GEUTEBRÜCK

G-Core Web API Documentation

Version: 2.0

04.01.2024

Table Of Contents

About This Documentation

Current software version: **G-Core Web API 2.0**.

Legal Notice

This documentation may not be copied, translated or converted to a machine-readable form, whether in whole or in part, without prior permission.

GEUTEBRÜCK GmbH cannot guarantee the correctness of any information provided in this documentation, nor for the software or the information it contains. Any suggested guarantee, assurance of marketable quality or suitability for a specific purpose of the documentation, the software or other information is hereby explicitly rejected.

Under no circumstances is GEUTEBRÜCK GmbH liable for direct or indirect subsequent damage or for special subsequent damage resulting from or in association with this documentation, regardless of whether this arises as a result of illegitimate action, of a contract, or for other reasons in association with this documentation, the software or of the information contained or used within it.

GEUTEBRÜCK GmbH retains the right to change this documentation or the information contained within it at any time without warning. The software described in it is subject to the conditions of a special license contract.

About the Web API

The G-Core Web API allows you to connect to G-Core through a set of platform-independent APIs. The API is designed to provide easy access to the core G-Core functions, such as video streams and actions.

The G-Core Web API is especially helpful to connect web-based services with your G-Core VMS. For native windows applications, you can choose between our C++/C# based SDK the G-Core Web API.

Installation

System Requirements

The following system requirements are required to install the Web API:

- G-Core 7.0 or newer
- .NET 6.0 runtime or newer
- Licenses (see **Licensing**)

Install the Web API

Install the Web API on the G-Core server.

1. Run the `G-Core_Web_API_installer_xxx.exe` file.
2. Accept the **License Agreement** and click **Next**.
3. In the **Ready to Install** dialog window click **Install**.
4. When the installation is completed, click **Finish**.

After the installation is completed, a running **G-Core Web API service** is displayed in your Windows services.

A self-signed HTTPS certificate is part of the installation. For further information see **HTTPS Certificate**.

Licensing

The G-Core Web API is included free of charge starting with G-Core version 7.0. The following free licenses are required to use the Web API:

License	Description
8.35000 - Web API	Web API license for the basic function - 1x per server
8.35001 - Web API Metadata	Web API Metadata license for metadata access (PLC endpoint) - 1x per server
8.35002 - Web API ChannelConnect	Web API ChannelConnect license for streaming channel access - 1x per channel

All current and new licenses are equipped with the necessary licenses. If you have a dongle without these features, generate a new license file (*.lic or *.slk) in your license portal or request a license file at options@geutebrueck.com.

Upgrade 1.5.x to 2.0

During the upgrade, the existing appsettings.json is renamed to appsettings-backup.json. A new appsettings.json is created with the default values.

Adjust your configuration accordingly. Normally, only the G-Core access data needs to be adjusted if you have not installed the Web API on the same system as G-Core.

HTTPS Certificate

An HTTPS certificate is required to access the Web API and its built-in API documentation.

A self-signed certificate is automatically installed and used during installation (computer certificates). Alternatively, a Let's Encrypt certificate can be used.

Use of Let's Encrypt (ACME Protocol)

Let's Encrypt allows you to manage certificates for a specified domain. Existing certificates are kept up to date by checking the expiration date and a new certificate is installed if none exists.

How to use Let's Encrypt:

1. Install the Web API.
2. Configure your network (see **Network Configuration**).
3. Edit the `Let's Encrypt` section in the appsettings.json file (see **Let's Encrypt Configuration**).
4. Restart the service.
5. Make sure the Let's Encrypt certificate is installed.
6. Add the new certificate to be used to the appsettings.json file (see **Use a Certificate**).
7. Restart the service.
8. Access the Swagger website via the new domain name to make sure the certificate is used (see **HTTPS REST API**).

Network Configuration

To create and manage a TLS certificate for a domain, you must configure your network correctly. The IPv4 listener of the Let's Encrypt module responds to port 13020. The service performs an HTTP-01 challenge to validate the certificate request.

The HTTP-01 challenge can only be performed on port 80. You must therefore configure port forwarding on your internet router / firewall from port 80 to the internal port 13020 of the Web API server.

The server must open port 13020 for this communication.

Let's Encrypt Configuration

To activate the Let's Encrypt module, you must configure the appsettings.json file.

