scientiamobile

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

# ONSITE .NET 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 .NET API: User Guide

## 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.**

The Wurfl.dll file must be added as a reference to any WURFL project, while the Wurfl.Aspnet.Extensions.dll file must be referenced only in ASP.NET projects where you plan to use WURFL. For example, you don't strictly need to reference Wurfl.Aspnet.Extensions.dll if you're using WURFL from within a Console Application.

> ***Note:*** *As of version 1.12.11.0 we support :*

1. .NET Framework 4.5.2, 4.6.2, 4.7.2, 4.8
2. .NET Core 3.1
3. .NET 5.0, 6.0, 7.0

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

### WURFL OnSite .NET API NuGet package

WURFL OnSite .NET API is available as a NuGet package too. To install, use the ScientiaMobile NuGet URL https://nuget.scientiamobile.com/repository/wurfl-onsite/. To browse from VisualStudio:

1. Go to **Tools -> NuGet Package Manager -> Package Manager Settings**
2. Then go to **Package Sources** and add a new Package Source
3. Give the new Package Source a **Name and Source URL** (https://nuget.scientiamobile.com/repository/wurfl-onsite/)
4. Next, go to **Tools -> NuGet Package Manager -> Package Manager Console** and run the following command:
   Install-Package WURFLOnSite (Install-Package WURFLOnSite.NETCore for .NET Core projects)
5. Once the command is executed, **enter your ScientiaMobile account credientials**

## Getting Started

WURFL OnSite .NET API bases its operations on two main objects:

- a **WURFL Manager** object implementing the IWURFLManager interface.
- a **Device** object implementing the IDevice interface.

The **WURFL Manager** object should be instantiated only once in your application.

The **WURFL Manager** object offers several methods (among others) for you to gain access to the in-memory representation of the **Device Definition Repository (DDR)**.

```
public interface IWURFLManager
{
   .
   .
   IDevice GetDeviceForRequest(String userAgent);
   IDevice GetDeviceForRequest(HttpRequest request);
   IDevice GetDeviceById(String deviceId);
   .
   .
}
```

All of these methods return a **Device** object (implementing the IDevice interface) which represents the matched device model.

The **Device** object offers several methods (among others) for you to access the matched device data.

```
public interface IDevice
{
   .
   .
   String GetCapability(String name);
   String GetVirtualCapability(String name);
   IDictionary<String, String> GetCapabilities();
   IDictionary<String, String> GetVirtualCapabilities();
   String Id { get; }
   .
   .
}
```

In the next sections we'll see a sample Console Application as well as an ASP.NET Web based application , using device detection and accessing WURFL static capabilities and virtual capabilities.

## Console Application Usage

In your Console Application project, add the Wurfl.dll assembly as a reference.

> ***Note:*** *Beginning with version 1.8.4, the System.Web assembly must be referenced, even if you are building a Console Application.*

Add a new class named **WURFLSimpleTest** to your project with the following code.

```
using WURFL;
using WURFL.Config;

namespace YourNameSpace
{
   class WURFLSimpleTest
   {
      static void Main(string[] args)
      {
```

Create the InMemoryConfigurer object setting the WURFL data file path;

```
         try
         {
            InMemoryConfigurer configurer = new InMemoryConfigurer()
               .MainFile("PATH_TO_YOUR_WURFL.ZIP");

            IWURFLManager manager = null;
```

Create the WURFL manager once, then lookup the UserAgent, and get the Device-Id, Static Capabilities, and Virtual Capabilities needed in your implementation (beware, Virtual Capabilities are calculated at runtime).

