scientiamobile

"The first step in a great mobile experience"

INFUZE CSHARP MODULE 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 C# : User Guide

WURFL InFuze for C# is a C# module wrapping the WURFL C API. InFuze for C# exploits the performance of the InFuze C API inside your C# applications without having to write your own binding code.

> ***Note:*** *As of version 1.13.3.0 we support :*

1. .NET Framework 4.5.2 or later
2. .NET Core 2.0, 2.1, 3.1
3. .NET 5.0, 6.0, 7.0, 8.0, 9.0

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.

To enable WURFL InFuze for C# on your application, you must download it from your File Manager. The WurflInFuze.dll file must be added as a reference to any WURFL project.

> ***Warning:*** *For installation on Windows, libwurfl v1.8.3.0 or greater is required.*
> ***Note:*** *A .NET Framework of at least 4.5.2 is required for installation.*

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.

Getting Started

In the next sections we'll see a sample Console Application, using device detection and accessing WURFL static and virtual capabilities.

Console Application Usage

Here is an example Console Application to get started:

```
using WURFLInFuze;

namespace WURFLInFuzeSimpleTest
{
   class Program
   {
      static void Main(string[] args)
      {
```

Create the InMemoryConfigurer object setting the WURFL data file path;

```
         try
         {
            // before creating a Wurfl engine, we download an updated version of the wurfl file in the current directo
ry
            // destination directory must be writable
            // replace this URL with your account snapshot data URL, to avoid error 402
            WURFLManager.WurflDownload("https://data.scientiamobile.com/xxxxx/wurfl.zip", ".");

            InMemoryConfigurer configurer = new InMemoryConfigurer()
               .MainFile("C:\\Program Files\\Scientiamobile\\InFuze\\wurfl.zip");
```

```
IWURFLManager manager = null;
```

Create the WURFL Manager once, then lookup UserAgent and get theDevice-Id, Static Capabilities and Virtual Capabilities required in your implementation (note that Virtual Capabilities are calculated at runtime).

For further information on Static Capabilities and Virtual Capabilities, please refer to the [WURFL capabilities page]

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

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

IDevice device = manager.LookupUserAgent(ua);

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

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

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

If you would like to look up a set of HTTP headers instead of a User-Agent (for example to look up requests with User-Agent Client Hints), you can use LookupHeaders instead of LookupUserAgent as shown below.

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

Dictionary<string, string> headers = new Dictionary<string, string>();
headers.Add("User-Agent", "Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.0.0 Mobile Safari/537.36");
headers.Add("Sec-Ch-Ua", "\"Chromium\";v=\"94\", \"HuaweiBrowser\";v=\"99\", \";Not A Brand\";v=\"99\"");
headers.Add("Sec-Ch-Ua-Full-Version", "99.123.456");
headers.Add("Sec-Ch-Ua-Platform", "Android");
headers.Add("Sec-Ch-Ua-Platform-Version", "12");
headers.Add("Sec-Ch-Ua-Model", "Pixel 6");

IDevice device = manager.LookupHeaders(headers);

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

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

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

You can even ask the device instance for the full list of its Static and Virtual Capabilities as well as its names and values.

```
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);
```

Dispose the device object when you don't need it anymore. This is necessary to release allocated resources for that instance on the WURFL C API side. It sets the C# instance as disposable for the garbage collector.

```
device.Dispose();
```

Dispose the manager object when you don't need it anymore. As for the IDevice object, it is necessary to release allocated resources on the WURFL C API side.

```
            manager.Dispose();
        }
```

WURFL will throw a WURFLException in the case of failure throughout the entire process

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

WURFL Cache

In order to increase performance while processing real HTTP traffic, we suggest setting up an LRU cache. The LRU caching strategy will speed up lookup operations on User Agents that have already been processed by keeping them in a Least Recently Used map. By default the cache will be set to 30000 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 more information, please see LRU Cache Mechanism

If you want to modify the cache size, you can do it in the InMemoryConfigurer object:

```
configurer.SetCacheSize(100000);
```

If you want to disable the cache feature:

```
configurer.DisableCache();
```

WURFL Updater

If you want to keep your wurfl.zip uptodate with Scientiamobile's data release schedule, then you might want to use the Updater feature.

To configure the Updater you need your account snapshot data URL, taken from Scientiamobile Customer Vault. You may configure the periodicity (frequency) you would like for update checks, choosing from two values: Daily and Weekly (the default value is Daily). Use UpdaterSetLogPath to set updater log path where you will find detailed logs on Updater activity. Do note that a wurfl.zip file must already be present in a writable path in order for the updater to check the file and determine whether or not it needs to update the file.

There are two ways to configure the WURFL Updater:

- using IWURFLManager instance methods:

```
manager.UpdaterSetDataUrl("https://data.scientiamobile.com/xxxxx/wurfl.zip");
manager.UpdaterSetDataFrequency(WurflUpdaterFrequency.WURFL_UPDATER_FREQ_DAILY);
//manager.UpdaterSetDataFrequency(WurflUpdaterFrequency.WURFL_UPDATER_FREQ_WEEKLY);
manager.UpdaterSetLogPath("C:\\Temp\\updater.log");
// start the updater explicitly
manager.UpdaterStart();
```

- using InMemoryConfigurer object:

```
configurer.UpdaterUrl("https://data.scientiamobile.com/xxxxx/wurfl.zip");
configurer.UpdaterFrequency(WurflUpdaterFrequency.WURFL_UPDATER_FREQ_DAILY);
//configurer.UpdaterFrequency(WurflUpdaterFrequency.WURFL_UPDATER_FREQ_WEEKLY);
configurer.UpdaterLogpath("C:\\Temp\\updater.log");
```

Note: *When using the InMemoryConfigurer object, you don't need to explicitly start the Updater. It will be done automatically when the IWURFLManager instance is created.*

IMPORTANT - Decommissioning of WurflMatchMode options

Prior to version 1.9 of the API, users could choose betweenWurflMatchMode.Performance and WurflMatchMode.Accuracy 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 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.

Customers who may find themselves in the unlikely situation of having to analyze significant amounts of legacy web traffic, may still enable the old WurflMatchMode.Performance behavior by set WurflMatchMode.FastDesktopBrowserMatch in their configuration.

Please note that users with the oldWurflMatchMode.Performance target engine will not receive an error.
The old behavior will not be triggered, though. TheWurflMatchMode.Default target (corresponding to the old WurflMatchMode.Accuracy) will be used instead.