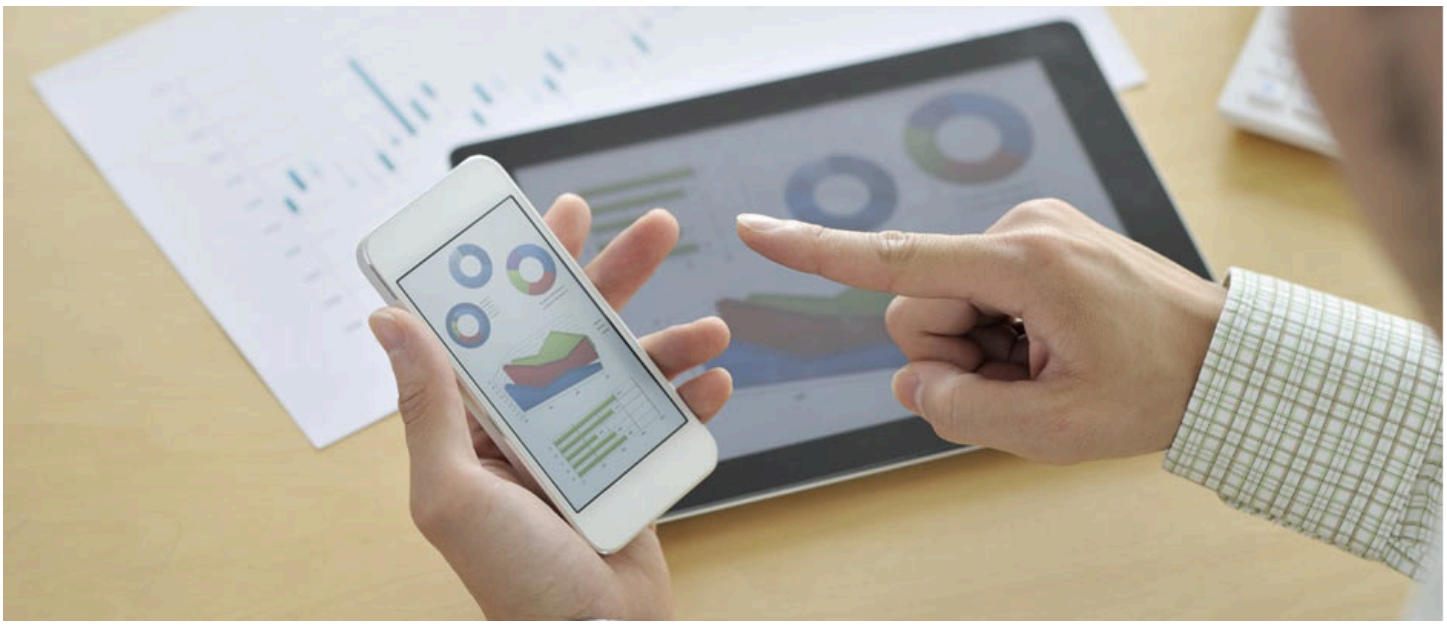
scientiamobile

"The first step in a great mobile experience"



WURFL ONSITE JAVA API: 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

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 OnSite Java API: User Guide

Introduction

WURFL OnSite for Java Application Programming Interface (API) is a mobile device detection software component that can quickly and accurately detect device models and over 500 capabilities of that device. The API library allows applications to perform real-time detection for a variety of uses, such as content adaptation, redirections, advertising, and data traffic analysis.

If you would like to learn general information about WURFL and device detection, you can read e-book here: https://www.devicedetection.com/

Operating System Support

Not applicable

Java version support

As of API version 1.12.3.0, WURFL OnSite for Java requires Java 8 or higher.

With the introduction of Jakarta EE 9, the enterprise Java application ecosystem has faced a huge change. The most impacting one is the naming change from the Oracle owned `javax.*` package namespace to the new `jakarta.*` In order to use the new namespace we provided a `wurfl-jakarta-X.Y.W.Z.jar` which is built using the new jakarta namespace. You can find it inside the OnSite Java API package, in the **Downloads** section of your license in the **Customer Vault**.

If you need to develop and deploy a web application that runs on Tomcat 10.x, Glassfish 6.x, Jetty 11.x or any other server that comply to this spec, you may want to use the jakarta compliant jar file.

