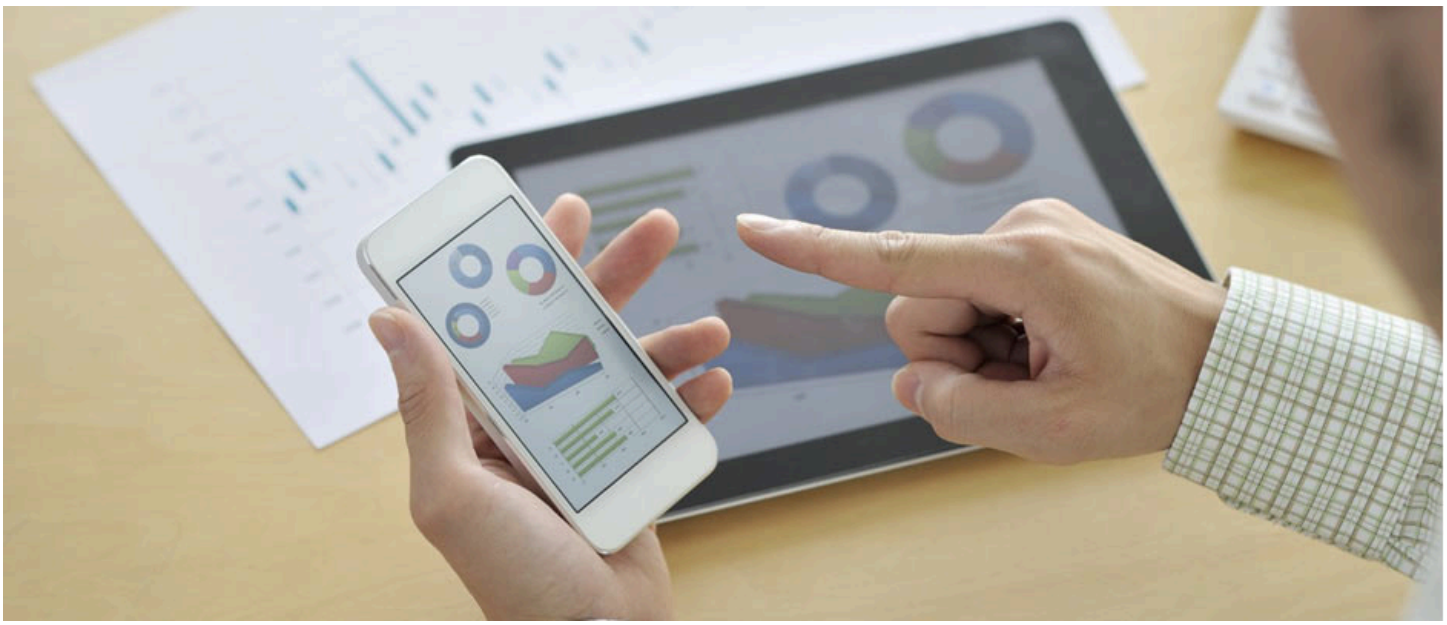
scientiamobile

"The first step in a great mobile experience"



WURFL MICROSERVICE CLIENT API FOR NODE.JS

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 Node.js

> ***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 (AWS, Azure, CGP) 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 access 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**.*

> ***IMPORTANT NOTE****: As of version 3.0, this library has adopted `async/await` pver the previous approach based on callbacks. Client API functions no longer require a callback function as one of the parameters. Consequently, applications relying on older WM client versions will need to be modified to use async/await or promises (see paragraph 'Using promises' later). For example, previous Client API versions that would get the WM server info with:*

```
client.getInfo(function (info, error) {
  // do something...
}
```

will now (version 3.0.0 on) to be modified as follows:

```
let info = await client.getInfo()
// do something
```

Obtaining, Installing and Running the `wmclient`

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

1. Obtain the package with the `wmclient` library from the **WURFL repository**, cloning it in your preferred location.

2. Install the wmclient package via npm:

```
cd src/app
npm install .
```

3. Edit `example.js` (in the package), modifying the `create()` call to include the public IP address of your AMI instance or Docker container service (e.g. replacing localhost with the IP address of your running AMI in AWS Marketplace, or the address of your 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.**

