



COMPOSER

How to setup connectors and cue points
using Vindral CDN and Web SDK.

Version 1.1

2023-02-07



Table of content

1 Introduction	2
1.1 Prerequisites	2
1.2 Topics	2
2 Setting up your RTMP-stream	3
2.1 Connect to the stream using the QOS Client	4
3 Creating the Vindral CDN Metadata Target	5
3.1 Sending your first metadata message	5
3.2 Use the QOS-Client to verify that the message has been broadcasted over the CDN	5
4 Using the /api/metadata/send API to send a message	6
5 Injecting metadata messages without using Composer	6
5.1 Postman example	6
5.2 Curl example	6
6 JSON or XML messages	7
7 Connectors and API Commands	8
7.1 Creating the connector	9
7.2 Adding ApiCommands	10
7.3 Dynamic parameters	11
8 Activating the Web API:s	11
8.1 Use HTTPS for encrypted communication and secure calls	11
9 Setting up API keys (apikey.json)	12
10 Web SDK - subscribe to channel and cue points	13

1 Introduction

Vindral Composer is a real-time video compositing software for event-driven video compositing, color correction, chroma key, visual effects, and live streaming.

Composer is part of the Vindral product family, which consists of Composer, Studio (encoder), and Streaming Engine (Playout/CDN). Both Composer and Studio are compatible with the Vindral Streaming Engine (CDN), and together they create a scalable end-to-end Ultra Low Latency streaming solution fed by real-time video compositions.

In this technical document, we will focus on Connectors and Cue points and describe how to setup, use, and consume metadata messages (cue) points using **Composer**, **Vindral CDN**, and the **Vindral Web SDK**.

For more generic documentation of Composer, see the User Guide and <https://docs.vindral.com>

1.1 Prerequisites

This guide does not describe the basics of Composer or how to install or operate Composer.

Before continuing, please make sure you have basic knowledge of the following topics and how to use Composer:

- Settings
- Scenes
- Inputs, operators, and targets
- RTMP-target
- Connectors

For basic information on the topics above, please read the Composer User Guide.

In order to follow the instructions in the guide, you also need:

- Composer installed on your workstation or server.
- Access to Vindral CDN
 - Access to ingest RTMP into a channel within the Vindral CDN
 - Knowledge about the **Private ID** of your stream. This ID is provided by RealSprint and is unique for your stream. The Private ID is used when ingesting the stream into the CDN.
 - Knowledge about the public **Channel ID** of your stream. This ID is provided by RealSprint and is unique to your stream. The CHANNEL ID is used when subscribing to the stream, or when connecting to the stream using the Vindral QOS Client.
 - Knowledge about the **Public Endpoint** of the Vindral CDN
- Access to the Vindral QOS-client. This is a web page provided by RealSprint.
- Access to the Vindral Web SDK. This is an npm package provided by RealSprint.

1.2 Topics

The topics covered in this document are:

- How to setup Connectors and API Commands
- How to activate the Web API:s
- How to setup API keys
- How to use the API:s to trigger a connector from an external application (http(s) calls)
- How to inject a metadata message ("cue point") into a channel within Vindral CDN
- How to use the QOS-client to verify that the metadata message has been sent
- How to retrieve the metadata message (and video stream) using the Vindral Web SDK