Installation Process

Start a Trial or Purchase a License

To access the installation package, please create a **free ScientiaMobile account** and **contact us** to initiate a trial or to purchase a license.

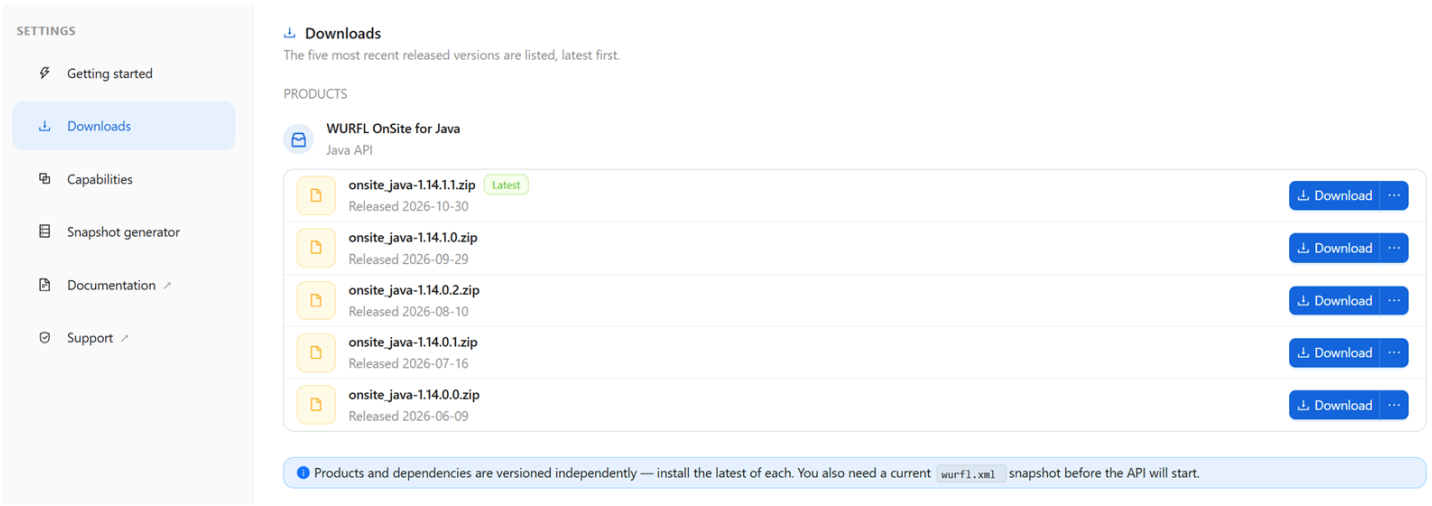
Where to Download the Installation Package

To download the installation package, **log in to your existing ScientiaMobile account**. On the **My Account** page of the Customer Vault, click **Manage** on your OnSite license, then click **Downloads**. Click **Download** next to a release of the WURFL API associated with your license to download it directly. To download it from the command line or another

machine instead, open the ▫ menu next to the release and choose **Copy download link**, **Copy curl command**, or **Show link and command...**

> ***Note:*** *Download links are short-lived and only download the selected package. Copy a new link each time you need one, rather than saving it in scripts.*

For details on downloading WURFL data snapshots, please refer to **this documentation**.

