Use the Wowza ClearCaster GraphQL API

Updated on 02/03/2023 11:26 am PST

The Wowza™ ClearCaster GraphQL API enables you to manage your Wowza ClearCaster™ appliance and broadcasts directly via the API. GraphQL APIs are structured so that the operations mirror the responses. This article describes how to create, configure, manage, publish, and end a broadcast with the Wowza ClearCaster GraphQL API. Some additional useful operations are also provided.

> **Note:** Wowza ClearCaster version 2.0.0.07 or later is required.

Video tutorial: Get started with the Wowza ClearCaster GraphQL API

Overview of the Wowza ClearCaster GraphQL API schema

If you're not familiar with the structure and terminology of GraphQL APIs, see GraphQL Schema and Types for more information.

The Wowza ClearCaster GraphQL API uses three root object types: query, mutation, and subscription. All example operations will use these root object type as operational keywords as well as the operation name. Query operations return an item or an array of items, while mutations create or modify objects and can return a field or an array of fields for the created or updated object. Note that mutation operations return object IDs, which are required as input for subsequent operations.

To make the API more intuitive to use, we've used the following naming patterns:

- Any operation that starts with **create** has a corresponding **delete** operation.
- Any operation that starts with **add** has a corresponding **remove** operation.
- Any operation that starts with **set** can add an item in addition to modifying a value.

For a complete list of operations and their arguments, see the documentation in the GraphiQL app or our Wowza ClearCaster GraphQL API reference documentation.

Known limitations

- Only the following video codecs and implementations are supported:
  - H.264
    - MainConcept software transcoding (default)
    - x264 software transcoding (x264)
    - Intel QuickSync hardware-accelerated transcoding (QuickSync)
  - H.265
    - Intel QuickSync hardware-accelerated transcoding (QuickSync)

Object hierarchy

To help you understand the schema on which the Wowza ClearCaster GraphQL API is built, we provide an object hierarchy graph as part of our reference documentation. The object hierarchy shows the different types of objects as well as how they relate to each other. To see this visualization of the API, see 'Object Hierarchy' in our Wowza ClearCaster GraphQL API reference documentation.

Use GraphiQL to access Wowza ClearCaster GraphQL API

We recommend using the GraphiQL app, an interactive, browser-based IDE for exploring and using GraphQL APIs. You can use GraphiQL the same way you might use Postman to test and learn about other types of APIs. For your convenience, we provide an instance of GraphiQL for you to use to explore the Wowza ClearCaster GraphQL API.

> **Note:** For more information about GraphiQL, see the GraphiQL GitHub repository.

1. Sign in to Wowza ClearCaster Manager at clearcaster.wowza.com.
2. Click **Manage** in the menu bar, click the **Integrations** tab, and then click **Generate a new ClearCaster API Key**.
3. Enter a **name** for your API key, and then click **Generate a new ClearCaster API Key**.

   > **Important!** You must save the API key, Secret Key, and GraphiQL URL provided. They are only displayed once, but you will use them every time you access GraphiQL.

4. Open a new browser tab, and enter your **GraphiQL URL**.
   The GraphiQL interface automatically opens using the account-specific API key and Secret Key integrated into the GraphiQL URL.

Create a broadcast

Next, create a broadcast using *yourNamespaceId* and replacing *broadcastName* with the name of the broadcast. Note that *inputs* takes an array of one or more source streams.

The following example creates the broadcast as well as defining the following objects associated with the broadcast: an array of at least one input, an array of at least one broadcast encoder, a stream target, and the video and audio encoding configurations. Note that each of these objects can be created and managed separately as well.

```
mutation createBroadcast {
  createBroadcast(namespaceId: "yourNamespaceId",
    input: {
      name: "broadcastName",
      inputs: {
        inputType: CAPTURE_HARDWARE,
        videoFrameWidthMax: 1920,
        videoFrameHeightMax: 1080,
        videoFrameRateMax: 30,
        broadcastInputEncoderIndex: 0
        },
      broadcastEncoders: [
        {
        encoderId: "yourEncoderId",
        streamTargetEncoderIndex: 0,
        broadcastInputEncoderIndex: 0
        }
      ],
      outputs: [
        {
          streamName: "1080p30",
          streamTargets:[
            {
              url: "rtmp://sourceURL",
              streamName: "yourStreamName",
              protocol: RTMP,
              streamTargetEncoderIndex: 0,
            },
      ],
          encodingConfiguration:
          {
            name: "1080p30",
            encodingConfigurationVideo:
            {
              codec: "H.264",
              implementation: "x264",
              frameSizeFitMode: "stretch",
              frameSizeWidth: 1920,
              frameSizeHeight: 1080,
              profile: "main",
              bitrate: 8000000,
              bitrateMin: 1000000,
              autoAdjustBitrate: true,
              keyFrameIntervalFollowSource: true,
              keyFrameInterval: 60,
              parameters: [
                {
                  name: "x264.preset",
                  value: "3",
                  type: "Long"
                }
```

```
            },
            {
              name: "x264.ref",
              value: "1",
              type: "Long"
            },
            {
              name: "x264.bframes",
              value: "1",
              type: "Long"
            }
          ]
        },
        encodingConfigurationAudio:
        {
          codec: "AAC",
          bitrate: 96000
        }
      }
    }
  ]
 }
) {
   id
   name
 }
}
```

This returns a broadcast ID (*yourBroadcastId*), which is used in other operations.

```
{
 "data": {
  "createBroadcast": {
   "id": "yourBroadcastId",
   "name": "broadcastName"
 }
}
```

Publishing the broadcast

A broadcast can be activated or deactivated. Activated broadcasts can have a status of IDLE (default), PREVIEW, LIVE, STOPPED.