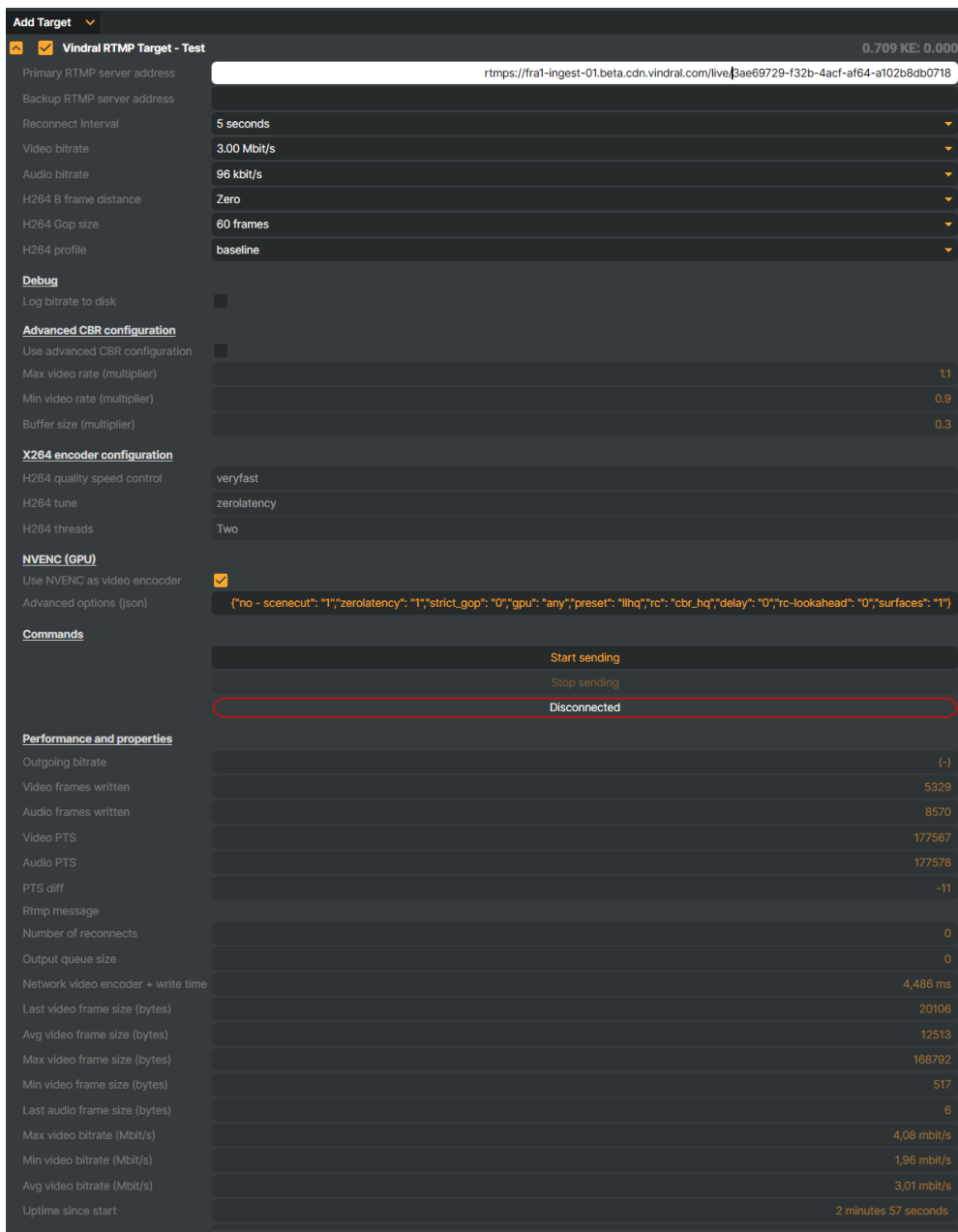
2 Setting up your RTMP-stream

In order to broadcast and receive metadata messages, you need an active channel within the Vindral CDN. To activate a channel, content needs to be ingested and that is typically done by ingesting an RTMP stream from Composer.

In order to **ingest a stream**, you need to know the Ingest URL and the **Private ID** of your channel. A typical ingest URL followed by the channel PRIVATE ID will have the following syntax (example):

<rtmps://fra1-ingest-01.cdn.vindral.com/live/your-private-id>

Add an **RTMP target** to your scene, and use the ingest URL and Private ID provided by RealSprint as the **Primary RTMP server address**:



Add Target ▼

Vindral RTMP Target - Test 0.709 KE: 0.000

Primary RTMP server address: `rtmps://fra1-ingest-01.beta.cdn.vindral.com/live/3ae69729-f32b-4acf-af64-a102b8db0718`

Backup RTMP server address: [Empty]