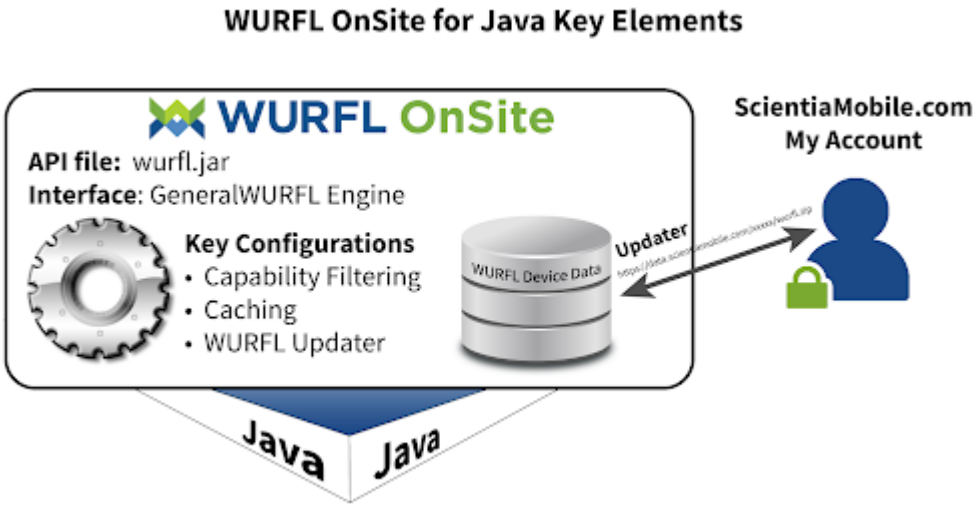


What's In The Installation Package

The provided zip file contains all the files you need to install and run WURFL. Some key components include:

`wurfl.jar` - this jar file is the WURFL OnSite API

`wurfl.zip` - contains the wurfl.xml device data file, also referred to as the WURFL Snapshot. This file contains all the devices known to WURFL. A limited `wurfl.zip` file is included with the package to quickly demo the product. You will eventually want to download a full copy of your `wurfl.zip` by using your personal WURFL Snapshot URL and applying the `WURFLUpdater` class (see below) to automate the weekly update of your WURFL Snapshot. If you would like to trial a different set of capabilities, then please contact your ScientiaMobile account representative.



Installing Dependencies

If you are using Maven to handle external dependencies, follow the instructions in the last paragraph to setup ScientiaMobile's private Maven repository usage in your project.

If you do not use Maven or any other dependency management tool, all the dependencies you need are packed into the onsite zip release file under core.zip/release.zip, where you will find a file `wurfl-version_number.jar` and a directory lib, which contains all the dependencies needed by the WURFL Java API to compile and run.

Installing WURFL

Copy the dependencies in your project and compile and run your Java programs using the `-cp` option. If you need more information about Java compilation from the command line, you can read this article by **Oracle**

> ***Note:*** *The WURFL API is closely tied to the* `wurfl.xml` *file. New versions of the* `wurfl.xml` *are compatible with old versions of the API by nature, but the reverse is* ***not*** *true. Old versions of the* `wurfl.xml` *are* ***not*** *guaranteed to be compatible with new versions of the API.*

Starting WURFL

The `WURFLEngine` is a high-level interface, introduced to further abstract and simplify management of the WURFL API.

Initializing the `WURFLEngine` is enough to start using the API and is the entry point to all WURFL functionalities. The default implementation of `WURFLEngine` is `GeneralWURFLEngine`.

Since v1.8.0.0, the package names have changed to com.scientiamobile. Please update your configuration accordingly. If you need to use the old package name (net.sourceforge.wurfl), a legacy release package will be present. New modules, such as `WURFLUpdater` *, will only be available with the new package names.*

***Note**: The release of WURFL OnSite for Java (v1.9.4.0) was the last release to contain support for the legacy package name (net.sourceforge.wurfl) and the legacy release package. Please update to the new package name (com.scientiamobile) to avoid any disruptions to your project.*

```
import com.scientiamobile.wurfl.core.Device;
import com.scientiamobile.wurfl.core.GeneralWURFLEngine;

public class Wurfl {

    public static void main(String[] args) {

        // Before initializing the WURFL Engine, we download an updated copy of the
wurfl file in the current directory (must be writable!)
        // replace this URL with your customer specific one
        String wurflUrl = "https://data.scientiamobile.com/xxxxx/wurfl.zip";
        try {
          GeneralWURFLEngine.wurflDownload(wurflUrl, "."); // you can also use "."
for current directory
        }
        catch(Exception e){
          // handle exception
        }

        // Once done, we initialize the WURFL Engine

        // wurfl.xml path can be either absolute or relative.
        WURFLEngine wurfl = new GeneralWURFLEngine("wurfl.zip");
        // load method is available on API version 1.8.1.0 and above
        wurfl.load();

        String user_agent = "Mozilla/5.0 (Linux; Android 4.2.1; N9600 Build/JOP40D)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.111 Mobile Safari/537.36";

        Device device = wurfl.getDeviceForRequest(user_agent);
        System.out.println("Is Tablet: " + device.getCapability('is_tablet'));
        System.out.println("Can Assign Phone Number: " +
device.getCapability('can_assign_phone_number'));
    }
}
```