For further details on Virtual Capabilities, click here

```
            manager = WURFLManagerBuilder.Build(configurer);

            String ua = "Dalvik/1.6.0 (Linux; U; Android 4.3; SM-N900T Build/JSS15J)";

            IDevice device = manager.GetDeviceForRequest(ua);

            Console.WriteLine("Device : {0}", device.Id);

            String capName = "brand_name";
            Console.WriteLine("Static Capability {0}: {1}", capName, device.GetCapability(capName));

            String vcapName = "is_android";
            Console.WriteLine("Virtual Capability {0}: {1}", vcapName, device.GetVirtualCapability(vcapName));
```

You can request a full list of Static and Virtual Capability name and values from the device instance.

```
            Console.WriteLine("--- Device Static Capabilities ---");
            foreach (KeyValuePair<string, string> dCap in device.GetCapabilities())
```

```
            Console.WriteLine("[{0}] = [{1}]", dCap.Key, dCap.Value);

        Console.WriteLine("--- Device Virtual Capabilities ---");
        foreach (KeyValuePair<string, string> vCap in device.GetVirtualCapabilities())
            Console.WriteLine("[{0}] = [{1}]", vCap.Key, vCap.Value);
    }
```

WURFL will throw Exceptions in case of failure during the entire process

```
    catch (Exception e)
    {
        Console.WriteLine("WURFLSimpleTest throws this exception : {0} - {1}", e.GetType(), e.Message);
    }
  }
 }
}
```

**Passing the entire HTTP request for device detection**

In addition to passing a User-Agent string to the WURFL API as shown in the example above, you can also pass a HTTP request for device detection. This is useful in cases where the HTTP request contains critical device information in places other than in the User-Agent header. This is most commonly seen in HTTP requests with User-Agent Client Hints.

You can either pass the HTTP request directly toIDevice GetDeviceForRequest(HttpRequest request) or instead pass a dictionary of the headers from the HTTP request:

```
    IDictionary<String, String> requestHeaders = new Dictionary<String, String>
    {
        {"User-Agent", "Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100
.0.0.0 Mobile Safari/537.36"},
        {"Sec-Ch-Ua", "\"Not_A Brand\";v=\"8\", \"Chromium\";v=\"120\", \"Google Chrome\";v=\"120\""},
        {"Sec-Ch-Ua-Platform", "Android"},
        {"Sec-Ch-Ua-Platform-Version", "13.0.0"},
        {"Sec-Ch-Ua-Model", "Pixel 6"},
        {"Sec-Ch-Ua-Mobile", "?1"},
        {"Sec-Ch-Ua-Full-Version-List", "\"Not_A Brand\";v=\"8.0.0.0\", \"Chromium\";v=\"120.0.6099.43\", \"Goo
gle Chrome\";v=\"120.0.6099.43\""},
        {"Sec-Ch-Ua-Arch", ""}
    };
    IDevice device = manager.GetDeviceForRequest(requestHeaders);
```

You can then request WURFL capabilities as shown in thesection above.

> ***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}
> ```

**Static Capability filtering**

In order to reduce memory usage and increase performance, you can specify a subset of the 500+ WURFL static capabilities that will be held by the **WURFL manager** object.

You can set capability filters as follows:

```
configurer.SelectCapabilities(new String[] { "device_os", "is_tablet" });
```

> ***Note:*** *In this case you will be able to access only the* `device_os` *and* `is_tablet` *Static Capabilities values of the detected devices. Looking for other Static Capabilities than whose filtered, will return an empty string.*

**WURFL Cache**

The WURFL manager has an LRU in-memory cache to preserve the result of previous detection.

If you want to enable the LRU cache, you can do it in InMemoryConfigurer object passing it the cache size:

```
configurer.SetCacheProvider(100000);
```

**WURFL Updater**

For API versions 1.8.1.1 and greater, you can keep your **wurfl.zip** file uptodate with Scientiamobile's data release schedule using the **WURFL Updater**.

To configure WURFL Updater, you will need your personal WURFL Snapshot URL (found in the Scientiamobile customer Vault). You may configure the frequency for update checks.

Begin by adding the Wurfl.Updater namespace to your application.

```
using Wurfl.Updater;
```

Then, create a WURFLUpdater instance passing it the manager instance and your updater url

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

**Note:** the path of the **wurfl.zip** specified in the configurer at the moment of **WURFL Manager** creation must be writable from the process/task, and a **wurfl.zip** file must already be present in order for the Updater to determine whether or not it needs to update.
that is executing the .NET API, since WURFLUpdater will update the file denoted by its path.