Reconnect Interval: 5 seconds

Video bitrate: 3.00 Mbit/s

Audio bitrate: 96 kbit/s

H264 B frame distance: Zero

H264 Gop size: 60 frames

H264 profile: baseline

Debug

Log bitrate to disk:

Advanced CBR configuration

Use advanced CBR configuration:

Max video rate (multiplier): 1.1

Min video rate (multiplier): 0.9

Buffer size (multiplier): 0.3

X264 encoder configuration

H264 quality speed control: veryfast

H264 tune: zerolatency

H264 threads: Two

NVENC (GPU)

Use NVENC as video encoder:

Advanced options (json): `("no - scenecut"; "1";zerolatency"; "1";strict_gop"; "0";gpu"; "any";preset"; "lhq";rc"; "cbr_hq";delay"; "0";rc-lookahead"; "0";surfaces"; "1")`

Commands

Start sending

Stop sending

Disconnected

Performance and properties

Outgoing bitrate	(-)
Video frames written	5329
Audio frames written	8570
Video PTS	177567
Audio PTS	177578
PTS diff	-11
Rtmp message	
Number of reconnects	0
Output queue size	0
Network video encoder + write time	4,486 ms
Last video frame size (bytes)	20106
Avg video frame size (bytes)	12513
Max video frame size (bytes)	168792
Min video frame size (bytes)	517
Last audio frame size (bytes)	6
Max video bitrate (Mbit/s)	4,08 mbit/s
Min video bitrate (Mbit/s)	1,96 mbit/s
Avg video bitrate (Mbit/s)	3,01 mbit/s
Uptime since start	2 minutes 57 seconds

Start the stream by pressing the Start sending button.

2.1 Connect to the stream using the QOS Client

Use the **Vindral QOS Client** to connect to the stream and to verify that the channel is up and running.

The URL of the QOS-client, the **Public Endpoint**, and the public **Channel ID** of your channel, have been provided by Realsprint.

Below is a screenshot of the QOS-client:

The screenshot displays the Vindral QOS Client interface. On the left, there are sections for 'CONNECT' (Public Endpoint: `https://fra1-lb-01.test.cdn.vindral.com/`), 'CHANNEL' (Channel ID: `realsprint_niclas_ci_976620f6-ffbe-4c92-f`), 'BUFFER' (slider from 1.0s to 4.0s), and 'FEATURES' (including 'Enable ABR Switching'). The main area shows a video player with a 'Switched to 3.1 mbit/s' notification and a progress bar. Below the player are several data panels:

- Metadata:**

Video		Audio	
Codec	h264	Codec	opus
Bit Rate	3 mbit/s	Bit Rate	64 kbit/s
Frame Rate	50p	Sample Rate	48 kHz
Resolution	1280x720	Codec String	N/A
Codec String	avc1.42c020	Language	No language set
- Bit Rate:** A line graph showing VIDEO at 3.1 mbit/s and AUDIO at 65.1 kbit/s.
- Playback:**

Playback State	playing
Last Buffer Event	filled
Buffer Drift	81.39ms
Buffer Drift Adjustments	2
Buffer Fullness	101%
Time Spent Buffering	00h 00m 10.931s
Time Spent Playing	00h 02m 26.514s
Time To First Frame	2211.00ms
- Connection:**

Edge U...	wss://fra1-edge-01.test.cdn.vindra...
Connection State	connected
RTT	32.4 ms
Estimated Bandwidth	14.6 mbit/s
Bit Rate	3.2 mbit/s
Connections Count	1
- Time:**

Uptime	00h 02m 38.545s
Server Edge Time	00h 00m 41.689s
Local Wallcloc...	Sat, 21 Aug 2021 18:3...
Server Wallclo...	Sat, 21 Aug 2021 18:1...

At the bottom left, there are 'Save settings' and 'Unload' buttons.

In order to be able to see incoming metadata messages in the QOS-client, make sure the **Show Metadata Events** feature is enabled. If not, press unload, enable Show Metadata Events, Save settings, and press Load.