Configure the `LetsEncrypt` section and set the `active` parameter to `true`. An example can look like this:

```
"LetsEncrypt": {
    "active": true,
    "staging": false,
    "user": "anybody.surname@company.com",
    "domain": "sample.dns.net",
    "chronSchedule": "0 0 * * * ?",
    "renewbeforeexpireddays": 30
  },
```

You can configure the following parameters in the `LetsEncrypt` section of the appsettings.json:

Setting	Description
active	To activate the Let's Encrypt module (ACME module), set this parameter to `true`. To use the generated certificate, change the subject of `HttpsInlineCertStore` to the domain name after the certificate has been successfully generated.
staging	Let's Encrypt provides a staging environment for testing purposes. If you set this parameter to `true`, the service communicates with this staging environment. **Do not use this parameter to generate certificates. If you use the staging envir-**

Setting	Description
	onment, an invalid certificate will be created but not installed. To request a valid certificate, set the parameter to `false`. **Note that the number of certificates for a domain name is limited by Let's Encrypt.**
user	The e-mail address of the user requesting a certificate. The module either creates a new account or automatically uses the existing one. The e-mail address is used by the Let's Encrypt service. For further information, see: **https://-letsencrypt.org/docs/expiration-emails/**
domain	The certificate managed by the service is issued for the specified domain. It is also the subject of the certificate when it is installed and is used to identify the certificates in the update routine of the module.
chronSchedule	The service checks the validity of the certificate periodically. The `chronSchedule` parameter defines the time period in which the checks are performed. By default, a check is performed hourly. When the service is started, the certificate is checked once and created or updated if necessary.
renewbeforeexpireddays	The `renewbeforeexpireddays` parameter determines how many days before expiration the certificate is automatically renewed.

Use the staging environment of Let's Encrypt to ensure that all other settings and the network configuration is correct. After that, you can safely deactivate the staging environment. For more information on using Let's Encrypt, see the Web API log files (see Service Log Files**).**

Use a Certificate

To select a certificate, configure the `HttpsInlineCertStore` section in the appsettings.json file. The service loads the certificate once on its starts. To use the certificate, set the `Subject` parameter to the subject of the certificate. The default setting is to load the self-signed certificate.

> **First generate a certificate with Let's Encrypt and then adjust the `Subject` for `HttpsInlineCertStore`. As long as no certificate is found, the Web API service will not start.**

The Let's Encrypt module creates a certificate for the domain name of your system. This domain is also the subject of the generated certificate.

Make sure that the `Url` parameter is not configured for a specific domain. Use `https://*:13333` or `https://0.0.0.0:13333` (port 13333 can be changed to any other port) to listen to any domain of the server. The service is accessible locally with localhost (127.0.0.1) or externally via the configured domain name.

```
"HttpsInlineCertStore": {
   "Url": "https://*:13333",
   "Certificate": {
     "Subject": "sample.dns.net",
     "Store": "My",
     "Location": "LocalMachine"
   }
 },
```

Further Information

Local Port Forwarding

If you need a local port forwarding on the system on which the Web API is installed, use the following cmd command:

```
netsh interface portproxy add v4tov4 listenport=80 listen-
address=0.0.0.0 connectport=13020 connectaddress=127.0.0.1
```

DynDNS

You can use a DynDNS service to get a domain name for your dial-up connections. Most routers support multiple DnyDNS services. DuckDNS.org, for example, is a free and easy-to-use service that you can use for this.

Fritz Box Users

If you activate the MyFritz function of your Fritz!Box, the Fritz!Box has a built-in domain name. This gives you a domain name for your dial-up connection such as: `1234abcd.myfritz.net`.

To be able to use this domain name, you must deactivate IPv6 in the internet connection. Otherwise, the port forwarding from port 80 to IPv4 will not work and will always redirect you to the login page of the Fritz Box.

Why Are HTTPS Certificates Important?

Creating an SSL certificate for an HTTPS connection helps ensure the security and privacy of your content and complies with the cyber security standards.

Complete Encryption of the Transmitted Data

An HTTPS connection with SSL certificate provides an additional level of security for your G-Core system and for the operator. The SSL certificate completely encrypts the data transmitted via the internet to protect sensitive information from interception and manipulation by third parties.

Secure Connection in Web Browsers

HTTPS connections are identified as insecure in almost all web browsers. The Firefox web browser even offers the extended security function "HTTPS-Only Mode". In future, HTTP connections will tend to no longer be permitted or access will become very difficult.

Personal Responsibility and Control of Your Content

By generating your own SSL certificates, you retain personal responsibility and control of your content. This ensure that your content is transmitted securely and according to your specific requirements.

Settings

The main settings for the G-Core Web API service can be found in the `appsettings.json` file in the Web API installation folder (`C:\Program Files\Geutebrueck\GCore Web API`).