There are two options in which you can invoke the updater. - using the **PerformUpdate()** method which performs a single update check and then stop.

```
updater.PerformUpdate();
```

- using the **PerformPeriodicUpdate()** method which performs update checks with a periodicity you can specify with the **SetFrequency(frequency)** method, chosing among DAILY WEEKLY (default is DAILY).

```
updater.SetFrequency(Wurfl.Updater.Frequency.WEEKLY);
updater.PerformPeriodicUpdate();
```

If you want to stop Periodic Updates, invoke the **StopPeriodicUpdate()** method

```
updater.StopPeriodicUpdate();
```

> ***Note:*** *The* ***WURFL Updater*** *will check to see if a new version of the wurfl.zip has been released and, if so, download it and reload the* ***WURFL manager*** *with the new version; all while the* ***WURFL manager*** *still running and serving requests.*

**Dependencies for .NET Core projects**

If your project uses .NET Core and you are manually referencing the Wurfl.dll assembly, you may need to add a PackageReference for Microsoft.AspNetCore.Http.Abstractions:

```
<ItemGroup>
   <PackageReference Include="Microsoft.AspNetCore.Http.Abstractions" Version="2.2.0" />
</ItemGroup>
```

If you use nuget to integrate the WURFL API into your project, you can safely skip this section.

## ASP.NET Web Application Usage

In your ASP.NET project, add as a reference both theWurfl.dll and Wurfl.Aspnet.Extensions.dll assemblies.

In your App_Code folder create a class WurflSampleASPNETApp which will hold the WURFL manager instance used for lookups.

```
using WURFL;

public static class WurflSampleASPNETApp
{
   public static IWURFLManager WurflManager;
}
```

In an ASP.NET Web application, the Application_Start method for your Global.asax file is the place where all one-off initializations will be performed. Here you can instruct the method to initialiaze the WurflManager instance.

```
 .
 .
<%@ Import Namespace="WURFL" %>
<%@ Import Namespace="WURFL.Aspnet.Extensions.Config" %>
 .
 .
 .
 .

 private void Application_Start(Object sender, EventArgs e)
 {
  try
  {
     WurflSampleASPNETApp.WurflManager = WURFLManagerBuilder.Build(new ApplicationConfigurer());
  }
  catch (Exception ex)
  {
     HttpRuntime.UnloadAppDomain();
     initializationError = ex;
  }
 }
 .
 .
 .
 .
```

The WURFL configuration should be placed in yourweb.config file, adding the following directives:

```
<wurfl>
  <mainFile path="~/App_Data/wurfl.zip" />
</wurfl>
```

This instructs the WurflSampleASPNETApp.WurflManager initialization to look for the *wurfl.zip* file in your application's **App_Data** folder.

The <wurfl> section is user-defined and needs to be registered before use. For this reason, you also need to add the following at the top of your web.config file:

```
<configuration>
 <configSections>
    <section name="wurfl" type="WURFL.Aspnet.Extensions.Config.WURFLConfigurationSection,Wurfl.Aspnet.Exte
nsions, Version=1.9.5.0, Culture=neutral" />
 </configSections>
 :
</configuration>
```

With the WURFL manager object instantiated by Application_Start, you are ready to lookup Useragent/Request.