3 Creating the Vindral CDN Metadata Target

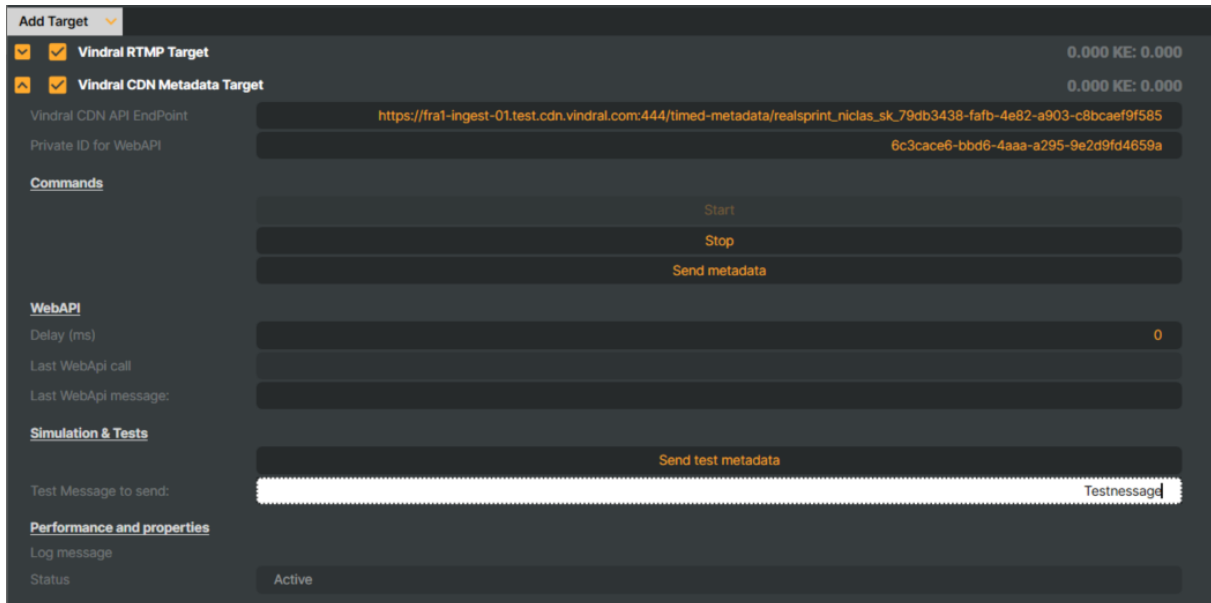
To add support for metadata messages, you need to add a Vindral CDN Metadata Target to your scene.

Add the Vindral CDN Metadata Operator to your scene, and use the **API URL for Timed Metadata** (provided by RealSprint) as the **Vindral CDN API Endpoint**.

The URL for Timed Metadata typically looks something like this (example):
<https://fra1-ingest-01.cdn.vindral.com:444/timed-metadata/your-private-id>



Note that this URL is not the same as the ingest URL, but the private id is the same.



3.1 Sending your first metadata message

To test the component, enter a test message into the **Test Message to send** field, and press the **Send test message button**. If the message is sent successfully the **Log Message** field should display the message Vindral CDN Metadata Target **StatusCode: Created**.

3.2 Use the QOS-Client to verify that the message has been broadcasted over the CDN

The message should arrive in the QOS client approximately 1.5 seconds after (assuming you are running the QOS-client with a 1500 ms buffer). The message will be shown in the upper right corner of the QOS-client:



4 Using the /api/metadata/send API to send a message

There are two ways of send a message using the Web API:s of Composer. The more advanced method is by using Connectors (described later in this guide), and using the api/metadata/send API.

```
/api/metadata/send?targetid=[target_guid]&message=[message]
```

Parameters:

- targetid: **private id** of the Vindral CDN Metadata target
- message: message to send

The API method should be called using a GET method.

If your system has been configured to use API keys, you need to provide a valid key when calling the API. API keys are described in a separate chapter later in this guide.