If you are configuring a web application using a `web.xml` file, you can:

```
<listener>
  <listener-class>
     com.scientiamobile.wurfl.core.web.WURFLServletContextListener
  </listener-class>
</listener>

<context-param>
  <param-name>wurfl</param-name>
  <param-value>/WEB-INF/wurfl.zip</param-value>
</context-param>

<context-param>
  <param-name>wurflPatch</param-name>
  <param-value>/WEB-INF/patch_1.xml,/WEB-INF/patch_2.xml</param-value>
</context-param>
```

WURFLEngine implementation is loaded lazily (ie: when getDeviceById or getDeviceForRequest are called for the first time), but starting from version 1.8.1.0 you can load it programmatically by invoking method `load`.

```
WURFLEngine wurfl = new GeneralWURFLEngine("wurfl.xml");
wurfl.load();
```

Connecting to a User Agent Source

The `WURFLServletContextListener` will automatically find the user agent string of an HTTP request. For testing purposes, you can manually define the User-Agents submitted to WURFL via `wurfl.getDeviceForRequest(user_agent)`.

If you wish to manually define a HTTP request, with User-Agent Client Hints for example, you can construct a map and manually place all the headers like so:

```
// Construct a header map
String ua = "Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/100.0.0.0 Mobile Safari/537.36";
Map<String,String> headers = new HashMap<>();
headers.put("User-Agent", ua);
headers.put("Sec-Ch-Ua", "\" Not A;Brand\";v=\"99\", \"Chromium\";v=\"96\", \"Google
Chrome\";v=\"96\"");
headers.put("Sec-Ch-Ua-Full-Version-List", "\" Not A;Brand\";v=\"99.0.1.2\",
\"Chromium\";v=\"96.1.2.3\", \"Google Chrome\";v=\"96.2.3.4\"");
headers.put("Sec-Ch-Ua-Platform", "Android");
headers.put("Sec-Ch-Ua-Platform-Version", "12");
headers.put("Sec-Ch-Ua-Model", "Pixel 6");
```

You can then use the constructed header map to create a `WURFLRequest` object which can be used to perform device detection:

```
// Perform the device detection using the WURFL OnSite for Java API using the
constructed header map
WURFLRequest req = new DefaultWURFLRequest(headers);
device = wurfl.getDeviceForRequest(req);
```

If you do not need to check the **header quality**, you can directly pass the constructed header map to `getDeviceForRequest()` method as shown here:

```
// Construct a header map
String ua = "Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/100.0.0.0 Mobile Safari/537.36";
Map<String,String> headers = new HashMap<>();
headers.put("User-Agent", ua);
headers.put("Sec-Ch-Ua", "\" Not A;Brand\";v=\"99\", \"Chromium\";v=\"96\", \"Google
Chrome\";v=\"96\"");
headers.put("Sec-Ch-Ua-Full-Version-List", "\" Not A;Brand\";v=\"99.0.1.2\",
\"Chromium\";v=\"96.1.2.3\", \"Google Chrome\";v=\"96.2.3.4\"");
headers.put("Sec-Ch-Ua-Platform", "Android");
headers.put("Sec-Ch-Ua-Platform-Version", "12");
headers.put("Sec-Ch-Ua-Model", "Pixel 6");
// Perform the device detection using the WURFL OnSite for Java API using the
constructed header map
Device device = wurfl.getDeviceForRequest(headers);
```

IMPORTANT: *Empty header values are treated as valid and those headers are not discarded. If you build your HTTP request programmatically from a data source such as logs, DB data, spreadsheet, etc., please make sure that you DO NOT add headers with empty strings as values (this may also be the result of "casting" a* `NULL` */* `NONE` */* `NaN` *to a string):*

Avoid:

```
{headerName}:{headerValue}
```

Use:

```
if notNullOrEmpty(headerValue):
    {headerName}:{headerValue}
```

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**.

Retrieving Static WURFL Capabilities

To retrieve information about a device, WURFL matches a user agent to a specific device profile and then performs a lookup of the device's WURFL capabilities.

There are two types of WURFL capabilities: **static** and **virtual** (see the next section for virtual capabilities instructions). Each type of capabilities require different methods to retrieve values, so be sure to determine whether your WURFL capabilities are static or virtual by consulting our WURFL capabilities documentation (https://www.scientiamobile.com/capabilities).