4. Run the example below:

```
let wmclient = require('wmclient')
const http = require("http")
const TEXT_SEPARATOR = '------------------------------------------------------------
--------------------'
separate = function () {console.log(TEXT_SEPARATOR)}

// Example of using the WURFL Microservice Client
const example = async () => {
    console.log('Running with node version ' + process.version)
    // First we need to create a WM client instance, to connect to our WM server API
at the specified host and port.
    let client
    try {
        client = await wmclient.create('http:', 'localhost', '8080', '')
    } catch (error) {
        console.log(`Error creating WURFL Microservice client: ${error.message}.
Terminating example`)
        process.exit(1)
    }
    console.log('wm created, printing some data')
    console.log('Static capabilities loaded: ' + client.staticCaps.length)
    console.log('Virtual capabilities loaded: ' + client.virtualCaps.length + '\n')
    separate()
    // Get server info
    try {
        let info = await client.getInfo()
        console.log('Server info: \n')
        console.log('WURFL API version: ' + info.wurflAPIVersion)
        console.log('WM server version: ' + info.wmVersion)
        console.log('WURFL file info:' + info.wurflInfo + '\n')
    } catch (error) {
        console.log("Unable to load WURFL Info")
    }
    separate()
    // Perform a detection using a whole HTTP request to WM server API
    // When building a request object for node, headers must be lowercase, according
to Node standard
    let req_headers = {
        'accept': 'text/html, application/xml;q=0.9, application/xhtml+xml,
image/png, image/webp, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1',
        'accept-encoding': 'gzip, deflate',
        'accept-language': 'en',
        'device-stock-ua': 'Mozilla/5.0 (Linux; Android 8.1.0; SM-J610G Build/M1AJQ;
wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/69.0.3497.100 Mobile
Safari/537.36',
        'forwarded': 'for=\"110.54.224.195:36350\"',
        'referer': 'https://www.cram.com/flashcards/labor-and-delivery-questions-
889210',
        'save-data': 'on',
```

```
        'user-agent': 'Opera/9.80 (Android; Opera Mini/51.0.2254/184.121; U; en)
Presto/2.12.423 Version/12.16',
        'x-clacks-overhead': 'GNU ph',
        'x-forwarded-for': '110.54.224.195, 82.145.210.235',
        'x-operamini-features': 'advanced, camera, download, file_system, folding,
httpping, pingback, routing, touch, viewport',
        'x-operamini-phone': 'Android #',
        'x-operamini-phone-Ua': 'Mozilla/5.0 (Linux; Android 8.1.0; SM-J610G
Build/M1AJQ; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0
Chrome/69.0.3497.100 Mobile Safari/537.36',
    }

    // Request refer a local WM server. Change the host and port to your WM server if
needed
    let options = {
        protocol: 'http:',
        host: 'localhost',
        port: '8080',
        method: 'POST',
        path: '/',
    }

    let req = http.request(options)
    req.headers = req_headers
    req.end()

    // Get device info using wmclient
    let device
    try {
        device = await client.lookupRequest(req)
    }
    catch(error) {
        console.log('Error detecting device from given headers:  ' + error.message)
    }
    console.log('WURFL device id ' + device.capabilities['wurfl_id'] + '\n')
    console.log('DEVICE BRAND & MODEL')
    console.log(device.capabilities['brand_name'] + ' ' +
device.capabilities['model_name'] + '\n')
    if (device.capabilities['is_smartphone'] === 'true') {
        console.log('This is a smartphone\n')
    }

    console.log('All received capabilities: \n')
    for (let key in device.capabilities) {
        if (device.capabilities.hasOwnProperty(key)) {
            console.log(key + ': ' + device.capabilities[key])
        }
    }
    separate()
    // Get some data using wmclient
```

```
// Get all the device manufacturers, and print the first twenty returned
let deviceMakes = await client.getAllDeviceMakes()
let limit = 20
console.log("Print the first " + limit + " Brand of " + deviceMakes.length)
// Sort the device manufacturer names
deviceMakes.sort()
for (let i = 0; i < limit; i++) {
    console.log(" - " + deviceMakes[i])
}
separate()
// Get all device model and marketing names produced by Apple
let brandName = "Apple"
try {
    let devsForMake = await client.getAllDevicesForMake(brandName)
    // Sort modelMktNames by their model name
    devsForMake.sort(compare)
    console.log("Print all Model for the Apple Brand")
    for (let i = 0; i < devsForMake.length; i++) {
        let n = " - " + devsForMake[i].modelName
        if (devsForMake[i].marketingName !== undefined) {
            n += devsForMake[i].marketingName
        }
        console.log(n)
    }
} catch (error) {
    console.log(`Error looking for  models for device, brand ${error.message}`)
}
separate()
// Get all operative system names
let oses = await client.getAllOSes()
// Sort and print all OS names
console.log("Print the list of OSes")
oses.sort()
for (let i = 0; i < oses.length; i++) {
    console.log(" - " + oses[i])
}

// Get all Android OS versions
let os = 'Android'
try {
    let versions = await client.getAllVersionsForOS(os)
    // Sort all os version numbers and print them.
    separate()
    console.log(`Print all versions for the ${os} OS`)
    versions.sort()
    for (var i = 0; i < versions.length; i++) {
        console.log(" - " + versions[i])
    }
}
```

```
    catch (error){
        console.log(`Error listing versions for ${os}  OS: ${error.message}`)
    }
}
// Run the example
example().then().catch()

// Comparator used to sort modelMktNames objects according to their model name
property, for which is used the String natural ordering.
function compare(a, b) {
    let comparison = 0
    if (a.modelName > b.modelName) {
        comparison = 1
    } else if (a.modelName < b.modelName) {
        comparison = -1
    }
    return comparison
}
```