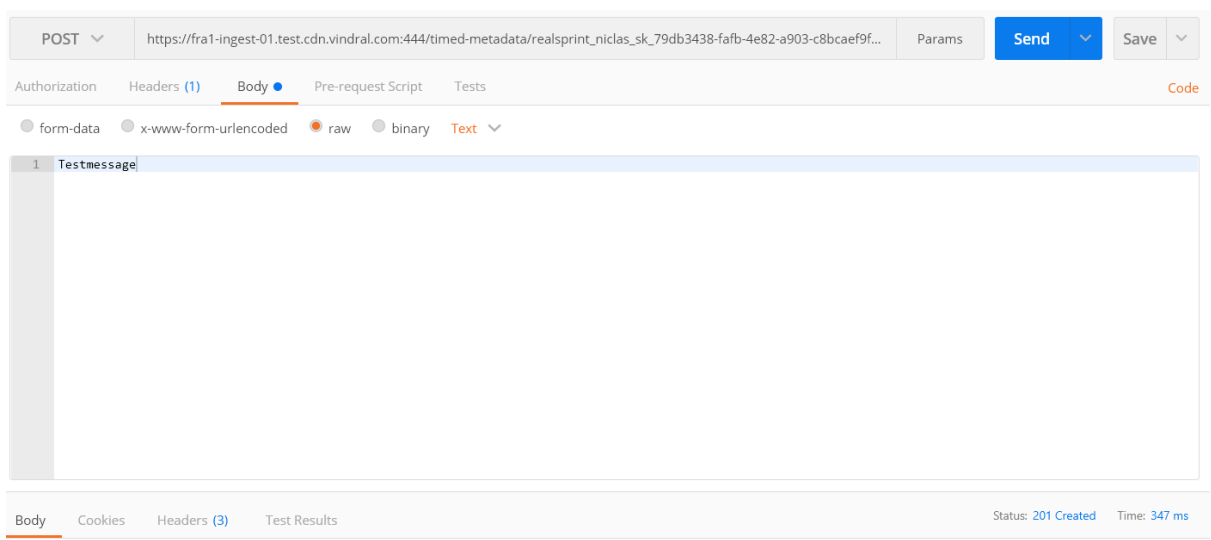
5 Injecting metadata messages without using Composer

Besides using Composer for ingesting metadata messages into Vindral CDN, you can use the API URL for Timed Metadata to inject a message.

Use **HTTP POST** and put the message into the **body** of the HTTP call. There are many ways of testing this, but you won't be able to do a POST in a web browser (without plugins). In this example, we will use the **Postman** application.

5.1 Postman example

Set the method to **POST**, and use the API URL for timed metadata and your **Private ID** as the URL. Add your message to the **Body** and press the **Send button**.



If the command is executed successfully, the **return status code** should be **201** (Created). In Postman, the return status code is shown in the lower right corner of the application.

5.2 Curl example

```
curl -XPOST -d "Testmessage" https://fra1-ingest-01.cdn.vindral.com:444/timed-metadata/your-private-id
```

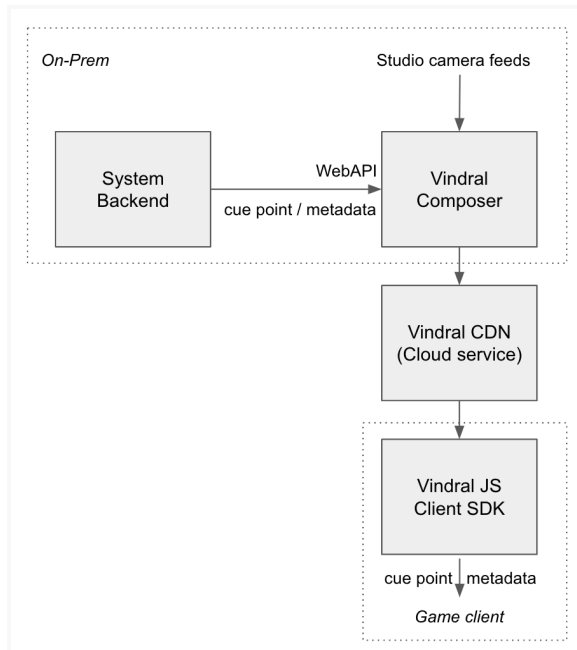
6 JSON or XML messages

Metadata messages often contain more than just a single text message. In many cases, JSON or XML messages are used in order to send multiple variables. Both JSON and XML messages are supported. Below is an example of a JSON message containing two variables:

```
{"Parameter 1": "Value of parameter 1", "Parameter 2": "Value of parameter 2"}
```


7 Connectors and API Commands

After setting up one or more scenes with relevant input sources, operators (effects), and outputs, you might want to control Composer from an external application. One such example might be a backend system used in your application:



Composer exposes HTTP(S) WEB APIs;s which can be used for external application control - either in-house/on-prem or as a service on the public Internet.