Static capabilities describe device and browser properties that do not (normally) change. For instance, a device's resolution_width will not change over time. Static properties can be stored in memory by the API, readily available for the next fast look-up.

To retrieve the value of a static capability, use the `getCapability()` method from a `Device` object:

For example, to retrieve the value of the static capability resolution_width (the screen width expressed in physical pixels) you would use the following:

```
device.getCapability('resolution_width')
```

Retrieving Virtual WURFL Capabilities

Virtual capabilities return a property whose value is based on the evaluation of other static capabilities or further inspection of parameters found in the HTTP request. Please check if your capabilities are among WURFL's virtual capabilities listed on this **table** Unlike static capabilities, virtual capabilities use the `getVirtualCapability()` method.

For example, to retrieve the value of the virtual capability 'is_smartphone' (a boolean value based on whether a device conforms to ScientiaMobile's definition of a smartphone), you would use the following:

```
device.getVirtualCapability('is_smartphone')
```

Note: You can override virtual capability return values by assigning values to their corresponding control capabilities (controlcaps) in a **patch file**. A more detailed write-up of control capabilities and how they work with virtual capabilities is available **here**.

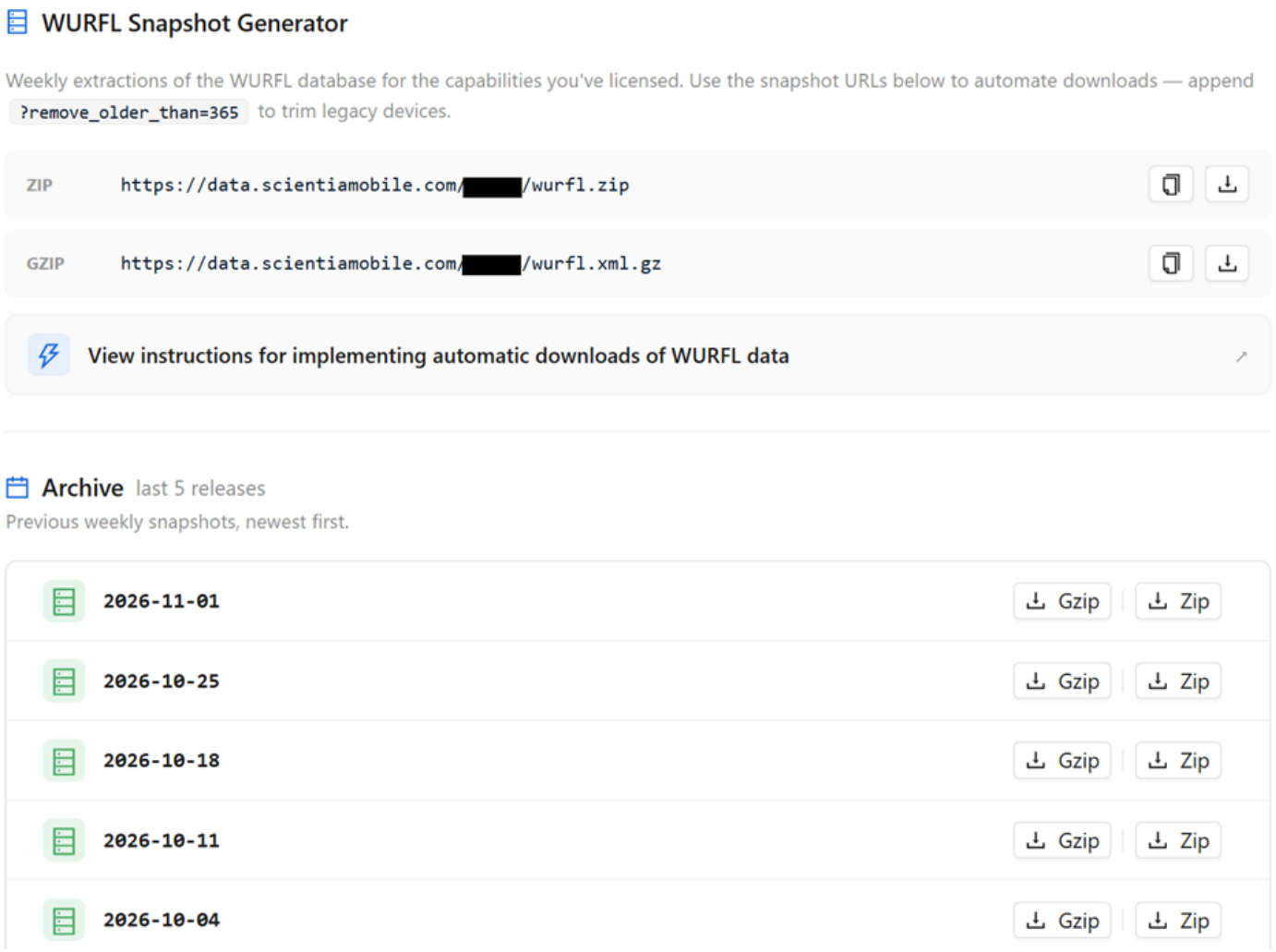
How to Improve and Maintain WURFL Accuracy

