Wowza ClearCaster GraphQL API resource limitations

Learn about working with resource limitations in the Wowza ClearCaster™ GraphQL API for Wowza ClearCaster appliances.

Note: Wowza ClearCaster version 2.0.0.07 or later is required.

About Wowza ClearCaster GraphQL API resource limitations

Operations performed with the Wowza ClearCaster GraphQL API are subject to limitations that protect shared server resources. This article provides an overview of current methods for calculating and tracking API rate limits, however, limitations and the formula used to calculate them are subject to change without notice.

The Wowza ClearCaster GraphQL API uses rate limits set per namespace for operations that use API keys. The rate limit is an amount of operation cost that you can perform in one minute of UTC time.

Note: Operations originating from the Wowza ClearCaster Manager don't count towards the limit set for a namespace.

Rate limit calculations

Each operation has a calculated cost according to these factors:

- **Number of fields** - The more data you request, the higher the cost.
- **Type of operation** - Mutations have a higher cost than queries or subscriptions.
- **Depth** - The more levels deep the query is, the higher the cost. We recommend operations no more than four levels deep.
Here are some examples of operations and their calculated costs:

- **Calculated cost for the following activateBroadcastEncoders example query:** 21

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

- **Calculated cost for the following createBroadcast example mutation:** 12

```
mutation createBroadcast {
  createBroadcast(nsamespaceId: "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:
        }
      ]
    }
  }
}
```
Calculated cost for the following setBroadcastStatus example query: 15

```json
{
  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
  }
}
```

- Calculated cost for the following setBroadcastStatus example query: 15
mutation setBroadcastStatusIDLE {
  setBroadcastStatus(broadcastId: "yourBroadcastId", status: IDLE) {
    id
    status
    liveAt
    previewedAt
    stoppedAt
  }
}

- Calculated cost for the following allNamespaces example query: **10**

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

- Calculated cost for the following simple allBroadcasts example query: **10**

```graphql
query allBroadcasts {
  allBroadcasts {
    id
    name
  }
}
```

- Calculated cost for the following complex allBroadcasts example query: **71**

```graphql
query allBroadcasts {
  allBroadcasts {
    id
    name
    inputs {
      inputType
      videoInput
      videoFrameWidthMax
      videoFrameHeightMax
      videoFrameRateMax
      videoAspectRatioMode
      videoAspectRatioWidth
      videoAspectRatioHeight
      videoAspectRatioRotation
      audioLevel
      overlayVendor
      overlayUrl
    }
    outputs {
      id
      encodingConfiguration {
        name
        encodingConfigurationVideo {
          codec
        }
      }
    }
  }
}
```
Check rate limit status

You can check the status of rate limit cost consumption for the operations you perform by inspecting the HTTP response headers.

<table>
<thead>
<tr>
<th>Header name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-rate-limit-cost-consumed</td>
<td>The current amount of cost consumed from the limit</td>
</tr>
<tr>
<td>x-rate-limit-cost-limit</td>
<td>The current cost limit per minute set for the</td>
</tr>
</tbody>
</table>
If you perform an operation in GraphQL, you can use the Chrome DevTools Network panel to view the response headers.

<table>
<thead>
<tr>
<th>Header name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-rate-limit-cost-request</td>
<td>The cost of the current request</td>
</tr>
<tr>
<td>x-rate-limit-cost-reset</td>
<td>The amount of time, in milliseconds, until the cost consumed resets</td>
</tr>
</tbody>
</table>

If you perform an operation in GraphQL, you can use the Chrome DevTools Network panel to view the response headers.

Exceeded rate limits

If you perform an operation that exceeds the one-minute rate limit set for your namespace, your operation will return a 429 “Rate Limit Reached” error.