Internet Reachability Verifier for Unity

Helper class for verifying actual connectivity to Internet, from Jetro Lauha / Strobotnik Ltd.

Table of Contents

Introduction

This package contains the Internet Reachability Verifier ("IRV") component, a helper for verifying that the internet can be truly reached. This is done with a technique called "captive portal detection", which is used by operating systems e.g. to determine if they should present a network login screen.

The Internet Reachability Verifier offers several methods modeled after the way how big companies implement the captive portal detection (such as Google, Microsoft, Apple and Ubuntu). You also have an option to use a custom way based on your own web hosting.

The main use-case is to get notification when you can do further WWW requests, for example to download extra content or to send scores to a highscore server. If you're making a networked real-time multiplayer game or so, then equivalent functionality might be already available in the networking library you're using.

HINT – Also check out this component's official web page: www.strobotnik.com/unity/internetreachabilityverifier/

Usage

Usage is quite simple:

1. **Create a new empty GameObject**.
   Recommendation: Add it to the scene which is loaded first.
2. **Drag the InternetReachabilityVerifier component to your new GameObject**.
   This also enables DontDestroyOnLoad on it, so the object with the verifier component will persist if you load a new scene.
3. When needed, **check internet access status like this**:
   ```
   bool netVerified = (InternetReachabilityVerifier.Instance.status == InternetReachabilityVerifier.Status.NetVerified);
   ```

In most cases you do not need to change the default settings of the component, except if you want to use Internet Reachability Verifier with a web-based platform, or if you want to use the custom method (read the relevant chapters).

****** However, if you are maintaining your own web server for the network features, it's recommended that you use the custom method! ******

See the example scenes and scripts for more details:

- `SkeletonIRVExample` – very **simple example**, including status change listening
- `IRVExample` – **main example** with some GUI to make test www fetches, possibility to force re-verification and see a log of what's happening.
- `CustomIRVExample` – example for using **custom method and custom verification**.

Important Note: Internet Reachability Verifier helps you to verify that internet connections should work. After you start networking, ***you still need to have code which checks for error situations (for example if network connection drops away right in middle of transferring data)***. However, when you detect a network error situation, you can ask IRV to re-verify the internet connection. For an example, see `void forceReverification()` in *SkeletonIRVExample.cs*.

Details

Below you can find more information about fields and methods of the class, the different "Captive Portal Detection" methods and more detailed explanation of what different fields of the `InternetReachabilityVerifier.Status` enum mean.

Extra Fields and Methods

`InternetReachabilityVerifier.Status status`

You can check the current `InternetReachabilityVerifier` status using this property.

`string lastError`

This contains the last error response string in case of WWW error when trying to verify internet access.

`static InternetReachabilityVerifier Instance`

Easy access to a common static instance of `InternetReachabilityVerifier`.

`float getTimeWithoutInternetConnection()`

Returns how long app has been without internet connection, or 0 when online (time in seconds).

`IEnumerator WaitForNetVerifiedStatus()`

Helper coroutine for waiting until status becomes NetVerified if it isn't already.
If status isn't already `PendingVerification`, will also force reverification first. See `SkeletonIRVExample` for usage example.

`void setNetActivityTimes(...)`

Call this method to use custom time periods instead of the default values.
`defaultCheckPeriodSeconds`
– interval for checking for status changes.
`errorRetryDelaySeconds`
– delay before retrying when encountering error.

`mismatchRetryDelaySeconds`
– delay before retrying on mismatch case (e.g. user needs to login into network)
The above times can also be set in the inspector using the fields `Default Check Period`, `Error Retry Delay` and `Mismatch Retry Delay`.

`void forceReverification()`
Convenience method for requesting that internet access is verified again.

You should call this after your own networking calls start to return errors which indicate effective loss of network connectivity. The Internet Reachability Verifier will not actively monitor if effective networking will suddenly stop working at middle of some operation. However, it will change to offline status if user disables all networking (e.g. by activating airplane/flight mode).

`void Stop()`
Stops the internal activity coroutine. (If you want to stop the verifier for some reason.)

Captive Portal Detection Methods

`DefaultByPlatform`
Use the “native” method for the current platform. In this case it means that if you're running on Google's Android platform, then the `Google204` method will be used, which is also how the Android operating system itself does similar network check. Similarly the `MicrosoftNCSI` and `Apple2` methods are selected for Microsoft's and Apple's platforms. All desktop Linux platforms default to using the `Ubuntu` method.

`Google204`
Google's "HTTP result 204" method used by e.g. Android. For this check method the http status code needs to be read from response headers, and the WWW class implementation of some Unity platforms/versions do not have the response headers (e.g. on iOS). There is a fallback which checks that there is no return data, which is similar to how the `GoogleBlank` method works.

`GoogleBlank`
Google's alternative method, checks that a blank page from google.com is blank. This is forced as a fallback for the `Google204` method on some platforms which cannot check the response headers.

