfldevice, const char *virtual_capability)

**Availability**:

1.5.0

**Description**:

Retrieves a virtual capability value as a boolean value from a device.

**Parameters**:

`wurfl_device_handle`: The WURFL device instance.

`virtual_capability`: The virtual capability name as a zero terminated ASCIIZ string.

**Returns**:

Returns the virtual capability value as a boolean.

**Example**:

```
wurfl_device_handle hdevice = wurfl_lookup_useragent(hwurfl, "USERAGENT");

if (hdevice) {

        bool isAndroid = wurfl_device_get_virtual_capability_as_bool(hdevice,
"is_android");

        if (isAndroid) {
                cout << "Do something useful for Android devices" << endl;
        }
        else {
                cout << "Do something useful for non Android devices" << endl;
        }
}
```

### int wurfl_device_is_actual_device_root(wurfl_device_handle hwurfldevice)

**Description**:

Verifies if the device is an actual device root.

**Parameters**

`hwurfldevice`: The WURFL device instance.

**Returns**:

`1` if the specified device is an actual device root, `0` otherwise.

### int wurfl_device_has_capability(wurfl_device_handle hwurfldevice, const char* capability)

**Availability**:

1.5.0

**Description**:

Verifies if the device has a specific capability.

**Parameters**:

`hwurfldevice`: The WURFL device instance. `capability`: The capability name as a zero terminated ASCIIZ string.

**Returns**:

`1` if the specified device has the requested capability, `0` otherwise.

**Example**:

```
wurfl_device_handle hdevice = wurfl_lookup_useragent(hwurfl, "USERAGENT");

if (hdevice && wurfl_device_has_capability(hdevice, "requested_capability")) {
        cout << "Requested capability has been found" << endl;
}
```

### int wurfl_device_has_virtual_capability(wurfl_device_handle hwurfldevice, const char* capability)

**Availability**:

1.5.0

**Description**:

Verifies if the device has a specific virtual capability.

**Parameters**:

`hwurfldevice`: The WURFL device instance.

`capability`: The virtual capability name as a zero terminated ASCIIZ string.

**Returns**:

`1` if the specified device has the requested virtual capability, `0` otherwise.

**Example**:

```
wurfl_device_handle hdevice = wurfl_lookup_useragent(hwurfl, "USERAGENT");

if (hdevice && wurfl_device_has_capability(hdevice,
"requested_virtual_capability")) {
        cout << "Requested virtual capability has been found" << endl;
}
```

### int wurfl_has_capability(wurfl_handle hwurfl, const char *capability)

**Availability**:

1.5.3

**Description**:

Verify if a capability is a valid wurfl capability.

**Parameters**:

`wurfl_handle`: The WURFL instance to use. `capability`: The capility to verify.

**Returns**:

`1` if the capility exists, otherwise `0`.

### int wurfl_has_virtual_capability(wurfl_handle hwurfl, const char *virtual_capability)

**Availability**:

1.5.3

**Description**:

Verify if a virtual capability is a valid wurfl virtual capability.

**Parameters**:

`wurfl_handle`: The WURFL instance to use. `virtual_capability`: The capility to verify.

**Returns**:

`1` if the capility exists, otherwise `0`.

## WURFL Device Methods:

### wurfl_capability_enumerator_handle wurfl_get_capability_enumerator(wurfl_handle hwurfl)

**Availability**:

1.5.1

**Description**:

Returns a new WURFL capability enumerator instance.

**Parameters**:

`wurfl_handle`: The WURFL handle instance to use.

**Example**:

```
wurfl_handle hwurfl = [...];

wurfl_capability_enumerator_handle hcapabilityenumerator =
wurfl_get_capability_enumerator(hwurfl);

while (wurfl_capability_enumerator_is_valid(hcapabilityenumerator)) {
        const char *name =
wurfl_capability_enumerator_get_name(hcapabilityenumerator);

        // do something with name

        wurfl_capability_enumerator_move_next(hcapabilityenumerator);
}

wurfl_capability_enumerator_destroy(hcapabilityenumerator);
```

### wurfl_capability_enumerator_handle wurfl_get_virtual_capability_enumerator(wurfl_handle hwurfl)

**Availability**:

1.5.1

**Description**:

Returns a new WURFL virtual capability enumerator instance

**Parameters**:

`wurfl_handle`: The WURFL handle instance to use.