To perform a lookup during your Default.aspx page loading, place the following code in Ã¬ts CodeBehind (the Default.aspx.cs file)

```
.
.
using WURFL;
using WURFL.Aspnet.Extensions.Config;
.
.
public partial class _Default : System.Web.UI.Page
{
  public IDevice wurflDevice;
  public String wurflDeviceId;
  public String wurflDeviceBrandName;
  public String wurflDeviceIsAndroid;

  protected void Page_Load(object sender, EventArgs e)
  {
     /**
     * on page load we populate wurflDevice and wurflDeviceId with wurfl detection results
     **/
     wurflDevice = WurflSampleASPNETApp.WurflManager.GetDeviceForRequest(Request);
     wurflDeviceId = wurflDevice.Id;
     wurflDeviceBrandName = wurflDevice.GetCapability("brand_name");
     wurflDeviceIsAndroid = wurflDevice.GetVirtualCapability("is_android");
  }
}
.
.
```

> ***Note:*** *You can lookup devices either by passing the whole HttpRequest or the simple User-Agent. In this last case, you may use the following code*

```
wurflDevice = WurflSampleASPNETApp.WurflManager.GetDeviceForRequest(Request.UserAgent);
```

> ***Note:*** *Using the whole HttpRequest will result in a more precise device lookup*

Now you can show the lookup result in your Default.aspx file

```
.
.
<body>
   <form id="form1" runat="server">
      <div>
         WURFL device Id = <%= wurflDeviceId %> <br/>
         WURFL device Brand Name = <%= wurflDeviceBrandName %> <br/>
         WURFL device Is Android = <%= wurflDeviceIsAndroid %> <br/>
      </div>
   </form>
 </body>
.
.
```

**Static Capability filtering**

In order to reduce memory usage and increase performance, you can specify a subset of the 500+ WURFL static capabilities that will be held by the **WURFL manager** object.

You can set capability filters in yourweb.config as follows:

```
<wurfl>
   <mainFile path="~/App_Data/wurfl.zip" />
   <filter caps="device_os,is_tablet" />
</wurfl>
```

> ***Note:*** *In this case you will be able to access only thedevice_os and is_tablet Static Capabilities values of the detected devices. Looking for other Static Capabilities than whose filtered, will return an empty string.*

**WURFL Updater**

Since API version 1.8.1.1, if you want to keep your **wurfl.zip** uptodate with Scientiamobile's data release schedule, then you might want to use the **WURFL Updater** features.

To configure the Updater you need to know your personal updater url taken from Scientiamobile customer Vault. You may configure which periodicity (the frequency) you would like for update checks.

To configure the WURFL Updater, add the Wurfl.Updater namespace to your Global.asax file and create a WURFLUpdater instance passing it the **WURFL manager** instance and your updater url

```
  .
  .

<%@ Import Namespace="Wurfl.Updater" %>
  .
  .
  .
  .

  private void Application_Start(Object sender, EventArgs e)
  {
     WurflSampleASPNETApp.WurflManager = WURFLManagerBuilder.Build(new ApplicationConfigurer());
     // remember to modify the url below with your personal WURFL updater url
     WURFLUpdater updater = new WURFLUpdater(WurflSampleASPNETApp.WurflManager, "https://data.scientiam
obile.com/xxxxx/wurfl.zip");
  }
  .
```

> ***Note:*** *the path of the* ***wurfl.zip*** *specified in your web.config must be writable from the process/task*
> *that is executing the .NET API, since WURFLUpdater will update the file denoted by its path.*

You can invoke the updater in two ways:

- using the **PerformUpdate()** method which performs a single update check and then stop.

```
updater.PerformUpdate();
```

- using the **PerformPeriodicUpdate()** method which performs update checks with a periodicity you can specify with the **SetFrequency(frequency)** method, chosing among DAILY WEEKLY (default is DAILY).

```
updater.SetFrequency(Wurfl.Updater.Frequency.WEEKLY);
updater.PerformPeriodicUpdate();
```

If you want to stop the Periodic Update, invoke the **StopPeriodicUpdate()** method

```
updater.StopPeriodicUpdate();
```

> ***Note:*** *The* ***WURFL Updater*** *will check to see if a new version of the wurfl.zip has been released and, if so, download it and reload the* ***WURFL manager*** *with the new version; all while the* ***WURFL manager*** *still running and serving requests.*

## 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 static capabilities that are found in WURFL).

Virtual Capabilities are calculated at runtime; in order to compute its final returned value, a virtual

capability may look at static 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.

