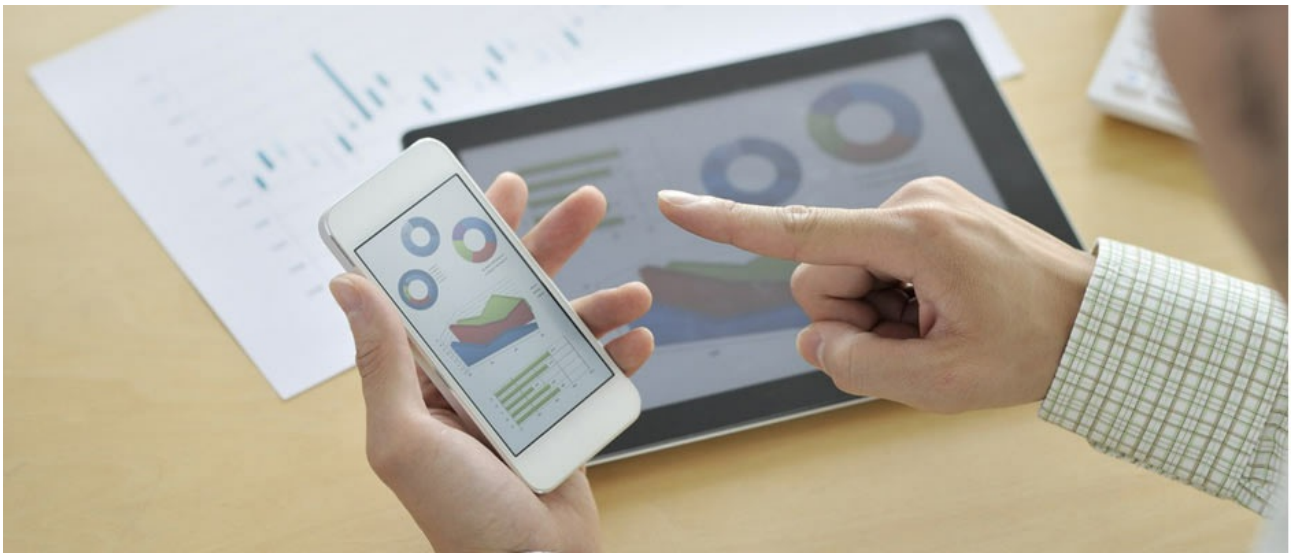

INFUZE NODE.JS USER GUIDE

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

www.scientiamobile.com
Tel +1.703.310.6650
E-mail: sales@scientiamobile.com

WURFL InFuze Module for Node.js: User Guide

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 foundhere. Release notes for each API can be found here.

Installing the Node.js Package

We provide a nodejs-mod_wurfl-X.Y.Z.tgz archive containing all the required files to install our module for Node.js and a README, where X.Y.Z stands for the currently used WURFL API version. Please note that Node.js modules only support three digit version numbers.

In this archive you will also find some JavaScript examples on how to configure and test WURFL through Node.js.

Before installing the WURFL Node.js module make sure that both Node.js version 0.10.26 (or greater) and npm (the Node.js tool to manage Node modules) are installed on your system.

Installing WURFL Node.js Module on on all operating systems

Once you have downloaded the Node.js module from your ScientiaMobile account, you are advised to install the WURFL Node.js module using the npm local (default) mode, because you will load the WURFL module from your sources using a require() call. For example, to install in a "testwurfl" directory:

```
mkdir testwurfl
cd testwurfl
npm install nodejs-mod_wurfl-X.Y.Z.tgz
```

During the installation process npm may show warnings regarding unmet dependencies. This is unrelated to WURFL and depends on which version of npm you are using and how those dependencies are handled by npm itself.

You can check for correct installation of WURFL Node.js module issuing the commandnpm list. You should see something similar to:

```
/path/to/testwurfl
â””â”€â”¬ nodejs-mod_wurfl@1.9.4.0
```