The API:s are divided into several parts and in this document we will focus on API:s for triggering Connectors and API Commands.

Connectors can be used in many different ways, but in this document we will focus on how to setup, use, and consume “cue points”, or metadata messages.

These metadata messages might also be referred to as out-of-band metadata.

Metadata messages are broadcasted separately from the audio and video stream and are only supported by the Vindral CDN. If you are using a different CDN for playback of your streams, Vindral metadata messages will not work.

Cue points or other types of metadata messages can be important for the synchronization of application logic/graphics and stream content, or to send commands or data to all clients subscribing to the video stream.

A Connector can trigger several *Actions* (one or more), and Composer can serve any number of unique Connectors.

Connectors are defined by a unique name, and are very easy to trigger using an http(s) GET request:

<https://composer.internal/api/connector/trigger?value=NoMoreBets>

In the example above, the machine name called **composer.internal** running Composer exposes a Connector called **NoMoreBets**.

HTTPS is typically not available by default, unless your server/workstation has a valid certificate installed, or if you are running private certificates in your server environment.

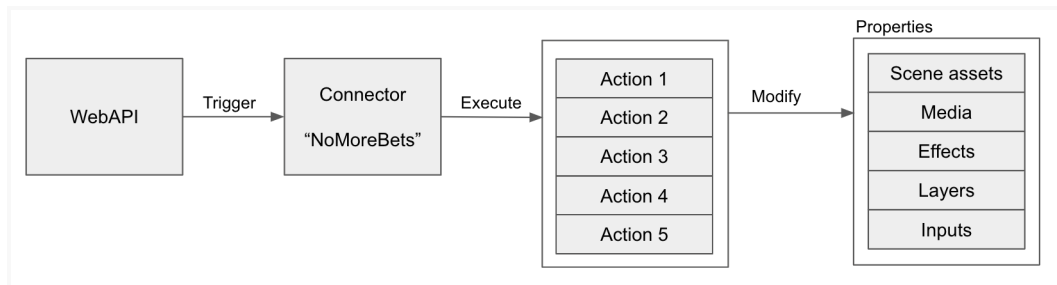
To call the API:s without HTTPS, use the IP address of your server/workstation, or localhost as hostname. HTTP calls require a port number (as specified in Composer Settings). The default port is 44333.

HTTP examples:

[http://localhost:44333/api/connector/trigger?value= \"NoMoreBets\"](http://localhost:44333/api/connector/trigger?value= \)

[http://192.168.1.33:44333/api/connector/trigger?value= \"NoMoreBets\"](http://192.168.1.33:44333/api/connector/trigger?value= \)

Triggering this Connector, using a valid API key, will trigger the actions defined for the NoMoreBets connector:



Actions can perform operations on all layers in Composer and operate on any effect, graphics layer, camera layers, etc. It's common to combine several actions into a Connector. Here are a few examples of actions (high level)

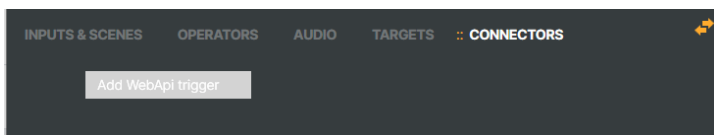
- Send cue points in streams - used for game/application synchronization, etc
- Change camera views
- Change scene
- Start/stop media clips
- Change volume
- Activate/deactivate effects
- Hide/show layers

In many scenarios, a combination of actions takes place in a single Connector. How the actions are set up depends on the type of game, type of video compositing, type of event, etc.