To get the value of a virtual capability:

```
var isSmartphone = device.GetVirtualCapability("is_smartphone");
```

The value associated with a virtual capability is always expressed as a string, even when it logically represents a number or a Boolean.

| Variable Name | Type | Description |
|---|---|---|
| **is_app** | boolean | Tells you if the Requesting HTTP Client is an App or not. |
| **is_smartphone** | boolean | This is a virtual capability that will tell you if a device is a Smartphone for some arbitrary (and subject to change) definition of Smartphone by ScientiaMobile.<br><br>The virtual capability returns true or false. Patch files can use the is_smartphone control capability to override the value returned by the virtual capability.<br><br>Control capability is_smartphone can take value default, force_true and force_false. private |
| **is_mobile** | boolean | This is just an ALIAS for is_wireless_device. There's no control capability associated to this virtual capability. private |
| **is_full_desktop** | boolean | This is just an ALIAS for ux_full_desktop. There's no control capability associated to this virtual capability. private |

| Variable Name | Type | Description |
| --- | --- | --- |
| **is_windows_phone** | boolean | Check if device runs any version of Windows Phone OS.<br>This virtual capability relies on the device_os (product_info group) capability. private |
| **is_ios** | boolean | Check if device runs any version of iOS.<br>This virtual capability relies on the device_os (product_info group) capability. private |
| **is_android** | boolean | Check if device runs any version of Android OS.<br>This virtual capability relies on the device_os (product_info group) capability. private |
| **is_touchscreen** | boolean | This virtual capability tells you whether a device has a touch screen. There is no control capability. Mostly an alias for pointing_method == touchscreen (product_info group) capability. private |
| **is_largescreen** | boolean | True if the device has a horizontal and vertical screen resolution greater than 480 pixels. Relies on the resolution_width and resolution_height (display group) capabilities. private |
| **is_wml_preferred** | boolean | True if the device is better served with WML. Capability relies on preferred_markup (markup group). private |
| **is_xhtmlmp_preferred** | boolean | True if the device is better served with XHTML MP (Mobile Profile). Capability relies on preferred_markup (markup group). private |

| Variable Name | Type | Description |
| --- | --- | --- |
| **is_html_preferred** | boolean | True if the device is better served with HTML. Capability relies on preferred_markup (markup group). private |
| **advertised_device_os** | string | This virtual capability will infer the name of the Device OS based on user-agent string analysis (and possibly the analysis of other HTTP headers and WURFL capabilities). private |
| **advertised_device_os_version** | string | This virtual capability will infer the version of the Device OS based on user-agent string analysis (and possibly the analysis of other HTTP headers and WURFL capabilities). private |
| **advertised_browser** | string | This virtual capability will infer the name of the browser based on user-agent string analysis (and possibly the analysis of other HTTP headers and WURFL capabilities). private |
| **advertised_browser_version** | string | This virtual capability will infer the version of the browser based on user-agent string analysis (and possibly the analysis of other HTTP headers and WURFL capabilities). private |
| **form_factor** | enumerable | This virtual capability will return one of the following values that identify a client's form factor: Desktop, Tablet, Smartphone, Feature Phone, Smart-TV, Robot, Other non-Mobile, Other Mobile private |
| **complete_device_name** | string | Concatenates brand name, model name and marketing name (where available) of a device into a single string. private |

| Variable Name | Type | Description |
| --- | --- | --- |
| **is_phone** | boolean | This is a virtual capability that will tell you if a device is a mobile phone .<br><br>The virtual capability returns true or false. Patch files can use the is_phone control capability to override the value returned by the virtual capability.<br><br>Control capability is_phone can take value default, force_true and force_false. private |
| **is_app_webview** | boolean | This virtual capability returns true if a HTTP request is from an app based webview. private |
| **device_name** | string | Concatenates brand name and marketing name of a device into a single string. If marketing name is not available, model name is used instead. private |
| **advertised_app_name** | string | This virtual capability will return the name of the application that generated the User-Agent or the HTTP request. private |
| **is_robot** | boolean | |