6. 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 `lookupRequest` method as follows:

```
let device = await result.lookupUserAgent(req);
```

7. This will provide optimal detection accuracy (see `examples/web.php`). 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

- **JSONDeviceData:** models a WURFL device data in JSON format

```
new JSONDeviceData(capabilities, error, mtime, ltime)
```

- **JSONInfoData:** provides server and API information

```
new JSONInfoData(wurflAPIVersion, wurflInfo, wmVersion, importantHeaders, staticCaps,
virtualCaps, ltime)
```

- **JSONMakeModel:** models simple device "identity" data in JSON format

```
new JSONMakeModel(brandName, modelName, marketingName)
```

- **JSONModelMktName:** holds model_name and marketing_name data in JSON format

```
new JSONModelMktName(modelName, marketingName)
```

- **Request:** data object that is sent to the WM server in POST requests

```
new Request(lookupHeaders, requestedCaps, requestedVCaps, wurflID, tacCode)
```

- **WmClient:** holds HTTP connection data to the server and the list of capabilities it must return in response

```
new WmClient(scheme, host, port, baseURI)
```

- **create:** create a WmClient instance

```
create(scheme, host, port, baseURI) → {WmClient}
```

- **getAllDeviceMakes:** returns a list of all device manufacturers in WM server

```
getAllDeviceMakes() → {string[]}
```

- **getAllDevicesForMake:** returns a list of models and marketing names for a manufacturer

```
getAllDevicesForMake(make) → {JSONModelMktName[]}
```

- **getAllOSes:** returns a list of of all OS in WM server

```
getAllOSes() → {JSONModelMktName[]}
```

- **getAllVersionsForOS:** returns a list of of all versions for an OS

```
getAllVersionsForOS(device_os) → {string[]}
```

- **getApiVersion:** returns the API version

```
getApiVersion() → {string}
```

- **getCapabilityCount:** returns the number of capabilities for the given device

```
getCapabilityCount(device) → {integer}
```

- **getInfo:** returns information about the running WM server and API

```
getInfo() → {JSONInfoData}
```

- **hasStaticCapability:** returns true if the given capName exists in the client's static capability set, false otherwise

```
hasStaticCapability(capName) → {boolean}
```

- **hasVirtualCapability:** returns true if the given capName exists in the client's virtual capability set, false otherwise

```
hasVirtualCapability(vcapName) → {boolean}
```

- **lookupDeviceID:** searches WURFL device data using its wurfl_id value

```
lookupDeviceID(wurflId) → {JSONDeviceData}
```

- **lookupRequest:** detects a device and returns its data in JSON format

```
lookupRequest(nodeReq) → {JSONDeviceData}
```

- **lookupUserAgent:** searches WURFL device data using the given user-agent for detection

```
lookupUserAgent(userAgent) → {JSONDeviceData}
```

- **setCacheSize:** sets the UA cache size

```
setCacheSize(uaMaxEntries)
```

- **SetHTTPTimeout:** sets the connection and transfer timeouts for the client in seconds - should be called before performing any connection to WM server

```
setHTTPTimeout(timeout)
```

- **setRequestedStaticCapabilities:** list of static capabilities to be requested from the WM server

```
setRequestedStaticCapabilities(stCaps)
```

- **setRequestedVirtualCapabilities:** list of virtual capabilities to be requested from the WM server

```
setRequestedVirtualCapabilities(vCaps)
```

Error in case of Missing Capabilities

Depending on the configuration of your WURFL Microservice, one or more WURFL capabilities may be unavailable. In this case, the `JSONDeviceData.Capabilities` map will miss the entry for those capabilities and the caller will get a return value of zero for it (nil).

The availability of a capability can be checked with:

```
value, ok := JSONDeviceData.Capabilities["has_cellular_radio"]
```

`ok` is a boolean (`true`/`false`).

Additional capabilities can be obtained by upgrading to a greater version of the WM Server in the respective marketplaces or by contacting ScientiaMobile to license additional capabilities for the Docker Image or other products.

Using Promises

The following examples illustrate how you can use `Promises` instead of `async/await`:

```
const wmclient = require('wmclient')
const clientPromise = wmclient.create('http:', 'localhost','8080','')
clientPromise.then((client) => {
    let userAgent = 'Mozilla/5.0 (Linux; Android 10; GM1911 Build/QKQ1.190716.003;
wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/83.0.4103.106 Mobile
Safari/537.36 Instagram 146.0.0.27.125 Android (29/10; 560dpi; 1440x3064; OnePlus;
GM1911; OnePlus7Pro; qcom; en_US; 221134037)'
    let device_promise = client.lookupUserAgent(userAgent)
    device_promise.then((device) => {
        console.log(`Detected device ${device.capabilities['brand_name']}
${device.capabilities['model_name']}`)
```

© 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.