scientiamobile

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

# ONSITE SCALA API

## 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 OnSite Scala API: User Guide

The Scala API is built as a wrapper around the WURFL Java API to expose methods to be available from within a Scala application. Below are the requirements that are needed in order to start including WURFL into your Scala project.

## Requirements

- Java 8 JDK or above
- Scala library 2.12.x/2.13.x/3.x
- WURFL Java jar 1.11.0.0 or greater (it is highly recommended to use the same version of Scala and Java jar files)

## Installation

To enable WURFL on your application, you must register for a free account on scientiamobile.com and **download the latest release from your File Manager.** Once you have downloaded the latest release, you will simply need to add the WURFL Java JAR and the wurfl-scala.jar to your build path. Current distribution contains four JAR files, one for each supported Scala version. Pick the one that fits your needs.

> ***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.*

## 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.

## Create a WURFL object

```
// Before initializing the WURFL object, we download an updated version of the WURFL file in
// a writable directory of our choice
// Replace this URL with your customer specific one (or you'll get a 402 error)
val wurflDataUrl = "https://data.scientiamobile.com/xxxxx/wurfl.zip"
val wurflDownloadPath = "." // in this example, we download the file in the current directory

try {
  Wurfl.wurflDownload(wurflDataUrl, wurflDownloadPath)
} catch {
  case ex: WURFLRuntimeException =>
    // handle the exception
  case ex: BadWurflExtensionException =>
    // handle the exception caused by an invalid WURFL file extension
}

// Create the WURFL object
val wurflScala = Wurfl.apply("wurfl.zip");
```

## Caching

It is possible to configure the API with a caching strategy in order to increase performance. Once you have instantiated your WURFL object, you must set the cache provider that you would like to use.

The available caching strategy is LRUMapCacheProvider.

**LRUMapCacheProvider:**

```
// Create a WURFL object
val wurflScala = Wurfl.apply("/path_to/wurfl.zip")
```

```
// Set cache provider
wurflScala.setCacheProvider(new LRUMapCacheProvider)
```

## Virtual Capabilities

Virtual capabilities are an important feature of the WURFL API that obtain values related to the requesting agent out of the HTTP request as a whole (as opposed to limiting itself to capabilities that are found in WURFL).

In order to compute its final returned value, a virtual capability may look at regular (non-virtual) capabilities as well as parameters derived from the HTTP request at run-time. Virtual capabilities are useful to model aspects of the HTTP Client that are not easily captured through the finite number of profiles in WURFL.

```
val wurflScala = Wurfl.apply("/path_to/wurfl.zip")

val userAgent = "Mozilla/5.0 (Linux; Android 14; Pixel 8) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.39
45.136 Mobile Safari/537.36"
// Create a Map to store headers
val headers = Map(
  "Accept" -> "*/*",
  "Accept-Encoding" -> "gzip, deflate, br, zstd",
  "Sec-Ch-Ua" -> "\"Chromium\";v=\"124\", \"Google Chrome\";v=\"124\", \"Not-A.Brand\";v=\"99\"",
  "Sec-Ch-Ua-Full-Version" -> "\"124.0.6367.82\"",
  "Sec-Ch-Ua-Full-Version-List" -> "\"Chromium\";v=\"124.0.6367.82\", \"Google Chrome\";v=\"124.0.6367.82\", \"
Not-A.Brand\";v=\"99.0.0.0\"",
  "Sec-Ch-Ua-Mobile" -> "?1",
  "Sec-Ch-Ua-Model" -> "\"Pixel 8\"",
  "Sec-Ch-Ua-Platform" -> "\"Android\"",
  "Sec-Ch-Ua-Platform-Version" -> "\"14.0.0\"",
  "User-Agent" -> userAgent
    )

    val device = wurflWrapper.deviceForRequest(headers)

// Access a virtual capability
var smartphone = device.virtualCapability("is_smartphone")
```

For further information on virtual capabilities, please refer to the [WURFL capabilities page]

## Configuring WURFL updater

API version 1.8.2.0 introduces the WURFL updater; a new set of classes which allow a client using WURFL to automatically update the WURFL file. In order to use the WURFL Updater, you must have your personal WURFL Snapshot url in the following format: https://data.scientiamobile.com/xxxxx/wurfl.zip where xxxxx is replaced with you personal access token. Also, do note that file path used to create the WURFL engine must be writable from the process/task that is executing the Scala API since Updater will update the file denoted by that path.

The suggested setting for the WURFL updater is "periodic" mode - in which theUpdater will periodically check to see if a new version of the wurfl.zip file has been released and, if so, download it and reload the engine with the new version. This can occur while the standard Wurfl engine is running and serving requests.

### Running "periodic" updates.

```
val updater = Updater.apply(engine, "https://data.scientiamobile.com/xxxxx/wurfl.zip", patchPaths, proxySettings)
updater.performPeriodicUpdate(Frequency.DAILY)
```

proxySettings and patchPaths can be null

Being a periodic task, the updater will run perpetually untilupdater.StopPeriodicUpdate() is called. Periodic update does not return a result. Failed/successful results must be checked in log files/console messages.

If needed, Updater can also run in "on demand" mode - in which it will check for a new version of the wurfl.zip once and then stop.

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

### Running "on demand" update.

```
val updater = Updater.apply(engine, "https://data.scientiamobile.com/xxxxx/wurfl.zip", patchPaths, proxySettings)
UpdateResult result = updater.PerformUpdate()
```

On demand update runs only 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.