`MicrosoftNCSI`
Microsoft Network Connectivity Status Indicator check. Verifies that the response from the check url is “Microsoft NCSI”.

`Apple`
Apple's method, fetches page from apple.com and checks that it contains "Success".

Apple2

Like the Apple method above, but uses captive.apple.com with random path.

AppleHTTPS

Like the Apple method, but uses HTTPS explicitly (this is the new default and can be used in case of any foreseeable problems with the App Transport Security).

Ubuntu

Ubuntu connectivity check. Returns a page with Lorem ipsum text, a soft check is done to verify that the Lorem ipsum text starts at the expected position.

Custom

Use your own url to fetch for connectivity check. Note that in some countries connectivity to servers of some of the above methods may be limited. For example, some have reported that Google can't always be reached from China. Using the Custom method may be a solution in such cases.

If “Custom Method With Cache Buster” checkbox is enabled, the url is appended with a query string like z=13371337 (the number is randomized). To be considered a success for verifying network connection, the result must start with the Custom Method Expected Data (or result must be empty if the field is empty). Alternatively you can use `customMethodVerifierDelegate` for checking the result. This is the only method available for web-based platforms (e.g. Web player plugin).

InternetReachabilityVerifier.Status enum

Offline

No internet access or network connectivity at all. (Only when Unity's built-in `Application.internetReachability` equals `NetworkReachability.NotReachable`).

PendingVerification

Internet access is being verified by using doing a WWW request to the active captive portal detection method.

Error

Verifying internet access resulted in a WWW error response. (See `lastError`.) Verification will be retried automatically after a delay.

Mismatch

Verification failed – a “captive portal” has been detected. That is, a page was successfully fetched but contained wrong data. This usually means that the user needs to login into network, and there's a good chance that the network will soon be available. Verification will be retried automatically after a delay.

NetVerified

Network & internet access has been verified. *You're online!*

Using Custom Method and Verifier Delegate

If you want, instead of using one of the pre-existing verification methods, you can also host your own verification check. See the CustomIRVExample example scene and script for more details.

Basically using custom method means that you should put a simple helper file with known non-changing contents to a web server. For example, you can use the `internetreachabilityverifier.txt` found in Examples folder, which only contains “OK”. Then you should set the Custom Method URL and Custom Method Expected Data of your `InternetReachabilityVerifier` component.

By default the page returned from Custom Method URL is expected to start with the Custom Method Expected Data. Alternatively you can set up `customMethodVerifierDelegate` which will get the WWW object and `customMethodExpectedData` string, and returns true on success.

If you have the possibility of using and maintaining your own server, it's recommended you use the custom method. See the Questions & Answers section for an explanation.

Using Internet Reachability Verifier with Web-based Platforms

If you want to use `InternetReachabilityVerifier` with the Web player, then your only option is to use the Custom method. Note that you might not need the verifier at all in this case – if you always load the game from a www page, then implicitly the user's internet connection already works, and there's less need to verify it.

The easiest way to get the verifier working with web player is to have the Custom Method URL point to a file in the **same server** which is hosting the game build (file with “`.unity3d`” extension in case of web player).

Alternatively you can use a `crossdomain.xml` file in the server where your file is, pointed by Custom Method URL. Read more in the *Implications for use of the WWW class* chapter in the *Security Sandbox of the Webplayer* page in Unity manual:
http://docs.unity3d.com/Manual/SecuritySandbox.html

Questions and Answers

But Unity has Application.internetReachability, isn't it enough?

> The built-in `Application.internetReachability` only tells if it is technically possible to try to open a network connection. This is not very helpful e.g. on desktop platforms where it will *always* return `NetworkReachability.ReachableViaLocalAreaNetwork`.

How do I know if the network is using carrier data?

When the status of `InternetReachabilityVerifier` is `NetVerified`, you can still use `Application.internetReachability` to check if the network is `ReachableViaCarrierDataNetwork` or `ReachableViaLocalAreaNetwork`.

I checked a WiFi at an airport and this software said the network is verified although I only reached the network login page without logging in. How is this possible?

Yes, this is possible. Unfortunately some captive portal software used to host Wi-Fi networks actually know about the common captive portal detection methods and they are configured to serve those checks properly while redirecting all other WWW requests to their login page. This is one reason why it's recommended you use the custom detection method. This way the detection is done with your custom URL, and that won't be listed as a special case in the captive portal software like the publicly known methods might be, which is why using a custom method can be more reliable. However, with custom method the onus is on you to keep the server always running (which contains the helper file), since all of your internet reachability checks will start failing if the server is down.

Feedback, Feature Suggestions and Bug Reports

Send by email: contact@strobotnik.com. Write “InternetReachabilityVerifier” in the subject.

Also from Strobotnik:

Google Universal Analytics for Unity
http://strobotnik.com/unity/googleuniversalanalytics/

Pixel-Perfect Dynamic Text
http://strobotnik.com/unity/dynamictext/