â””â”€â”€ nan@2.6.2

Sample Usage

This is an example of how to use the WURFL Node.js module:

```
var wurfl_nodejs_module = require('nodejs-mod_wurfl');
var wurfl = new wurfl_nodejs_module.Wurfl();

var config = {
    // on osX, wurfl.zip is located here: /usr/local/share/wurfl/wurfl.zip
    root: "/usr/share/wurfl/wurfl.zip",
}

var debug = true;

wurfl.initialize(config, debug);

// print general WURFL engine info
console.log("WURFL Info: " + wurfl.getInfo());
console.log("WURFL API Version: " + wurfl_nodejs_module.WurflAPIVersion());
console.log("WURFL Last Load Time: " + wurfl.getLastLoadTime());

// retrieve a device from an user-agent string
var device =  wurfl.lookup("Mozilla/5.0 (Linux; Android 5.0; SAMSUNG SM-G925 Build/LRX21V) AppleWebKit/537.3
6 (KHTML, like Gecko) SamsungBrowser/4.0 Chrome/44.0.2403.133 Mobile Safari/537.36");

// ask for some capabilities and properties:

// device id
console.log("Device ID is: " + device.getDeviceId());

// static capabilities
console.log("model_name = " + device.getCapability("model_name"));
console.log("brand_name = " + device.getCapability("brand_name"));
console.log("device_os = " + device.getCapability("device_os"));

// virtual capabilities
console.log("complete_device_name = " + device.getVirtualCapability("complete_device_name"));
console.log("form_factor = " + device.getVirtualCapability("form_factor"));
```

Please note that all cleanup/destroy actions are automatically handled by the NodeJS garbage collection mechanism.

Testing WURFL Node.js Module on all operating systems

Installation of the WURFL module through NPM, in any directory, will create a node_module directory. It also creates the nodejs-mod_wurfl directory within node_modules and, within node_modules/nodejs-mod_wurfl/, test and build. The node_modules/nodejs-mod_wurfl/build directory is the path which var wurfl_nodejs_module = require('nodejs-mod_wurfl'); tries to resolve to.

In order to reach the wurfl_nodejs_module.node plugin, any JS source file which does a require('nodejs-mod_wurfl') must be able to solve the path from where it resides to ../build/Release/wurfl_nodejs_module (a relative path), in order to reach the wurfl_nodejs_module.node plugin.

You must put your source in a directory at the same level of the test or build directories in node_modules/nodejs-mod_wurfl, or else Node will not be able to resolve to that relative path to the plugin.

Together with the WURFL Node.js module files we provide some sample JavaScript files, located in the "nodejs-mod_wurfl/test" directory, to demonstrate some common uses of the WURFL InFuze module for

Node.js.

Assuming that you've installed our WURFL module into~/testwurfl, you should find the module installed in ~/testwurfl/node_modules/nodejs-mod_wurfl.

Open a terminal, cd into ~/testwurfl, copy the wurfl.zip file (which, depending from your system, could be located in /usr/local/share/wurfl/wurfl.zip or /usr/share/wurfl/wurfl.zip), and type:

```
node node_modules/nodejs-mod_wurfl/test/test_minimal.js
```

The test_minimal.js example is the exact same source given above as sample usage, so you should see something like:

```
Setting WURFL file to ./wurfl.zip
Loading WURFL...
WURFL Info: Root:./wurfl.zip:WURFL API 1.8.4.0 - full, db.scientiamobile.com - 2017-03-29 14:24:22
WURFL API Version: 1.8.4.1
WURFL Engine Target: HIGH_PERFORMANCE
WURFL Useragent Priority: OVERRIDE SIDELOADED BROWSER USERAGENT
WURFL Last Load Time: Wed Jun  7 11:34:13 2017

Device ID is: samsung_sm_g925_ver1
model_name = SM-G925
brand_name = Samsung
device_os = Android
complete_device_name = Samsung SM-G925 (Galaxy S6 Edge)
form_factor = Smartphone
```

You can find more complex examples, like executing a lookup with full HTTP request headers to increase accuracy and/or using WURFL as a server in the test/test_wurfl.js and test/test_cli.js examples.

The Internal WURFL InFuze Updater

Since InFuze 1.8.3.0, a native internal Updater Module is available to automatically keep yourwurfl.zip up-to-date with the ScientiaMobile data release schedule.

All Updater functions are accessed via *XXXUpdaterXXX* Wurfl class methods. Also, the WurflInFuze.js helper contains, in its prototype.initialize() method, the code needed to set up everything. To use it, set the updater_data_url configuration parameter to your personal WURFL Snapshot URL ("https://data.scientiamobile.com/xxxxx/wurfl.zip", with "xxxxx" replaced with your personal access token - located in your license account page).:

```
var config = {
    root: "./wurfl.zip", // the only really mandatory parameter for the engine
    updater_log: "./wurfl_updater.log",
    updater_data_url: "https://data.scientiamobile.com/<your access token>/wurfl.zip", // the only really mandatory
 parameter for the updater
    updater_frequency: "DAILY",
}
```

Do note that the root path should be writable, and a wurfl.zip file must already be present in order for the Updater to determine whether or not it has to pull an update.

