scientiamobile

"The first step in a great mobile experience"

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

scientiamobile

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 found here. 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 command npm 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
```

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.

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();
```

```
// Config contains some optional settings that can be used to initialize WURFL engine:
// - root: where the WURFL file is located. If not set, the WURFL file is searched in the current directory.
// - patches: an array of XML files used to customize WURFL file data.
// - cache: cache configuration, in the form <type: "value">. Possible values are:
//   - "none": no cache is used
//   - "single-lru": a single LRU cache is used (in this case you can specify max number of UAs cached)
// - updater_log: path of the updater log file. If not set, the updater log file is not created.
// - updater_data_url: URL to download WURFL file. If not set, the updater is not used.
//   - if updater_data_url is not set, even WurflDownload(), that gets an up-to-date WURFL file before engine creatio
n, isn't called.
// - updater_frequency: frequency of update attempt. Possible values are:
//   - "DAILY": daily update attempt (default)
//   - "WEEKLY": weekly update attempt
// - updater_timeouts: connection and data transfer timeouts in the form <connection: value, data_transfer: value
>.
//   - connection: connection timeout, in milliseconds. If not set, the default value is 10000 ms (10 seconds).
//   - data_transfer: data transfer timeout, in milliseconds. If not set, the default value is 600000 ms (600 seconds).
//   - Both values can be set to 0 to disable timeouts.
var config = {
    root: "./wurfl.zip", // Location of WURFL file.
//  patches: ["patch1.xml"],
//  cache: { type: "none" },
//  cache: { type: "single-lru", max_useragents: 10000 },
//  updater_log: "./wurfl_updater.log",
  updater_data_url: "https://data.scientiamobile.com/xxxxx/wurfl.zip",
//  updater_frequency: "DAILY",
//  updater_timeouts: { connection: 10000, data_transfer: 600000 },
}

var debug = true;

// initialize method (located in WurflInFuze.js file) sets up the WURFL engine
// It also call WurflDownload() if updater_data_url is set
wurfl.initialize(config, debug);

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

// lookup method with list of important headers (located in WurflInFuze.js file) retrieve a device
var device =  wurfl.lookupRequest({
    "user-agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.0.0 Safar
i/537.36",
    "sec-ch-ua": "\"Chromium\";v=\"122\", \"Not(A:Brand\";v=\"24\", \"Veera\";v=\"122\"",
    "sec-ch-ua-full-version-list": "\"Chromium\";v=\"122.0.0.0\", \"Not(A:Brand\";v=\"24.0.0.0\", \"Veera\";v=\"122.0.
0.0\"",
    "sec-ch-ua-mobile": "?1",
    "sec-ch-ua-platform": "Android",
});

// ask for some caps and properties
console.log("\n");
console.log("Device ID is: " + device.getDeviceId());

// Print out some static capabilities
console.log("model_name = " + device.getStaticCapability("model_name"));
console.log("brand_name = " + device.getStaticCapability("brand_name"));
console.log("device_os = " + device.getStaticCapability("device_os"));

// Print out some 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/examples" 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/node_modules/nodejs-mod_wurfl/examples, edit test_minimal.js and modify this line updater_data_url: "https://data.scientiamobile.com/xxxxx/wurfl.zip", setting it to your personal WURFL Snapshot URL (with "xxxxx" replaced with your personal access token - located in your license account page), and type:

```
node 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 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 some other examples in node_modules/nodejs-mod_wurfl/examples directory.

The Internal WURFL InFuze Updater

Since InFuze 1.8.3.0, a native internal Updater Module is available to automatically keep your wurfl.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",
    updater_log: "./wurfl_updater.log",
    updater_data_url: "https://data.scientiamobile.com/xxxxx/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 example/test_updater.js.

If you prefer to fully configure and control the updater from your code, you can find take a look to prototype.initialize() method in WurflInFuze.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://data.scientiamobile.com/xxxxx/wurfl.zip");