Configuring WURFL Updater

A new WURFL device data snapshot is released weekly on Sunday night. An out-of-band WURFL snapshot may be released to our customers between weekly snapshots in case of a significant improvement to the data (for example, if a high-profile device is released mid-week and it is not yet in WURFL).

API version 1.8.0.0 introduced WURFL updater; a new set of classes which allow a client to automatically update their WURFL Snapshot. In order to use the `WURFLUpdater`, you must have your personal WURFL Snapshot URL in the following format: https://data.scientiamobile.com/xxxxx/wurfl.zip. Your personal WURFL Snapshot URL can be found by logging into your **ScientiaMobile account**:

My Account -> **Manage** (for your licensed product) -> **Snapshot generator**
(https://data.scientiamobile.com/xxxxx/wurfl.zip)



Do note that the `rootPath` passed to the `WURFLEngine` constructor must be writable from the process/task that is executing the Java API since `WURFLUpdater` will update `rootPath` file, and a wurfl.zip file must already be present in order for the Updater to determine whether or not it needs to update the file.

The suggested setting for WURFL updater is "periodic" mode; i.e. `WURFLUpdater` will periodically check to see if a new version of the wurfl.zip has been released, and if so, download it and reload the engine with the new version; all while the standard `WURFLEngine` is running and serving requests.

Running "periodic" Updates.

```
String rootPath = "wurfl.zip";
WURFLEngine engine = new GeneralWURFLEngine(rootPath);
// remember to modify the url below with your personal WURFL Snapshot url
WURFLUpdater updater = new WURFLUpdater(engine,
"https://data.scientiamobile.com/xxxxx/wurfl.zip");
updater.setFrequency(Frequency.DAILY);
updater.performPeriodicUpdate();
```

Being a periodic task, the updater will run perpetually until `updater.stopPeriodicUpdate()` is called. Periodic updates do not return a result. Failed/Successful results must be checked in log files/console messages.

If needed, `WURFLUpdater` can also run in "on demand" mode, i.e. check for a new version of the `wurfl.zip` once and then stop.

Running "on demand" update.

```
String rootPath = "wurfl.zip";
WURFLEngine engine = new GeneralWURFLEngine(rootPath);
// remember to substitute url below with your personal WURFL Snapshot url
WURFLUpdater updater = new WURFLUpdater(engine,
"https://data.scientiamobile.com/xxxxx/wurfl.zip");
UpdateResult result = updater.performUpdate();
```

On demand updates run once per call and returns a result that can be used to programmatically check if update has been successful or, in case of failure, get an error message.

Proxy Configuration for WURFL Updater

Sometimes you may need to configure a proxy to run a `WURFLUpdater` instance. In this case, the initialization becomes:

```
String rootPath = "wurfl.zip";
WURFLEngine engine = new GeneralWURFLEngine(rootPath);
// proxy settings
String proxyHost = "proxy.example.com" // replace it with your proxy host
int proxyPort = 80; // replace with your proxy port
ProxySettings proxy = new ProxySettings(proxyHost, proxyPort, Proxy.Type.HTTP);
// ProxySettings proxy = new ProxySettings(proxyHost, proxyPort, Proxy.Type.SOCKS);
// SOCKS proxy type is also supported.

// remember to modify the url below with your personal WURFL Snapshot url
WURFLUpdater updater = new WURFLUpdater(engine,
"https://data.scientiamobile.com/xxxxx/wurfl.zip", proxy);
```

Updating the WURFL API

While there is no rigid schedule for API updates, historically ScientiaMobile releases API updates on a quarterly basis. In addition to weekly updates of the device data snapshot, updating the API is also critical to maintaining accuracy and performance. As part of the WURFL license, customers are entitled to upgrading to the latest and greatest API release. These API updates will appear in the **Downloads** section of your license in the Customer Vault when they become available.