Connectors can be defined for any arbitrary event. Within iGaming, such events can include game states as NewRound, WinningNumber, PlaceYourBets, or similar.

7.1 Creating the connector

To create the connector for sending a cue point, add a new WebApi trigger by **right-clicking** in the Connectors tab:

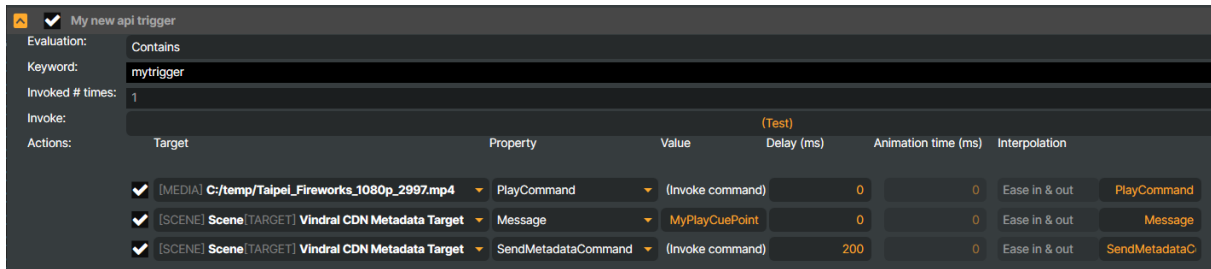


Give the Target a name of your choice. In this example we will name the Connector "My new api trigger".

7.2 Adding ApiCommands

After the connector has been created, you need to define a **keyword** that will be used when calling the APIs and triggering the Connector.

As per default, the evaluation method is set to **Contains** which means that any API call that contains a value that contains



In the example above the connector called "My new api trigger" is defined and listens to API-calls with this syntax:

[https://composer.internal/api/connector/trigger?value="mytrigger"](https://composer.internal/api/connector/trigger?value=)

(The FQDN composer.internal is just an example of a hostname, You can also use HTTP and local IP or localhost in your call)

When triggered three (3) actions will take place:

1. Start playback of the "Taipei_Fireworks_1080p_2997.mp4" video clip.
2. Set the metadata message to "MyPlayCuePoint" on the target "Vindral CDN Metadata Target".
3. After 200 ms, send the metadata command

A Connector may also utilize parameters that are useful for some actions. One example is cue points where Composer should send a cue point containing an eventid (or any other type of metadata). This eventid/metadata can be supplied in any Connector definition and is used by some actions. Below is an example call:

[https://composer.internal/api/connector/trigger?value="WinningNumber"&Param1="32"&apikey=494df910-9c09-4c7d-b240-37ba8b358de8](https://composer.internal/api/connector/trigger?value=)

Or without HTTPS and without an apikey:

[http://your-ip:44433/api/connector/trigger?value="WinningNumber"&Param1="32"](http://your-ip:44433/api/connector/trigger?value=)
[http://localhost:44433/api/connector/trigger?value="WinningNumber"&Param1="32"](http://localhost:44433/api/connector/trigger?value=)

In the examples above, the Param1 value of 32 can be used in an Action where a cue point is ingested into the outgoing video stream.

7.3 Dynamic parameters

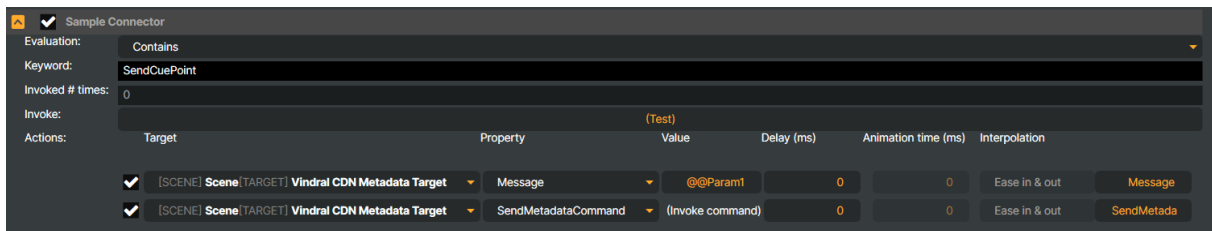
Dynamic parameters are a way of forwarding incoming values from an API call to a parameter in a Vindral input, operator, or target.