Connection

Name	Type	Description	Example
Connection:Address	string	G-Core server IP address.	127.0.0.1
Connection:username	string	G-Core username to authenticate to the Web API service. If empty, the auto login is used.	sysadmin
Connection:password	string	G-Core password to authenticate to the Web API service. If empty, the auto login is used.	masterkey

Auto Login

If the username and password are empty, the G-Core Web API service uses the auto login feature to connect to the G-Core server. The auto login only works with a G-Core installed on the same server as the Web API.

Access the Web API

The Web API is divided into two main parts:

- The **HTTPS REST API** is used for requests like authentication or resources.
- The **WebSocket API** is used for connected requests like video streaming or PLC data.

HTTPS REST API

The HTTPS REST API is documented with Swagger. You can access the documentation via `https://<server-ip>:13333/swagger/index.html` (for example: `https://127.0.0.1:13333/swagger/index.html`) in any modern browser (Chrome, Safari, Firefox, and Edge).

The G-Core Web API is installed as a service and contains a swagger API where developer can get the description of the complete interface (https://-localhost:13333/swagger/index.html).

You can build an HTTP webclient and access these GET and POST commands or you can build a c# application and use the wrapped HTTP function in a class.

WebSocket API

The WebSocket API is using the WebSocket protocol based on a TCP connection.

The respective WebSocket API documentation can be accessed via the following URLs:

- **Streaming:** `https://<server-ip>:13333/asyncapi/media/ui/index.html`
- **PLC:** `https://<server-ip>:13333/asyncapi/plc/ui/index.html`

On these pages you will find the necessary documentation for using the WebSocket API endpoints.

Authentication

Use the authentication endpoint **/api/1/Login** to authenticate with the G-Core Web API. The G-Core username and password are used for authentication.

Authentication in Swagger

You can perform the authentication directly in the Swagger UI. The result is a tuple, which you should subsequently use in any other requests to be authenticated. The tuple consists of an AccessToken and a RefreshToken.

Token	Valitity	Use
AccessToken	Valid for a period of 15 minutes.	For HTTP requests, the token can be used in the Authorization header. For example: "Authorization: Bearer <AccessToken>"
RefreshToken	Valid for 7 days. **Each RefreshToken is only valid once.**	The token can be used to acquire new access tokens via the **/api/1/ RefreshLogin** endpoint, which also returns a tuple of AccessToken and RefreshToken.

The authentication endpoint /api/1/Login is rate limited to a certain number of requests in the default settings. Per client IP there are 100 requests per 10 Minutes allowed. This setting can be configured in the appsettings.json file in the IpRateLimiting section.

In Swagger, enter the token to perform the authentication:

1. In Swagger, click **Authorize** in the top right corner.
2. Enter the word **Bearer** followed by a space and the token you received from the login endpoint (Bearer <AccessToken>).
3. Then you can try any other endpoint in Swagger.

→ This authentication is also required for the WebSocket API.

Authentication of the WebSocket API

There are two supported ways to authenticate the WebSocket API:

- You can set the token in an authorization cookie.

```
ClientWebSocket _webSocket = new ClientWebSocket();
var token = "...";
_webSocket.Options.Cookies = new System.Net.CookieContainer();
_webSocket.Options.Cookies.Add(new Uri(String.Format("ws://
{0}/", host)), new System.Net.Cookie("Authorization", token));
_webSocket.ConnectAsync(
        new Uri(String.Format("ws://{0}/ap-
i/1/stream/video?MediaChannelIdentifier={1}", host, medi-
aChannel)), cancellationToken).Wait();
```

- You can set the token as in the protocol header.

```
this.socket = new WebSocket(url, accessToken);
```

