Internet Reachability Verifier for Unity® version **1.2**

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

Table of Contents

1 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", that is also used by operating systems 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 custom method is recommended).

The main use-case is to get notification when you can do further http/https 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/

2 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.
You may want to use the “custom” method though (read the relevant chapters).

****** However, if you are using your own web server for additional network-based features, it's <u>recommended</u> that you use the “custom” method with your own URL! ******

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 http(s) fetches, possibility to force re-verification, and 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 do detect a network error situation, you can ask IRV to re-verify the internet connection. For an example, see `void forceReverification()` in `SkeletonIRVExample.cs`.

3 Details About Fields, Methods and Captive Portal Detection Methods

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.

3.1 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 http request error when trying to verify internet access.

`static InternetReachabilityVerifier` **`Instance`**

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

`float` **`getTimeWithoutInternetConnection()`**

Returns how many seconds app has been without internet connection, or 0 when online.

`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.cs`** 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 that 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.)

3.2 Captive Portal Detection Methods

`DefaultByPlatform`

Selecting this means that when IRV initializes, it is instructed to switch to a “native” method for the current platform. For example, if your app is 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 `AppleHTTPS` 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. (Warning: some old Unity versions lack response headers, e.g. on iOS. For those legacy cases there is a fallback which checks that there is no return data, which is similar to how the `GoogleBlank` method works.)

`Google204HTTPS`

Like Google204 method above, but uses HTTPS explicitly. Successful result with a https-based url does not guarantee that plain http connections are usable.

`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 and old Unity versions which cannot check the response headers.

`MicrosoftNCSI`

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

`MicrosoftNCSI_IPV6`

IPV6 version of the MicrosoftNCSI method. This one only works in IPV6-capable networks, and it will always fail otherwise.

`Apple`

Apple's method, fetches page from apple.com and checks that html 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 for Apple platforms, and can be used in case of any foreseeable problems with the App Transport Security. Successful result with a https-based url does not guarantee that plain http connections are usable.

`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.

`UbuntuHTTPS`

Like the Ubuntu method, but uses HTTPS explicitly. Successful result with a https-based url does not guarantee that plain http connections are usable.

`MicrosoftConnectTest`

Microsoft's alternative connection tester. Note: this one seems to set `Access-Control-Allow-Origin` header in server-side. Because of that, this is used as the default method for normal WebGL platform, since that header is required by browsers.

`MicrosoftConnectTest_IPV6`

IPV6 version of the MicrosoftConnectTest method. This one only works in IPV6-capable networks, and it will always fail otherwise.

`Custom`

Using this method is **recommended**. It lets you to use your own **http** or **https** url to fetch for connectivity check. Successful result with a https-based url does not guarantee that plain http connections are usable.

Note: in some countries connectivity to servers of some of the other detection methods may be limited, which is why using the Custom method may be the best solution. For example, some have reported that Google servers can't always be reached from China.

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.

If you want to use Custom method with WebGL platform, your server must set a correct `Access-Control-Allow-Origin` header. Additionally on Facebook with WebGL you are required to use an **https** url. With a web-based platform it is strongly recommended that you also enable the Cache Buster (described above), since browsers may otherwise cache a successful request.

This is also the only method available for legacy web-based platforms (e.g. Web player plugin), which are also only usable with legacy versions of Unity.

3.3 InternetReachabilityVerifier.Status enum

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

`PendingVerification`
: Internet access is being verified by doing a http(s) request to the currently active captive portal detection method.

`Error`
: Verifying internet access resulted in a http 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!*

4 Using Custom Method

The following sub-chapters explain more details about using the Custom method.

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 some additional information.

Also, if you use WebGL with custom method (or Facebook with WebGL), be sure to read the chapter 5.

4.1 Using Custom Method with 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 text file `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 a request object and `customMethodExpectedData` string, and returns true on success. (Type of the request object is UnityEngine.WWW, or UnityEngine.Networking.UnityWebRequest if you're using Unity 2018.3+ or you edited IRV source code to manually enable `IRV_USE_WEBREQUEST`).

4.2 Using Custom Method on Android Platforms

Starting on Android version 9.0 "Pie" (API level 28), there's a new change in the network security configuration model: by default cleartext (http) support is disabled. However, there's a way to make exceptions: you can create a `network_security_config.xml` resource which is referred to in the application manifest file.

IRV already includes a pre-configured AAR file just for that purpose. It's located in: `InternetReachabilityVerifier/Plugins/Android/`**`InternetReachabilityVerifier_NetworkSecurityConfig.aar`**
This included `.aar` file contains network security config which allows cleartext (http) traffic for all of the domains needed by the supported 3rd-party captive portal detection methods. (Note that the domain which demonstrates use of custom method is not included in that aar file, as that example url is not meant to be deployed in any production release builds).

If you want to use the Custom detection method on Android with an https url, then it should work just “as is”. But, if you want to use the Custom method using an http url, then you need to create your own `network_security_config.xml` resource, which has an exception for your domain. In that case you need to read the following instructions for making your own network security config.