**Returns**:

`wurfl_capability_enumerator_handle`: A new capability enumerator handle.

**Example**:

```
wurfl_handle hwurfl = [...];
wurfl_capability_enumerator_handle hvirtualcapabilityenumerator =
wurfl_get_virtual_capability_enumerator(hwurfl);

while (wurfl_capability_enumerator_is_valid(hvirtualcapabilityenumerator)) {

        const char *name =
wurfl_capability_enumerator_get_name(hvirtualcapabilityenumerator);

        // do something with name

        wurfl_capability_enumerator_move_next(hvirtualcapabilityenumerator);
}

wurfl_capability_enumerator_destroy(hvirtualcapabilityenumerator);
```

### void wurfl_capability_enumerator_destroy(wurfl_capability_enumerator_handle hwurflcapabilityenumeratorhandle)

**Availability**:

1.5.1

**Description**:

Releases a `wurfl_capability_enumerator_handle` instance.

**Parameters**:

`wurfl_capability_enumerator_handle`: The capability enumerator instance to destroy.

### int wurfl_capability_enumerator_is_valid(wurfl_capability_enumerator_handle hwurflcapabilityenumeratorhandle)

**Availability**:

1.5.1

**Description**:

Verifies if the WURFL enumerator instance is still valid.

**Parameters**:

`wurfl_capability_enumerator_handle`: The WURFL enumerator handler instance to check.

**Returns**:

`1` if the specified device is valid, `0` otherwise.

### void wurfl_capability_enumerator_move_next(wurfl_capability_enumerator_handle hwurflcapabilityenumeratorhandle)

**Availability**:

1.5.1

**Description**:

Moves the WURFL enumerator instance to the next capability

**Parameters**:

`wurfl_capability_enumerator_handle`: The WURFL enumerator handler instance to check.

### const char* wurfl_capability_enumerator_get_name(wurfl_capability_enumerator_handle hwurflcapabilityenumeratorhandle)

**Availability**:

1.5.1

**Description**:

Returns the name of the current capability as a string.

**Parameters**:

`wurfl_capability_enumerator_handle`: The WURFL enumerator handler instance to check.

**Returns**:

Returns the name of the current capability as an ASCIIZ string, or a null pointer if the function failed.

### wurfl_device_id_enumerator_handle wurfl_get_device_id_enumerator(wurfl_handle hwurfl)

**Availability**:

1.5.2

**Description**:

Returns a new WURFL device id enumerator instance.

**Parameters**:

`wurfl_handle`: The WURFL instance.

**Example:**

```
wurfl_handle hwurfl = [...];
wurfl_device_id_enumerator_handle hdeviceidenumerator =
wurfl_get_device_id_enumerator(hwurfl);

while (wurfl_device_id_enumerator_is_valid(hdeviceidenumerator)) {
        const char *name =
wurfl_device_id_enumerator_get_device_id(hdeviceidenumerator);

        // do something with name

        wurfl_device_id_enumerator_move_next(hdeviceidenumerator);
}

wurfl_device_id_enumerator_destroy(hdeviceidenumerator);
```

### int wurfl_device_id_enumerator_is_valid(wurfl_device_id_enumerator_handle hdeviceidenumerator)

**Availability**:

1.5.2

**Description**:

Verifies if the WURFL enumerator instance is still valid

**Parameters**:

`wurfl_device_id_enumerator_handle`: The WURFL device id enumerator to verify.

**Returns**:

`1` if the enumerator is valid, otherwise `0`.

### void wurfl_device_id_enumerator_move_next(wurfl_device_id_enumerator_handle hdeviceidenumerator)

**Availability**:

1.5.2

**Description**:

Moves the WURFL enumerator instance to the next capability.

**Parameters**:

`wurfl_device_id_enumerator_handle`: The WURFL device id enumerator to use.

### void wurfl_device_id_enumerator_destroy(wurfl_device_id_enumerator_handle hdeviceidenumerator)

**Availability**:

1.5.2

**Description**:

Releases a wurfl_device_id_enumerator_handle instance.

**Parameters**:

`wurfl_device_id_enumerator_handle`: The WURFL device id enumerator to use.

### const char* wurfl_device_id_enumerator_get_device_id(wurfl_device_id_enumerator_handle hdeviceidenumerator)

**Availability**:

1.5.2

**Description**:

Returns the name of the current capability as an ASCIIZ string, or a null pointer if the function failed.