Working with the Web API

JavaScript WebSocket Example

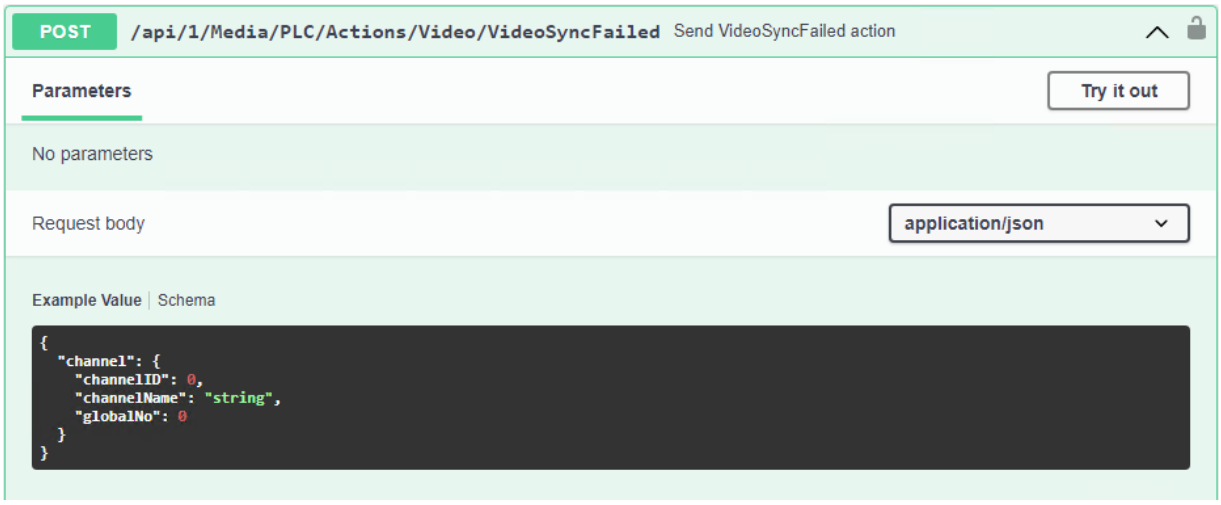
This opens a new stream with the media channel ID number 1.

```
const websocketAddress =
    "ws://<server ip>:13332/ap-
i/1/stream/video?MediaChannelIdentifier=1";
let websocket = new Websocket(websocketAddress);
```

Parameter "Channel"

You can trigger actions in G-Core via the Web API.

The actions can be configured in the Swagger user interface. Many actions are assigned to a media channel. This assignment is made via the `"channel"` parameter:



You must specify one of the following values for the `"channel"` parameter:

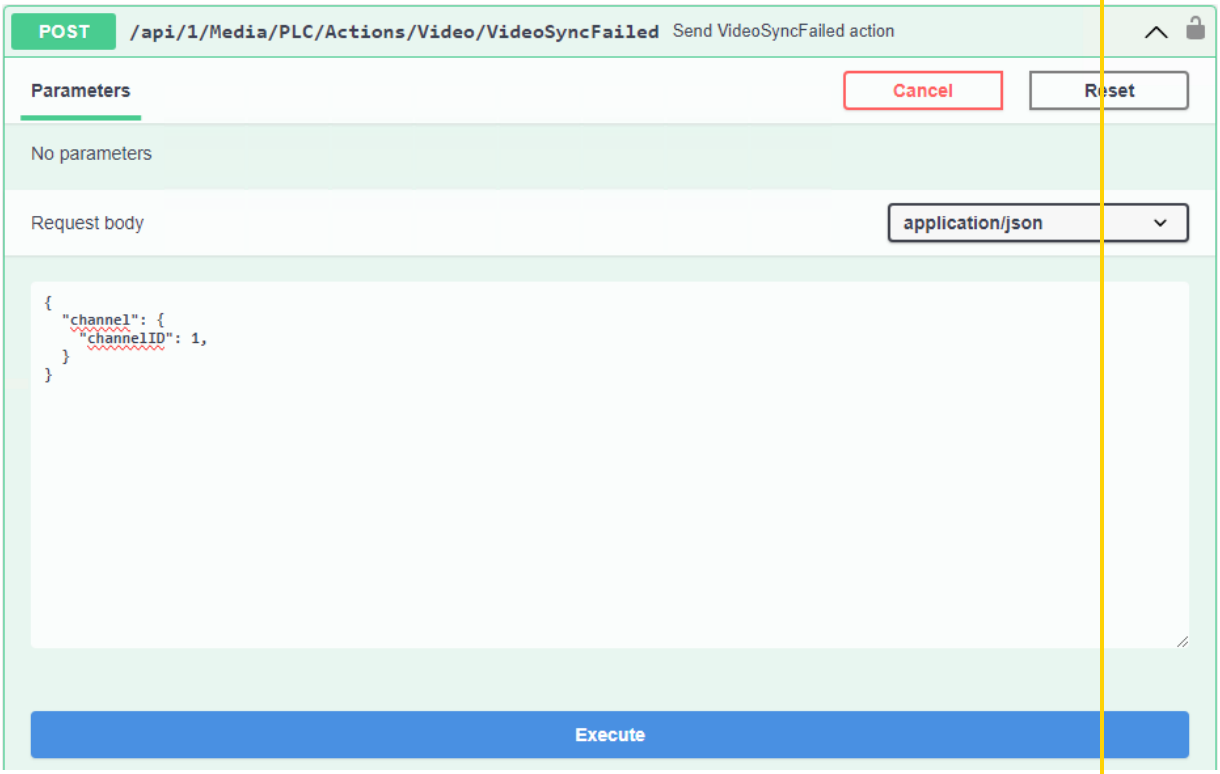
Value	Description
"channelID"	The local number of the media channel.