We will send a notification of the API update and details on the improvements included in the new release to the email associated with the account. You may manually subscribe to the ScientiaMobile eNewsletter list if you have another address where you'd like to receive the notifications.

Java users who use Docker may also consider using **WURFL Microservice for Docker** to simplify the deployment and maintenance process involved in API updates.

How to Accelerate Performance and Scaling

Through caching and leveraging multi-threaded processing, WURFL OnSite for Java can achieve strong performance and scaling.

Cache Configuration

In order to increase performance while processing real HTTP traffic, we suggest setting up an Least Recently Used (LRU) cache. The LRU caching strategy will speed up look-up operations on processed User Agents by keeping them in an LRU map. For example, the following configures the size of the map to 100,000 entries:

```
cp = new LRUMapCacheProvider(100000);
engine.setCacheProvider(cp);
```

By default the cache will be set to 30,000 entries which accounts for 7 to 10 MB of additional memory usage. Users are advised to size their cache generously (100,000 or more) to increase performance.

For Users of the old DOUBLE LRU Cache Provider (pre 1.9.1): Backwards compatibility, older configurations are still supported and will not generate errors or warnings, but internally the new SINGLE LRU Cache is adopted for better performance.

For more information, please see LRU Cache Mechanism.

> ***Note:*** `engine.setCacheProvider(cp);` *must be called before calling* `engine.load()`.

Checking User-Agent frozenness and HTTP headers quality

Starting from version 1.12.5.0 the WURFL Onsite Java API provides two new methods of the `WURFLEngine`: `isUaFrozen()` and `headerQuality()`.

`boolean isUaFrozen(String userAgent)` returns a boolean value which, if true, means that the input User-Agent string won't be updated by the sender browser.

`HeaderQuality headerQuality(HttpServletRequest request)` returns an enumeration value that describes the HTTP headers quality. It has three possible values:

- **HEADER_QUALITY_FULL**: all the headers needed for a successful WURFL detection are present. Eg. a header with all UA-CH header fields present
- **HEADER_QUALITY_BASIC**: only some of the headers needed for a successful WURFL detection are present
- **HEADER_QUALITY_NONE**: no UA-CH headers are present

Multi-threading and Multi-processor

WURFL OnSite for Java can improve performance through multi-threaded processing. All WURFL APIs are thread safe. Users should share the same WURFL engine across all different threads that need to execute lookups.

Customizing WURFL with patch files

The idea behind OnSite and InFuze is that you get an updated version of the `wurfl.xml` file as new and more accurate information becomes available and is added to WURFL. Developers can create and add a `wurfl_patch.xml` file to their system, which stores modified/enhanced groups and capability lists for new or existing WURFL devices.

When the `wurfl.xml` file is parsed, the patch file is also imported to build a modified version of the device database. To learn more, refer to our **patch file documentation**

Configuring with the Spring Framework

The WURFL API (starting with 1.4) is completely decoupled from the Spring framework. This allows non-Spring users to import and use WURFL without having to include the Spring library and everything involved with it.

Using Spring does not mean that the `web.xml` file is no longer involved. You must tell your application that all the features of the API are now supported through Spring.

In practice, this means that the `web.xml` file will contain the following lines:

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/wurfl-ctx.xml</param-value>
</context-param>
  :
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```

In the `wurfl-helloworld-spring-{version}.war` web application, you can find an easy way to configure the WURFL API using Spring. The `wurfl.ctx` context file will contain:

```
<bean id="WurfEngine" class="com.scientiamobile.wurfl.core.GeneralWURFLEngine">
  <constructor-arg index="0" value="classpath:/wurfl.zip" />
  <!-- <constructor-arg index="1" value="<< patch here >>"/> -->
  <!-- <constructor-arg index="2" value="<< more patches here >>"/> -->
</bean>

<!-- DeviceCacheProvider -->
<bean id="deviceCacheProvider"
class="com.scientiamobile.wurfl.core.cache.LRUMapCacheProvider" />
<!-- <bean id="deviceCacheProvider"
class="com.scientiamobile.wurfl.core.cache.NullCacheProvider" /> -->
```