Activate the broadcast

> **Note:** A broadcast must be activated to be published, however, if the broadcast you're activating already has a status of LIVE, it will go live immediately. Always make sure the broadcast status is IDLE before activating the broadcast. For more information, see Set the broadcast status.

To activate the broadcast, use the following operation with *yourBroadcastId*.

```
mutation activateBroadcastEncoders {
  activateBroadcastEncoders(broadcastId: "yourBroadcastId") {
    id
    broadcastEncoders {
      id
      encoder {
        id
        deviceId
        name
      }
    }
  }
}
```

This returns:

```
{
  "data": {
    "activateBroadcastEncoders": {
      "id": "yourBroadcastId",
      "broadcastEncoders": [
        {
          "id": "yourBroadcastEncoderId",
          "encoder": {
            "id": "yourEncoderId",
            "deviceId": "encoderUUID",
            "name": "encoder1"
          }
        }
      ]
    }
  }
}
```

Set the broadcast status

Use the *setBroadcastStatus* operation with *yourBroadcastId* to update the status of a broadcast. The status can be set to:

- *IDLE* - (Default) Although a status can be set when creating a broadcast, we recommend always resetting the broadcast status to IDLE before going live.
- *PREVIEW* - Use the PREVIEW status to test the broadcast before going live.
- *LIVE* - Use the LIVE status to publish the broadcast and deliver it to any configured stream target destinations.
- *STOPPED* - Use the STOPPED status to end a broadcast. If you're done with the broadcast, you'll usually want to deactivate it as well. See Deactivate the broadcast.

The following operation demonstrates how to set the broadcast status to IDLE. The same operation can be used to set the other statuses by changing 'status: IDLE' to one of the other defined status values.

```
mutation setBroadcastStatusIDLE {
  setBroadcastStatus(broadcastId: "yourBroadcastId", status: IDLE) {
    id
    status
    liveAt
    previewedAt
    stoppedAt
  }
}
```

This returns:

```
{
  "data": {
    "setBroadcastStatus": {
      "id": "yourBroadcastId",
      "status": "IDLE",
      "liveAt": null,
      "previewedAt": null,
      "stoppedAt": null
    }
  }
}
```

Deactivate the broadcast

Use the following operation, with *yourBroadcastId*, to deactivate a broadcast.

```
mutation deactivateBroadcastEncoders {
  deactivateBroadcastEncoders(broadcastId: "yourBroadcastId") {
    id
  }
}
```

When you've successfully deactivated your broadcast, you should see a response similar to the following:

```
{
  "data": {
    "deactivateBroadcastEncoders": {
      "id": "yourBroadcastId"
    }
  }
}
```

> **Note:** Deactivated broadcasts can be reactivated and used again. However, if you intend to use a broadcast again make sure to reset the broadcast's status to IDLE after deactivating it. This ensures it's in a good state when you use it again.

Useful queries

The following is a list of useful queries. For a complete list, see the documentation in the GraphiQL app.

- All namespaces

```
query allNamespaces {
  allNamespaces {
    id
    name
  }
}
```

- All encoders

```
query allEncoders {
  allEncoders {
    id
    name
    deviceId
  }
}
```

- All encoders by deviceId

```
query encoderByDeviceId {
  encoderByDeviceId(deviceId: "encoderUUID") {
    id
    name
    deviceId
  }
}
```

- All broadcasts

```
query allBroadcasts {
 allBroadcasts {
  id
  name
  inputs {
    inputType
    videoInput
    videoFrameWidthMax
    videoFrameHeightMax
    videoFrameRateMax
    videoAspectRatioMode
    videoAspectRatioWidth
    videoAspectRatioHeight
    videoAspectRatioRotation
    audioLevel
    overlayVendor
    overlayUrl
  }
  outputs {
   id
   encodingConfiguration {
    name
    encodingConfigurationVideo {
     codec
     bitrate
    }
    encodingConfigurationAudio {
     codec
     bitrate
    }
   }
   streamTargets {
    id
    url
    protocol
    streamTargetEncoderIndex
   }
   recordings {
    id
    format
   }
  }
  displays {
   id
  }
  broadcastEncoders {
   id
   streamTargetEncoderIndex
   broadcastInputEncoderIndex
   encoder {
    id
    deviceId
    name
   }
  }
 }

}
```

- A broadcast

```
query broadcasts {
  broadcast(id: "yourBroadcastId") {
    id
    name
    createdAt
    updatedAt
    previewedAt
    liveAt
    stoppedAt
    inputs {
      inputType
      videoInput
      videoFrameWidthMax
      videoFrameHeightMax
      videoFrameRateMax
      videoAspectRatioMode
      videoAspectRatioWidth
      videoAspectRatioHeight
      videoAspectRatioRotation
      audioLevel
      overlayVendor
      overlayUrl
    }
    broadcastEncoders {
      id
      streamTargetEncoderIndex
      broadcastInputEncoderIndex
      encoder {
        id
        deviceId
        name
      }
    }
    outputs {
      id
      encodingConfiguration {
        name
        encodingConfigurationVideo {
          codec
          bitrate
        }
        encodingConfigurationAudio {
          codec
          bitrate
        }
      }
      streamTargets {
        id
        url
        protocol
        streamTargetEncoderIndex
      }
      recordings {
        id
        format
      }
    }
  }
}
```