Value	Description
"channelName"	The name of the channel. This must match exactly to ensure assignment.
"globalNo"	The global number of the media channel.

Enter only one value for the parameter. If you specify more than one value, G-Core cannot find the assigned media channel and the action will not be triggered.

Example

In this example, the value `"channelID"` is specified. The other values must be removed as they must not be specified.



G-Core Actions Reference

In this PDF document you will find a complete definition of all G-Core actions and parameters: **G-Core Actions Reference.**

RTSP

Use RTSP Streaming

To use RTSP streaming, add the following section to the appsettings.json file (`C:\Program Files\Geutebrueck\GCore Web API\appsettings.json`).

You can use these parameters to configure RTSP streaming (see **RTSP Server**) and the streaming of recording gaps (see **Streaming**).

```
"Streaming": {
    "DBPlaybackMaximumGapMs": 5000,
    "DBPlaybackGapRecoverMs": 40
  },
  "RTSPServer": {
    "EnableRTSP": true,
    "ListenV6": "[::]",
    "ListenV4": "0.0.0.0",
    "ListenPort": 554
    //,"LogStreamFilePath": "c:\\temp"
  },
```

RTSP Server

To enable RTSP streaming, the EnableRTSP parameter must be set to **true**.

Encryption the RTSP Stream

You can encrypt the RTSP stream with a VPN connection between the client and the Web API.

If you are only using the RTSP stream, an SSL tunnel is also sufficient.

Authentication

The client sends the user name and password to the G-Core server from which the client expects to receive the stream. These credentials are then used to authenticate the user for the stream. If the authentication fails, an RTSP 401 response is sent.

Only basic authentication is supported.

> ⚠ **IMPORTANT:** This type of user authentication is not secure and is not recommended.

Recorded and Live Channel

The Web API distinguishes between the playback of recorded and live channels.

Recorded Channel:

When a specified time frame is called, the Web API checks whether the requested time frame exists and plays it at the normal 1.0x speed.

> Example
>
> `rtsp://localhost?MediaChannelIdentifier=1&Start=2022-01-31T13:05:04.447&End=2022-01-31T14:05:04.447`
>
> `Start=` Start time in the format `year-month-day T hours : minutes : seconds`.
>
> `End=` End time in the format `year-month-day T hours : minutes : seconds`.

Live Channel:

If neither a start time nor an end time is requested, the Web API plays the requested media channel as fast as possible and equivalent to the live view.

> Example
>
> `rtsp://localhost?MediaChannelIdentifier=1`

Streaming

If there are gaps in the database, these are skipped during playback and the next available image is displayed. You can configure the playback of recording gaps in the `Streaming` section.

Parameter	Description
DBPlaybackMaximumGapMs	Defines in milliseconds how large a gap is that

Parameter	Description
	is not considered a gap.
DBPlaybackGapRecoverMs	Defines in milliseconds after which the next available image is played.

Performance

To test the limitations of the G-Core Web SDK and Web API as well as some functionalities, some performance measurements were perfomed.

Specifications

The following specifications we used for the performance measurement:

- **Operating system:** Windows 10 x64
- **CPU:** Intel i7-7700
- **RAM:** 8 GB

Test Results

Scenario: Full HD 12,5 fps / outdoor quite

To make the test results comparable, the duration of each measurement was ~10 mins:

Viewer Count	CPU Usage	Memory Consumption	Test Result
4	~20%	~45%	Passed(stable)
8	~20%	~60%	Passed(stable)
10	~20%	~80%	Passed(stable)
14	~40%	~70%	Passed(stable)
18	~50%	~80%	Failed (unstable)

Framerate
We observed that some RTSP clients have problems with very low frame rates. We therefore recommend using a frame rate of at least 5 FPS.

Service Log Files

The Web API logs the most important messages to the Windows event log.

A more detailed log file can be found here: `%PROGRAMDATA%\Geutebrueck-\GCoreWebApi\GCoreWebApiLog.log`.

All outputs of the service are logged in this file.

Technical alterations reserved.

GEUTEBRÜCK GmbH
Im Nassen 7-9 | D-53578 Windhagen
Tel. +49 (0)2645 137-0 | Fax-999
info@geutebrueck.com

www.geutebrueck.com