**Parameters**:

`wurfl_device_id_enumerator_handle`: The WURFL device id enumerator to use.

**Returns**:

The current capability as an ASCIIZ string, or a null pointer if the function failed.

### void wurfl_device_capability_enumerator_destroy(wurfl_device_capability_enumerator_handle enumerator_handle)

**Availability**:

1.5.0

**Description**:

Releases a `wurfl_device_capability_enumerator_handle` (see `wurfl_device_get_capability_enumerator`).

**Parameters**:

`hwurfl_device_capability_enumerator_handle`: The enumerator instance to be released.

**Example:**

```
if (hdevice) {
        const char *capabilityValue = wurfl_device_get_capability(hdevice,
"brand_name");

        if (capabilityValue == "Nokia") {
                cout << "Do something useful for Nokia device" << endl;
        }
        else if (capabilityValue == "HTC") {
                cout << "Do something useful for HTC device" << endl;
        }
}
```

### const char* wurfl_device_capability_enumerator_get_name(wurfl_device_capability_enumerator_handle)

**Availability**:

1.5.0

**Description**:

The current capability name (see `wurfl_device_get_capability_enumerator`).

**Parameters**:

`hwurfl_device_capability_enumerator_handle`: The enumerator instance.

**Returns**:

The name of the current capability as an ASCIIZ string, or a null pointer if the function failed.

### const char* wurfl_device_capability_enumerator_get_value(wurfl_device_capability_enumerator_handle)

**Availability**:

1.5.0

**Description**:

The current capability value (see `wurfl_device_get_capability_enumerator`).

**Parameters**:

`hwurfl_device_capability_enumerator_handle`: The enumerator instance.

**Returns**:

The value of the current capability as an ASCIIZ string, or a null pointer if the function failed.

### int wurfl_device_capability_enumerator_get_value_as_int(wurfl_device_capability_enumerator_handle enumerator_handle)

**Availability**:

1.5.0

**Description**:

The current capability value as an integer (see `wurfl_device_get_capability_enumerator`).

**Parameters**:

`hwurfl_device_capability_enumerator_handle`: The enumerator instance.

**Returns**:

The value of the current capability value as an integer, or a null pointer if the function failed.

### int wurfl_device_capability_enumerator_get_value_as_bool(wurfl_device_capability_enumerator_handle enumerator_handle)

**Availability**:

1.5.0

**Description**:

The current capability value as an boolean value (see `wurfl_device_get_capability_enumerator`).

**Parameters**:

`hwurfl_device_capability_enumerator_handle`: The enumerator instance.

**Returns**:

The value of the current capability value as an boolean value, or a null pointer if the function failed.

### int wurfl_device_capability_enumerator_is_valid(wurfl_device_capability_enumerator_handle)

**Availability**:

1.5.0

**Description**:

Verifies if the enumerator is still valid (see `wurfl_device_get_capability_enumerator`).

**Parameters**:

`hwurfl_device_capability_enumerator_handle`: The enumerator instance.

**Returns**:

`1` if the enumerator is valid, `0` otherwise.

### void wurfl_device_capability_enumerator_move_next(wurfl_device_capability_enumerator_handle)

**Availability**:

1.5.0

**Description**:

Moves the enumerator to the next capability (see `wurfl_device_get_capability_enumerator`).

**Parameters**:

`hwurfl_device_capability_enumerator_handle`: The enumerator instance.

**wurfl_device_capability_enumerator_handle**
**wurfl_device_get_capability_enumerator(wurfl_device_handle hwurfldevice)**

**Availability**:

1.5.0

**Description**:

Creates a new capability enumerator for a device

**Parameters**:

`hwurfldevice`: The WURFL device instance.

**Returns**:

A new WURFL capability enumerator instance.

**Example**:

```
wurfl_device_handle hdevice = wurfl_lookup_useragent(hwurfl, "USERAGENT");

if (hdevice) {
        wurfl_device_capability_enumerator_handle hdevicecaps =
wurfl_device_get_capability_enumerator(hdevice);

        while (wurfl_device_capability_enumerator_is_valid(hdevicecaps)) {
                cout << "   " <<
wurfl_device_capability_enumerator_get_name(hdevicecaps) << "=" <<
wurfl_device_capability_enumerator_get_value(hdevicecaps) << endl;
                wurfl_device_capability_enumerator_move_next(hdevicecaps);
        }
        wurfl_device_capability_enumerator_destroy(hdevicecaps);
}
```

**wurfl_device_capability_enumerator_handle**
**wurfl_device_get_virtual_capability_enumerator(wurfl_device_handle hwurfldevice)**

**Availability**:

1.5.0

**Description**:

Creates a new virtual capability enumerator for a device.

