Work in progress

The content of this page was not yet updated for Godot `4.5` and may be **outdated**. If you know how to improve this page or you can confirm that it's up to date, feel free to open a pull request.

HTTP client class

HTTPClient provides low-level access to HTTP communication. For a higher-level interface, you may want to take a look at HTTPRequest first, which has a tutorial available here.

Warning

When exporting to Android, make sure to enable the `INTERNET` permission in the Android export preset before exporting the project or using one-click deploy. Otherwise, network communication of any kind will be blocked by Android.

Here's an example of using the HTTPClient class. It's just a script, so it can be run by executing:

GDScript C#

```
c:\godot> godot -s http_test.gd
```

It will connect and fetch a website.

GDScript C#

```
extends SceneTree

# HTTPClient demo
# This simple class can do HTTP requests; it will not block, but it needs
to be polled.

func _init():
    var err = 0
    var http = HTTPClient.new() # Create the Client.

    err = http.connect_to_host("www.php.net", 80) # Connect to host/port.
    assert(err == OK) # Make sure connection is OK.

    # Wait until resolved and connected.
    while http.get_status() == HTTPClient.STATUS_CONNECTING or
http.get_status() == HTTPClient.STATUS_RESOLVING:
        http.poll()
        print("Connecting...")
        await get_tree().process_frame

    assert(http.get_status() == HTTPClient.STATUS_CONNECTED) # Check if the
connection was made successfully.

    # Some headers
    var headers = [
        "User-Agent: Pirulo/1.0 (Godot)",
        "Accept: */*"
    ]

    err = http.request(HTTPClient.METHOD_GET, "/ChangeLog-5.php", headers)
# Request a page from the site (this one was chunked..)
    assert(err == OK) # Make sure all is OK.

    while http.get_status() == HTTPClient.STATUS_REQUESTING:
        # Keep polling for as long as the request is being processed.
        http.poll()
        print("Requesting...")
        await get_tree().process_frame

    assert(http.get_status() == HTTPClient.STATUS_BODY or http.get_status()
== HTTPClient.STATUS_CONNECTED) # Make sure request finished well.

    print("response? ", http.has_response()) # Site might not have a
response.

    if http.has_response():
        # If there is a response...
```

```
        headers = http.get_response_headers_as_dictionary() # Get response
headers.
        print("code: ", http.get_response_code()) # Show response code.
        print("**headers:\\n", headers) # Show headers.

        # Getting the HTTP Body

        if http.is_response_chunked():
            # Does it use chunks?
            print("Response is Chunked!")
        else:
            # Or just plain Content-Length
            var bl = http.get_response_body_length()
            print("Response Length: ", bl)

        # This method works for both anyway

        var rb = PackedByteArray() # Array that will hold the data.

        while http.get_status() == HTTPClient.STATUS_BODY:
            # While there is body left to be read
            http.poll()
            # Get a chunk.
            var chunk = http.read_response_body_chunk()
            if chunk.size() == 0:
                await get_tree().process_frame
            else:
                rb = rb + chunk # Append to read buffer.
        # Done!

        print("bytes got: ", rb.size())
        var text = rb.get_string_from_ascii()
        print("Text: ", text)

    quit()
```

Previous Next

User-contributed notes

Loading comments...