Finally, you can simply access the `WURFLEngine` instance with the code here (taken from a common Servlet method):

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
  WebApplicationContext wac =
WebApplicationContextUtils.getWebApplicationContext(getServletContext());
  WURFLEngine engine = (WURFLEngine)wac.getBean(WURFLEngine.class.getName());
  [...]
}
```

Generating a List of Possible Values

Frequently, developers need to generate lists of possible values for WURFL capabilities. These lists can be useful in building filtering, drop-down lists, or selection interfaces in applications.

Please contact our support for tools to generate such lists.

Sample Applications

The Java OnSite API release contains two web application samples that can be helpful in understanding how you can setup your own WURFL based web application:

- wurfl-helloworld-servlet.zip
- wurfl-helloworld-spring.zip

For the application to work, a `wurfl.xml` (.zip) file must be added in the application classpath before compiling it.

Getting Support

All current customers should contact us with technical/support questions through our **Enterprise Support Portal**. Customers receive enterprise-level ticketed support. Support specialists with over 30 years combined experience in device detection provide fast responses to developers. ScientiaMobile actively moderates and responds to the enterprise-level tickets.

Change History

There were breaking changes in API release v1.8. For documentation of releases before API v1.8 please refer to the README file included with that version. If you need additional help for versions prior to 1.8 feel free to contact us at support@scientiamobile.com

***Decommissioning of engine target options**: Prior to v1.9 of the API, users could choose between* `set_engine_target_high_accuracy` *and* `set_engine_target_high_performance` *engine optimization options.These options had been introduced years ago to manage the behavior of certain web browsers and their tendency to present "always different" User-Agent strings that would baffle strategies to cache similar WURFL queries in memory. As the problem has been solved by browser vendors, the need to adopt this strategy has diminished and ultimately disappeared (i.e. there was no longer much to be gained with the high-*

performance mode in most circumstances) and ScientiaMobile elected to "remove" this option to simplify configuration and go in the direction of uniform API behavior in different contexts.

Configuring your Builds to work with ScientiaMobile's Private Maven Repository

If you are a commercial licensee of the WURFL OnSite Java API, it is as easy as adding your scientiamobile.com credentials into your settings.xml file and including WURFL as a dependency in your pom.xml file.

You will first need to define an ID using a username and password that are associated with your license on scientiamobile.com and then place that into your settings.xml in our Maven installation. File settings.xml and the Maven dependency repository are usually located under the .m2 Maven directory, which is - by default - created under the user home directory (ie: if your home directory on Ubuntu Linux is jamie, the .m2 directory full path will be `/home/jamie/.m2`)

```
<server>
     <id>com.scientiamobile.wurfl</id>
     <username>YourUsername</username>
     <password>YourPassword</password>
</server>
```

Now that your credentials have been added, you will need to add the WURFL repository in your pom.xml file:

```
<project>
(...)
     <repositories>
          <repository>
          <!-- Define the ID that was added in your Maven's settings.xml -->
          <id>com.scientiamobile.wurfl</id>
          <!---You do not need to change the following lines -->
          <name>com.scientiamobile.wurfl</name>
          <url>https://maven.scientiamobile.com/repository/wurfl-onsite/</url>
          </repository>
     </repositories>
</project>
```

Please, note that in case depencies other than WURFL are needed, you may need to configure more than one repository.

Let's also add WURFL as a dependency:

```
<project>
(...)
     <dependencies>
     (...)
          <dependency>
               <groupId>com.scientiamobile.wurfl</groupId>
               <artifactId>wurfl</artifactId>
               <version>1.13.2.0</version>
          </dependency>
          (...)
     </dependencies>
     (...)
</project>
```

After everything has been configured with Maven, you are now ready to build your project and get started integrating WURFL!

Building your own project with Maven

After you have completed the WURFL integration in your project, you can build it by using the following command in the project directory that contains the pom.xml file (usually the project's root directory)

```
mvn clean install
```

This command cleans the build output directory, downloads the needed dependencies, compiles the application, executes the unit tests - if any -, generates the executable (.jar or .war depending on the build configuration file) and store it in your local Maven repository.

If you are new to Maven build manager and want to get started with it, you may want to take a look to **Maven website**

© 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.