**Parameters**:

`hwurfldevice`: The WURFL device instance.

**Returns**:

A new WURFL virtual capability enumerator instance.

**Example**:

```
wurfl_device_handle hdevice = wurfl_lookup_useragent(hwurfl, "USERAGENT");

if (hdevice) {
        wurfl_device_capability_enumerator_handle hdevicecaps =
wurfl_device_get_virtual_capability_enumerator(hdevice);

        while (wurfl_device_capability_enumerator_is_valid(hdevicecaps)) {
                cout << "   " <<
wurfl_device_capability_enumerator_get_name(hdevicecaps) << "=" <<
wurfl_device_capability_enumerator_get_value(hdevicecaps) << endl;
                wurfl_device_capability_enumerator_move_next(hdevicecaps);
        }

        wurfl_device_capability_enumerator_destroy(hdevicecaps);
}
```

# WURFL Updater Methods:

### wurfl_error wurfl_updater_set_log_path(wurfl_handle hwurfl, const char* log_path)

**Availability**:

1.8.3.0

**Description**:

Instructs the internal WURFL InFuze updater to log to file any operation/error. This is an optional call: if not used, the updater will not log anything.

**Parameters**:

`wurfl_handle`: The WURFL instance.

`log_path`: the path where the updater log file is created.

**Returns**:

`WURFL_OK`: If the function terminated correctly.

`wurfl_error`: If the function failed.

### wurfl_error wurfl_updater_set_data_url(wurfl_handle hwurfl, const char* data_url)

**Availability**:

1.8.3.0

**Description**:

Set remote data URL to be downloaded via internal WURFL InFuze updater. This is a MANDATORY call if you want to use the InFuze Updater.

**Parameters**:

`wurfl_handle`: The WURFL instance.

`data_url`: the ScientiaMobile updater URL where to seek for updated WURFL data file, in the form "https://data.scientiamobile.com/xxxxx/wurfl.zip". "xxxxx" must be replaced with your personal access token, which is shown on the **Snapshot generator** page of your license in the Customer Vault.

**Notes**

- The WURFL data file and the path where it resides, specified in the `setRoot()` call, MUST have write/rename access: the old data file will be replaced (i.e, a rename operation will be performed) with his updated version upon successful update operation completion, and the directory will be used for remote file download, etc.
- ScientiaMobile does not distribute uncompressed XML data files via updater. This means that, if you plan to use the updater feature, you MUST use a compressed (i.e, a ZIP or a XML.GZ) file as data root in the setRoot() call.
- The WURFL InFuze Updater functionality relies on availability and features of the well-known and widely available `curl` command-line utility. Among others, also a check for `curl` availability is done in the `wurfl_updater_set_data_url()` call.

**Returns**:

`WURFL_OK`: If the function terminated correctly.

`wurfl_error`: If the function failed.

### wurfl_error wurfl_updater_set_data_frequency(wurfl_handle hwurfl, wurfl_updater_frequency freq)

**Availability**:

1.8.3.0

**Description**:

Sets how often the updater checks for any new/updated WURFL data file to be downloaded and used by the engine.

**Parameters**:

`wurfl_handle`: The WURFL instance.

`freq`: the frequency of online checks, one of `WURFL_UPDATER_FREQ_DAILY` (default) or `WURFL_UPDATER_FREQ_WEEKLY`

**Notes**

`wurfl_updater_set_data_frequency()` sets how often the updater checks for any updated data file, not how often the engine data file is actually updated.

**Returns**:

`WURFL_OK`: If the function terminated correctly.

`wurfl_error`: If the function failed.

### wurfl_error wurfl_updater_set_data_url_timeouts(wurfl_handle hwurfl, int connection_timeout, int data_transfer_timeout)

**Availability**:

1.8.3.0

**Description**:

Set internal WURFL InFuze Updater timeouts, in milliseconds.