Some example client code using the pre-configured updater and issuing some synchronous and asynchronous calls can be found in test/test_cli.js.

If you prefer to fully configure and control the updater from your code, you can find a commented usage example in test/test_updater.js. Basically, this is an outline of what you have to do:

```
// OPTIONAL but highly suggested: set a log file
wurfl.setUpdaterLogPath("updater.log");

// MANDATORY: set data URL.
wurfl.setUpdaterDataURL("https://your-path-to-update-URL");

// OPTIONAL: set frequency of checks for an updated data file
wurfl.setUpdaterDataFrequency(1); // WEEKLY

// OPTIONAL: set timeouts for the connection and the data transfer phases, in milliseconds
// A lot of options here, please read documentation
wurfl.setUpdaterDataURLTimeouts(2000, 10000);
```

A correctly configured updater can then be used in two ways:

- syncronously, via the updaterRunonce() call
- asynchronously with updaterStart() and updaterStop()

Here "asynchronous" means that a low level (i.e. libwurfl C) thread is created and run in background, while "synchronous" means that the call is blocking at the libwurfl C level.

```
// start a libwurfl blocking update
wurfl.updaterRunonce();
```

or

```
// start and stop a non-blocking libwurfl background thread
wurfl.updaterStart();
....
wurfl.updaterStop();
```

It is up to the client to decide when to start asynchronous (updaterStart()/updaterStop()) or synchronous (updaterRunonce()) update operations.

Please note that the only mandatory call for the updater module to work is setUpdaterDataURL(), which depends on a successful setRoot() call:

- 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 the updater. If you plan to use this feature, you *MUST* use a compressed (i.e, a ZIP or a XML.GZ) file as the data root in the setRoot() call.

Explicitly setting the update frequency and timeouts is optional and have the defaults specified in the above documentation, while enabling file logging is optional but highly recommended.

> ***Note:*** *setUpdaterDataFrequency() sets how often the updater checks for an updated data file.*

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 setUpdaterDataURL() call

WURFL Node.js Module's function list

The module exposes a set of functions to be used to setup WURFL, query WURFL for specific capabilities, get general purpose information, and so on.

WURFL methods (class Wurfl, maps to InFuze wurfl_handle)

Function	Description	Availability (WURFL version)
setRoot(path_to_root_xml)	This function sets the root WURFL data file to be used by WURFL to a specific path in your file system. Please note that if you plan to use the updater feature, you MUST use a compressed (i.e, ZIP or XML.GZ) wurfl data file.	1.5.1.2
addPatch(path_to_patch_xml)	This function adds a patch to WURFL by taking the path to the patch xml file.	1.5.1.2
addRequestedCapability(capability_name)	Adds a new capability to the capabilities filter. If not used, all capabilities are loaded	1.5.1.2
setCacheProvider(cache_mode, max_useragents, max_devices)	This function sets the WURFL Cache provider to be used. Choose "cache_mode" between 0 (no cache) or 1 (single LRU cache, default). If you choose the single LRU cache you also need to pass a "max_useragents" integer which tells how many user agents WURFL can cache (recommended: "100000").	1.5.1.2
load()	Loads the WURFL Instance with the previously selected modes (engine target, cache, root data file..).	1.5.1.2
lookupUseragent(useragent)	This function is responsible to query WURFL for a device matching the passed "useragent" as a string. It returns a wurfl_device_handle structure.	1.5.1.2

Function	Description	Availability (WURFL version)
lookupWithHeaderResolverFunction(header_resolver)	This function is responsible to query WURFL for a specific device. The header_resolver function passed as a parameter must tell WURFL how to retrieve the header values. Please note that the header-retrieval functions should be case-insensitive. It returns a wurfl_device_handle structure.	1.5.1.2
getDevice(device_id)	This function is responsible to query WURFL for a specific device matching a specific wurfl device identifier as a string. It returns a wurfl_device_handle structure.	1.5.1.2
getLastLoadTime()	This function returns a string describing the timestamp of the latest successful WURFL load.	1.5.1.2
getInfo()	This function returns a string describing some information regarding the loaded WURFL database and optional patch files.	1.5.1.2
setUpdaterLogPath(file_path)	Instructs the internal WURFL InFuze updater to log to file any operation/error. If not used, the updater will not log anything.	1.8.3.0
setUpdaterDataURL(url)	Sets remote data file URL to be downloaded via internal WURFL InFuze updater. This is the only MANDATORY call if you want to use the InFuze Updater	1.8.3.0
setUpdaterDataFrequency(check_frequency)	Sets how often the updater checks for any new/updated WURFL data file to be downloaded and used by the engine (DAILY (default) or WEEKLY).	1.8.3.0