## Sample

A sample getting started project can be found below:

```
import com.scientiamobile.wurfl.{Updater, Wurfl}
import com.scientiamobile.wurfl.core.cache.LRUMapCacheProvider
import com.scientiamobile.wurfl.core.exc.WURFLRuntimeException
import com.scientiamobile.wurfl.core.updater.Frequency
import com.scientiamobile.wurfl.core.updater.exc.BadWurflExtensionException

object Demo {

  def main(args: Array[String]) = {

    print("Running scala version ")
    println(scala.util.Properties.scalaPropOrElse("version.number", "unknown scala version"))

    // Download an updated version of the WURFL file in the current directory before  creating the WURFL engine
    // Replace this URL with your customer specific one (or you'll get a 402 error)
    val wurflDataUrl = "https://data.scientiamobile.com/xxxxx/wurfl.zip"
    val wurflDownloadPath = "."

    try {
      Wurfl.wurflDownload(wurflDataUrl, wurflDownloadPath)
    } catch {
      case ex: WURFLRuntimeException =>
        println(ex.getMessage)
        sys.exit(1)
      case ex: BadWurflExtensionException =>
        println(s"Invalid WURFL file extension in WURFL url: ${ex.getMessage}")
        sys.exit(1)
    }


    // Create WURFL passing a GeneralWURFLEngine object with a wurfl xml
    val wurflWrapper = Wurfl.apply("wurfl.zip");

    // Set cache provider
    wurflWrapper.setCacheProvider(new LRUMapCacheProvider(5000))

    // Create and start a WURFL updater that checks for new WURFL files on a daily basis
    val updater: Updater = Updater.apply(wurflWrapper, wurflDataUrl)
    try {
      updater.performPeriodicUpdate(Frequency.DAILY)
      println("Started updater")

      // User-Agent here
      val userAgent = "Mozilla/5.0 (Linux; Android 15; Pixel 7 Build/AP31.240517.022; wv) AppleWebKit/537.36 (KHT
ML, like Gecko) Version/4.0 Chrome/126.0.6478.133 Mobile Safari/537.36App/3.2.4b47 (Android App)"

      // Create a Map to store headers
```

```
    val headers = Map(
      "Accept" -> "*/*",
      "Accept-Encoding" -> "gzip, deflate, br, zstd",
      "Accept-Language" -> "en-GB,en-US;q=0.9,en;q=0.8",
      "Sec-Ch-Ua" -> "\"Not/A)Brand\";v=\"8\", \"Chromium\";v=\"126\", \"Android WebView\";v=\"126\"",
      "Sec-Ch-Ua-Mobile" -> "?1",
      "Sec-Ch-Ua-Platform" -> "\"Android\"",
      "User-Agent" -> userAgent
    )

    val device = wurflWrapper.deviceForRequest(headers)

    // if you only have the user-agent, you can use this method to get the device
    //val device = wurflWrapper.deviceForRequest(userAgent)

    println("Device id: " + device.id)

    val wireless = device.capability("is_wireless_device")
    println("Is wireless: " + wireless.get)

    val formFactor = device.virtualCapability("form_factor")
    println("Form factor: " + formFactor.get)

    // loop through all capabilities, then compute virtual capabilities and print them out
    println("CAPABILITIES: ")
    println("----------------------------------------------------------------------")
    val capabilities = device.capabilities
    // foreach can receive a tuple (in this case key value one and two)
    capabilities.foreach { kv => println(kv._1 + " - " + kv._2) }
    println("VIRTUAL CAPABILITIES: ")
    device.virtualCapabilities.foreach { kv => println(kv._1 + " - " + kv._2) }
    println("----------------------------------------------------------------------")

   }
    catch {
      case ex: WURFLRuntimeException => println("Could not initialize  WURFL engine or updater: " + ex.getMess
age)
      case ex: Exception => println("An unexpected exception occurred: " + ex.getMessage)
    }
   finally {
    println("Stopping updater")
    if (updater != null){
      updater.stopPeriodicUpdate()
    }
   }
  }
}
```

> ***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}
> ```

### Accessing ScientiaMobile's private Maven repository using SBT

SBT is the de facto standard for building and packaging Scala applications. Many Scala developers prefer it over the classic Maven client to access Maven repositories too.

You can access ScientiaMobile's private Maven repository using SBT 1.5.0 or above, using this configuration.

1. Create a file called repositories under <user_home>/.sbt and put this content inside it:

```
[repositories]
local
maven-local
maven-central
my-ivy-proxy-releases: https://maven.scientiamobile.com/repository/wurfl-onsite/, [organization]/[module]/(scala_[
scalaVersion]/)(sbt_[sbtVersion]/)[revision]/[type]s/[artifact](-[classifier]).[ext]
my-maven-proxy-releases: https://maven.scientiamobile.com/repository/wurfl-onsite/
```

This will configure a list of repositories: your local one, Maven Central and ScientiaMobile's private repo.

2. Then create a file called credentials under the same directory and copy this content inside it, replacing the asterisks (***) with your actual repository credentials:

```
realm=Sonatype Nexus Repository Manager
host=maven.scientiamobile.com
user=***
password=***
```

3. The last step is to add a reference to the credentials file to your build file with this line:

```
credentials += Credentials(Path.userHome / ".sbt" / "credentials")
```

You can now use the usual sbt update, sbt compile commands on your project and download the WURFL API dependencies.