// 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(10000, 600000);
```

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 issetUpdaterDataURL(), which depends on a successful setRoot() call:

- The WURFL data file and the path where it resides, specified in thesetRoot() 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
DEPRECATED - setEngineTarget(wurfl_engine_target)	Method is deprecated, does nothing.	1.5.1.2
DEPRECATED - getEngineTarget()	Method is deprecated, return always DEFAULT.	1.5.1.2
DEPRECATED - setUserAgentPriority(useragent_priority)	Method is deprecated, does nothing.	1.5.2
DEPRECATED - getUserAgentPriority()	Method is deprecated, returns always OVERRIDE_SIDELOADED_BROWSER_USERAGENT	1.5.2

Function	Description	Availability (WURFL version)
setCacheProvider(cache_mode, max_useragents, max_devices)	This function sets the WURFL Cache provider to be used. Choose "cache_mode" between 0 (no cache), 1 (single LRU cache). Double LRU is deprecated and single LRU will be used instead if cache mode is set to 2 (old double LRU cache)	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
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

Function	Description	Availability (WURFL version)
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
DEPRECATED - listCapabilities()	This function returns a list of all capability names handled by the loaded WURFL.	Deprecated, use listStaticCapabilities instead. 1.11.8.1
listStaticCapabilities()	This function returns a list of all static capability names handled by the loaded WURFL.	1.13.2.0
listVirtualCapabilities()	This function returns a list of all virtual capability names handled by the loaded WURFL.	1.11.8.1
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

Function	Description	Availability (WURFL version)	
updaterStop()	Stops the asynchronous WURFL InFuze background update thread.	1.8.3.0	
updaterRunonce()	Call a WURFL InFuze synchronous update.	1.8.3.0	
getAllDeviceId()	Returns a list of all device id strings.	1.11.7.0	
asyncLookup(obj)	Performs a lookup in asynchronous mode, using the promise pattern. (see here)	1.11.8.1	
asyncGetDevice(wurfl_id)	Returns the WURFL device for the given `wurfl_id` in asynchronous mode, using the promise pattern. (see here)	1.11.8.1	
setAttr(attribute, value)	Set an attribute value (attributes are described in wurfl.h)	WARNING: using this method directly can cause unwanted behaviours. Use SetAttr method from WurflInFuze.js instead (see section below)	1.12.4.0
getAttr(attribute)	Get an attribute current value (attributes are described in wurfl.h)	WARNING: using this method directly can cause unwanted behaviours. Use GetAttr method from WurflInFuze.js instead (see section below)	1.12.4.0
isUAFrozen(useragent)	Return true if user agent is frozen, false if not	1.12.5.0	

Function	Description	Availability (WURFL version)
headerQualityWithHeaderResolverFunction(header_resolver)	This function is responsible to check quality of HTTP headers used by WURFL API. 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 quality value (0 = none; 1 = basic; 2 = full).	1.12.5.0
headerQualityToString(header_quality)	This function converts header quality integer values to their corresponding string values (0 will return None; 1 will return Basic; 2 will return Full).	1.12.6.0
hasStaticCapability(cap)	Tells if there is a static capability named "cap".	1.13.2.0
hasVirtualCapability(vcap)	Tells if there is a virtual capability named "vcap".	1.13.2.0

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

Function	Description	Availability (WURFL version)	
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	
DEPRECATED - hasCapability(cap)	Tells if this device has a capability named "cap".	Replaced by WURFL method hasCapability.	1.5.1.2
DEPRECATED - hasVirtualCapability(vcap)	Tells if this device has a virtual capability named "vcap".	Replaced by WURFL method hasCapability.	1.5.1.2
DEPRECATED - getCapability(cap)	Gets the capability value of the capability named "cap" as a string.	Replaced by getStaticCapability.	1.5.1.2
getStaticCapability(cap)	Gets the value of the static capability named "cap" as a string.	1.13.2.0	
getVirtualCapability(vcap)	Gets the virtual capability value of the virtual capability named "vcap" as a string.	1.5.1.2	
DEPRECATED - getCapabilityAsInt(cap)	Gets the capability value of the capability named "cap" as an integer.	Replaced by getStaticCapabilityAsInt.	1.5.1.2
getStaticCapabilityAsInt(cap)	Gets the value of the static capability named "cap" as an integer.	1.13.2.0	
getVirtualCapabilityAsInt(vcap)	Gets the virtual capability value of the virtual capability named "vcap" as an integer.	1.5.1.2	

Function	Description	Availability (WURFL version)
DEPRECATED - getCapabilityAsBool(cap)	Gets the capability value of the capability named "cap" as a boolean.	Replaced by getStaticCapabilityAsBool.
getStaticCapabilityAsBool(cap)	Gets the value of the static capability named "cap" as a boolean.	1.13.2.0
getVirtualCapabilityAsBool(vcap)	Gets the virtual capability value of the virtual capability named "vcap" as a boolean.	1.5.1.2
DEPRECATED - getAllCapabilities()	Deprecated, use getAllstaticCapabilities() instead	1.11.8.1
getAllStaticCapabilities()	Returns all the capabilities available for the current device. Capabilities are accessible as a dictionary (ie: capabilities['brand_name'])	1.11.8.1
getAllVirtualCapabilities()	Returns all the virtual capabilities available for the current device. Virtual capabilities are accessible as a dictionary (ie: virtual_capabilities['is_app'])	1.11.8.1
getStaticCaps(caps)	Received an array of static capabilities, and returns a dictionary of them with their values as strings.	1.13.2.0
getVirtualCaps(vcaps)	Received an array of virtual capabilities, and returns a dictionary of them with their values as strings.	1.13.2.0

Global methods (class WurflGlobals, no need of wurfl or device objects to use them)

Function	Description	Availability (WURFL version)
WurflAPIVersion()	This function returns the version of currently used libwurfl.	1.8.3.0
WrapperVersion()	This function returns the version of the nodeJS C++ module.	1.8.3.0
WurflDownload(URL, folder)	This function download the WURFL data file from the specified URL and saves it to the specified folder.	1.13.1.1

IMPORTANT NOTE: For examples on how to use these functions, please take a look at the JavaScript examples provided within the archive.

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 to initialize WURFL with a series of functionalities like the cache mode and a custom capabilities filter. The first parameter "config" is a dictionary in which the keys are the functionalities names and the values are the correspondent specific values. Please refer to test_minimal.js test file (included into "examples" directory) for a complete description of possible options. 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

Function	Description	Availability (WURFL version)
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
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
checkIsUAFrozen(obj)	This function is the main interface to get if UA is frozen or not. You can choose to pass two types of objects to this function. Specifically, you can pass a simple user-agent string, or a dictionary of header names and values. This function returns true if UA is frozen, false if not (or no UA are passed to it in the dictionary).	1.12.5.0
headerQuality(obj)	This function is the main interface to get quality of headers passed to WURFL. You can choose to pass two types of objects to this function. Specifically, you can pass a function which describes how to get the header values from the request object or ultimately a dictionary of header names and values. This function returns a quality value: 0 for none, 1 for basic and 2 for full.	1.12.5.0

Function	Description	Availability (WURFL version)
SetAttr(attribute, value)	Set an attribute value (See wurfl_attr_values in WurflInFuze.js for possible values)	1.12.6.0
getAttr(attribute)	Get an attribute value (See wurfl_attr_values in WurflInFuze.js for possible values)	1.12.6.0

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.*