Function	Description	Availability (WURFL version)
setUpdaterDataURLTimeouts (connection_timeout, data_transfer_timeout)	Sets internal WURFL InFuze Updater timeouts, in milliseconds. 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., `updaterRunonce()`) API call. The asynchronous background updater invoked by `updaterStart()`/`updaterStop()` always runs with `curl` behaviour and timeouts (i.e., it will wait "as long as needed" for a new data file to be downloaded)	1.8.3.0
updaterStart()	Starts the asynchronous WURFL InFuze background update thread.	1.8.3.0
updaterStop()	Stops the asynchronous WURFL InFuze background update thread.	1.8.3.0
updaterRunonce()	Call a WURFL InFuze synchronous update.	1.8.3.0

WURFL Device methods (class WurflDevice, maps to wurfl_device_handle)

Function	Description	Availability (WURFL version)
getDeviceId()	This function retrieves the deviceId of a specific wurfl_device_handle as a string.	1.5.1.2
getRootId()	This function retrieves the root device identifier of a specific wurfl_device_handle as a string.	1.5.1.2
getOriginalUseragent()	Returns the original useragent of this device (as of WURFL database).	1.5.1.2
getNormalizedUseragent()	Returns the normalized useragent of this device.	1.5.1.3
isActualDeviceRoot()	Tells if this device is a root device in the device hierarchy.	1.5.1.2
hasCapability(cap)	Tells if this device has a capability named "cap".	1.5.1.2
hasVirtualCapability(vcap)	Tells if this device has a virtual capability named "vcap".	1.5.1.2
getCapability(cap)	Gets the capability value of the capability named "cap" as a string.	1.5.1.2
getVirtualCapability(vcap)	Gets the virtual capability value of the virtual capability named "vcap" as a string.	1.5.1.2
getCapabilityAsInt(cap)	Gets the capability value of the capability named "cap" as an integer.	1.5.1.2
getVirtualCapabilityAsInt(vcap)	Gets the virtual capability value of the virtual capability named "vcap" as an integer.	1.5.1.2
getCapabilityAsBool(cap)	Gets the capability value of the capability named "cap" as a boolean.	1.5.1.2

Function	Description	Availability (WURFL version)
getVirtualCapabilityAsBool(vcap)	Gets the virtual capability value of the virtual capability named "vcap" as a boolean.	1.5.1.2

We also provide a WurflInFuze.js wrapper which contains some useful utility functions:

WurflInFuze.js utility functions

Function	Description	Availability (WURFL version)
initialize(config, debug)	This function is responsible for initializing WURFL with a series of functionalities such as cache mode and custom capabilities filter. The first parameter "config" is a dictionary in which the keys are the functionalities' name, and the values are the correspondent specific values. Please refer to the `test_cli.js` test file (found in the "test" directory) for a complete example. The second parameter "debug" is a boolean which, if set to true, makes the loading process verbose.	1.5.1.2
addRequestedCapabilities(capabilities)	Adds a specific set of capabilities to the capabilities filter. The "capabilities" parameter must be an array of capability names.	1.5.1.2
lookupRequest(headers)	This function is useful to get a wurfl_device having passed a list of custom header names and values in the "headers" parameter. This function returns a wurfl_device_handle structure.	1.5.1.2

Function	Description	Availability (WURFL version)
lookup(obj)	This function is the main interface to get a wurfl_device. You can choose to pass three types of objects to this function. Specifically, you can pass a simple user-agent string, or a function which describes how to get the header values from the request object or ultimately a dictionary of header names and values (as for lookupRequest). This function returns a wurfl_device_handle structure.	1.5.1.2

Note: *If you decide to use the WurflInFuze.js wrapper you may need to edit the default_wurfl_root variable, which must point to the WURFL database you're using. See also the set_root() documentation.*