**Parameters**:

`wurfl_handle`: The WURFL instance.

`connection_timeout`: timeout for the connection phase of updater data file download operation

`data_transfer_timeout`: timeout for the data transfer phase of updater data file download operation

**Notes**

The values are mapped to `curl` `--connect-timeout` and `--max-time` parameters (after millisecs-to-secs conversion). Connection timeout has a WURFL InFuze default value of 10 seconds (10000 ms) and refers only to connection phase. Passing 0 will use `curl` value "no timeout used". Data transfer timeout has a InFuze default value of 600 seconds (600000 ms). Passing 0 will use `curl` default value "no timeout used". So, pass 0 to either parameter to invoke `curl` "no timeout used" behaviour. Pass -1 to either parameter to use WURFL InFuze default values (10 secs, 600 secs). The specified timeouts (if any) are only used in the synchronous (i.e., `runonce_updater()`) API call. The asynchronous background updater invoked by `start_updater()`/`stop_updater()` always runs with `curl` behaviour and timeouts (i.e., it will wait "as long as needed" for a new data file to be downloaded)

**Returns**:

`WURFL_OK`: If the function terminated correctly.

`wurfl_error`: If the function failed.

### wurfl_error wurfl_updater_reload_root(wurfl_handle hwurfl, const char* newroot)

**Availability**:

1.8.3.0

**Description**:

Reload a given WURFL data file, rebooting engine and overwriting the old one.

**Parameters**:

`wurfl_handle`: The WURFL instance.

`newroot` : the WURFL data file to reload.

**Returns**:

`WURFL_OK`: If the function terminated correctly.

`wurfl_error`: If the function failed.

### wurfl_error wurfl_updater_runonce(wurfl_handle hwurfl)

**Availability**:

1.8.3.0

**Description**:

Call a WURFL InFuze synchronous update.

**Parameters**:

`wurfl_handle`: The WURFL instance.

**Returns**:

`WURFL_OK`: If the function terminated correctly.

`wurfl_error`: If the function failed.

### wurfl_error wurfl_updater_start(wurfl_handle hwurfl)

**Availability**:

1.8.3.0

**Description**:

Starts the asynchronous WURFL InFuze background update thread.

**Parameters**:

`wurfl_handle`: The WURFL instance.

**Returns**:

`WURFL_OK`: If the function terminated correctly.

`wurfl_error`: If the function failed.

### wurfl_error wurfl_updater_stop(wurfl_handle hwurfl)

**Availability**:

1.8.3.0

**Description**:

Stops the asynchronous WURFL InFuze background update thread.

**Parameters**:

`wurfl_handle`: The WURFL instance.

**Returns**:

`WURFL_OK`: If the function terminated correctly.

`wurfl_error`: If the function failed.

# Error Handling:

### void wurfl_has_error_message(wurfl_handle hwurfl)

**Availability**:

1.5.0

**Description**:

Indicates if the last call to the WURFL library on a specific `wurfl_handl` instance was successful.

**Parameters**:

`wurfl_handle`: The WURFL instance.

**Example:**

```
wurfl_add_requested_capability(hwurfl, "unavailable_capability");

if (wurfl_has_error_message(hwurfl)) {
        cout << wurfl_get_error_message(hwurfl) << endl;
}
```

### const char* wurfl_get_error_message(wurfl_handle hwurfl)

**Availability**:

1.5.0

**Description**:

Returns the last error message (if any) produced by a call to the WURFL library on a specific `wurfl_handle` instance.

**Parameters**:

`wurfl_handle`: The WURFL instance.

**Returns**:

The last error message as a zero terminated ASCII string, or a null pointer if the function failed.

**Example:**

```
if (error != WURFL_OK) {
        cout << wurfl_get_error_message(hwurfl) << endl;
}
```

### void wurfl_clear_error_message(wurfl_handle hwurfl)

**Availability**:

1.5.0

**Description**:

Clears the last error message (if any) produced by the last call to the WURFL library on a specific `wurfl_handle` instance.

**Parameters**:

`wurfl_handle`: The WURFL instance.

**Example:**

```
wurfl_add_requested_capability(hwurfl, "unavailable_capability");

if (wurfl_has_error_message(hwurfl)) {
        cout << wurfl_get_error_message(hwurfl) << endl;
        wurfl_clear_error_message(hwurfl);
}
```