This is useful when the message itself contains non-static content, such as cue point identifiers, timers, or any kind of unique data that is non-static.

Example:

[https://composer.internal/api/connector/trigger?value="SendCuePoint"&Param1="32"](https://composer.internal/api/connector/trigger?value=)

To pass the **Param1** value into an API command, use the **@@** prefix before the parameter name. Below is an example of how the **@@Param1** value is used for forwarding Param1 as a message (cue point) into the Vindral CDN Metadata Target:



You can use multiple dynamic parameters, and you can use a parameter on multiple API commands.

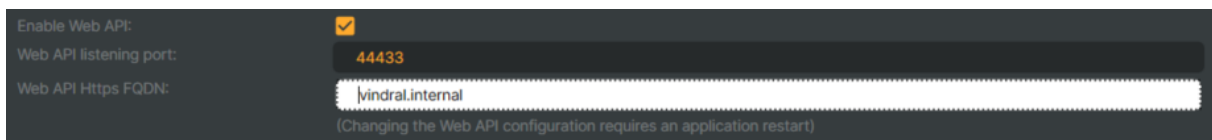
8 Activating the Web API:s

Before you can utilize the Web APIs of Composer, you need to activate them using the Settings window.

Make sure the APIs are **activated** and set a **PORT** of your choice.

Also, make sure the server **firewall** does not prohibit communication on the selected port.

To activate the optional **HTTPS** support, provide the hostname into the Web API Https FQDN field and make sure your server has a valid certificate.



8.1 Use HTTPS for encrypted communication and secure calls

We **strongly recommend** using HTTPS for all communication with Composer - even on private networks.

9 Setting up API keys (apikey.json)

API keys are used to protect the Web APIs in Composer from unauthorized access. By default Composer comes with no defined API keys which means that the APIs are accessible by anyone that can reach the server via the local network or via the internet. This is not a very secure setup, and we **strongly recommend** using API keys.

You can define any number of API keys, and depending on your setup, architecture, and security policies you might only need one API key, or, you can create multiple API keys and share the keys with different backend systems.

API-keys are defined in a JSON file, **apikey.json**, found in the **application root folder**.

Below is an example in which two API keys has been defined:

```
{
  "Keys": [
    "494df910-9c09-4c7d-b240-37ba8b358de8",
    "a1cc9a6f-8c40-49bc-8611-5b4131499fa9"
  ]
}
```

If API-keys are defined in `apikey.json`, any API-call must include a valid API-key in order for the API-call to be accepted. If no API keys are defined, no API key is required.

Example of an Api-call which includes an apikey query parameter:

https:

<https://composer.internal/api/connector/trigger?value=WinningNumber&Param1=32&apikey=494df910-9c09-4c7d-b240-37ba8b358de8>

http:

<https://composer.internal:44433/api/connector/trigger?value=WinningNumber&Param1=32&apikey=494df910-9c09-4c7d-b240-37ba8b358de8>

10 Web SDK - subscribe to channel and cue points

The Vindral Web SDK can be used to create a web page or application that connects to a channel, play the video stream, and subscribe to metadata messages.

Below is a very simple example that connects to a stream, and logs all incoming metadata messages in the web browser Console. The example will create an instance of the Core Player which is a bare-bone player without any visible controls.

This document does not cover how to install the Web SDK, or setting up build steps.

```
import {Vindral} from '@vindral/web-sdk'
const vindral = new Vindral({url:'https://public-end-point', channelId:
'your-public-id'})
vindral.attach(document.getElementById('root'))
vindral.connect()
vindral.on('metadata', (metadata)=>{
  console.log(metadata)
})
```

Replace **your-public-id** with the actual public channel id, and the **public-end-point** with **Public Endpoint** of the Vindral CDN.

For more information on Vindral Web SDK, see the online documentation.