Here’s the required steps for making your own network security config for Android 9.0+:

1. Create Android Studio project for Android Library.
   More info: https://developer.android.com/studio/projects/android-library
2. Include a new `res/xml/network_security_config.xml` file. Below is an example of its contents (be sure to edit the applicable places).

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <domain-config cleartextTrafficPermitted="true">
    <!-- If custom captive portal detection url for Internet Reachability Verifier
         would be "http://example.test/IRV-check.txt", then you would put
         example.test below in place of YOUR.OWN.DOMAIN.HERE. -->
    <domain includeSubdomains="false">YOUR.OWN.DOMAIN.HERE</domain>
  </domain-config>
</network-security-config>
```

   More info: https://developer.android.com/training/articles/security-config
3. If needed, add `AndroidManifest.xml` to your new library, or modify the existing one, and add reference to `network_security_config`:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="YOUR.APPLICATION.PACKAGE.NAME">
    <application android:networkSecurityConfig="@xml/network_security_config" />
</manifest>
```

If you already have a `network_security_config.xml` resource in an Android project, you can add a `domain-config` block to it manually, instead of using the provided AAR file or doing the above steps. See step 2 in the above example for contents of the `domain-config` block.

4.3 Using Custom Method on Apple Platforms

Apple has transformed to new security model called App Transport Security (ATS). By default apps can only use do https web requests and http is disallowed. Because of this IRV uses the AppleHTTPS method as the default on Apple platforms. With these platforms all of your own web-based network traffic should also use https urls. Also your custom method url should be an https url.

Alternatively you can try to configure an exception to ATS. Read more info from here:
https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html
Specifically, you could add NSExceptionDomains (to NSAppTransportSecurity). If you use custom method, configure the exception for your own domain. (In theory, you could also pick

one of the existing detection methods, check its domain from IRV source getCaptivePortalDetectionURL-method, and add that domain as the exception.)

Note that it is quite possible that apps using these exceptions will get rejected in the Apple App Store Review process. Because of this, using exceptions is not a recommended way, and that's why there's also no effort in IRV to include ready-made exceptions for certain domain(s). Trying to configure exceptions on Apple platforms should only be done when you know it is the only possible way for you and you can give a solid explanation why an exception is required (in case you're asked for such an explanation in the above mentioned review process).

5 Using Internet Reachability Verifier with Web-based Platforms

Note that using IRV with a web-based platform may not be needed at all: since the user is already able to access your app using the web, you know the user is already online. But if you still choose to use IRV on a web-based platform, you can do so.

Using IRV with WebGL

You need to use a detection method which properly sets `Access-Control-Allow-Origin` header. For now the only built-in method which sets the header is MicrosoftConnectTest, which is set by DefaultByPlatform method on WebGL. Note that if you use Facebook platform with WebGL, then you must use custom method with an https url (there is no https variant of MicrosoftConnectTest method).

Configuring your own web servers is beyond the scope of this asset or support requests, but nevertheless, you may find useful this list of overall steps to configure Apache 2.4 server. Note that you need to have administrator / root access, and you must have understanding of security implications yourself.

Configuration hints for Apache 2.4 server and Access-Control-Allow-Origin header:

1. Your Apache config has `<Directory /.../...>` entry for top-level path where your custom method url points. Add `AllowOverride FileInfo` (or `AllowOverride All`) inside that directory entry. This is needed so that a "`.htaccess`" file can use `Header` directive.
2. Enable "headers" module in Apache if it's not enabled yet: (enter command in console)
   `# a2enmod headers`
3. Make a separate directory under the previously mentioned top-level path.
4. Add new "`.htaccess`" file there, with following contents:

```
<IfModule mod_headers.c>
    Header set Access-Control-Allow-Origin "*"
</IfModule>
```

Better option would be to allow only a certain site instead of `"*"` (any site).

That list is only meant to give you an idea of what is needed. If you need more information, you should search the internet. It is up to you to know how to maintain your servers.

Legacy Information for Unity 3 and 4, for Unity Web Player Plugin and Flash

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~~ https://web.archive.org/web/20160709224207/http://docs.unity3d.com/Manual/SecuritySandbox.html

6 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.

If user was already verified to be online, why going offline is not detected?

Note that “going offline” is detected when app user disables all networking (e.g. by activating airplane/flight mode). However, other than that, IRV will not actively monitor the current quality of internet connectivity. You’re supposed to add code to your own networking, which checks for error situations. For example, if network connection drops away right in middle of transferring data. After you detect such an error, you can always then ask IRV to verify the network connection again.

Moreover, it’s important to note the difference in trying to connect when being offline (zero traffic until user is online), and monitoring connectivity when already online (requires constant traffic). Such constant “unnecessary” traffic would be impolite towards end-users, because in many places users still have to pay by amount of used traffic.

7 Feedback, Feature Suggestions and Bug Reports

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

Also from Strobotnik for Unity®:

NEW: ***Klattersynth TTS***
http://strobotnik.com/unity/klattersynth/

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

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