## Table of Contents

What is NICE DCV Web Client SDK? .................................................................................................................. 1
Prerequisites .................................................................................................................................................. 1
Supported features ...................................................................................................................................... 1
Browser support ........................................................................................................................................ 2
Versioning convention ............................................................................................................................... 2
Getting started ............................................................................................................................................. 3
  Connect to a NICE DCV server and get the first frame ............................................................................... 4
    Step 1: Prepare your HTML page .............................................................................................................. 4
    Step 2: Authenticate, connect, and get the first frame ............................................................................. 4
    Bonus: Automatically create an HTML login form ..................................................................................... 6
Work with NICE DCV features ....................................................................................................................... 7
  Understanding the featuresUpdate callback function ................................................................................ 7
  Handling feature updates ............................................................................................................................ 7
Use NICE DCV Web UI SDK ........................................................................................................................... 8
  Prerequisites ............................................................................................................................................... 8
  Step 1: Prepare your HTML page .............................................................................................................. 9
  Step 2: Authenticate, connect and render the DCVViewer React component ........................................ 9
SDK reference .................................................................................................................................................. 13
  DCV module .............................................................................................................................................. 13
    Methods .................................................................................................................................................. 13
    Members ................................................................................................................................................ 15
    Type and callback definitions .................................................................................................................. 18
  Connection Class ....................................................................................................................................... 43
    Methods .................................................................................................................................................. 13
  Authentication Class .................................................................................................................................. 60
    Methods .................................................................................................................................................. 13
  Resource Class .......................................................................................................................................... 61
    Methods .................................................................................................................................................. 13
NICE DCV Web UI SDK .................................................................................................................................. 61
  Components .............................................................................................................................................. 62
Release Notes and Document History .......................................................................................................... 66
  Release Notes ......................................................................................................................................... 66
    1.2.1 — July 21, 2022 ............................................................................................................................. 66
    1.2.0 — June 29, 2022 ............................................................................................................................. 66
    1.1.3 — May 23, 2022 ............................................................................................................................. 67
    1.1.2 — May 19, 2022 ............................................................................................................................. 67
    1.1.1 — March 23, 2022 .......................................................................................................................... 67
    1.1.0 — February 23, 2022 ...................................................................................................................... 67
    1.0.4 — December 20, 2021 .................................................................................................................... 67
    1.0.3 — September 01, 2021 ................................................................................................................... 68
    1.0.2 — July 30, 2021 .............................................................................................................................. 69
    1.0.1 — May 31, 2021 .............................................................................................................................. 69
    1.0.0 — March 24, 2021 .......................................................................................................................... 69
Document History ............................................................................................................................................. 69
What is the NICE DCV Web Client SDK?

NICE DCV is a high-performance remote display protocol. It lets you securely deliver remote desktops and application streaming from any cloud or data center to any device, over varying network conditions. By using NICE DCV with Amazon EC2, you can run graphics-intensive applications remotely on Amazon EC2 instances. You can then stream the results to more modest client machines, which eliminates the need for expensive dedicated workstations.

The NICE DCV Web Client SDK is a JavaScript library that you can use to develop your own NICE DCV web browser client applications. Your end users can use these applications to connect to and interact with a running NICE DCV session.

Using the NICE DCV Web Client SDK as a building block, you can build customized web applications that provide users with instant access to their desktop or applications from anywhere, with a responsive and fluid performance that is almost indistinguishable from a natively installed application.

This guide explains how to use the NICE DCV Web Client SDK to build your custom web browser client applications to interact with NICE DCV sessions within your workflows.

Topics
- Prerequisites (p. 1)
- Supported features (p. 1)
- Browser support (p. 2)
- Versioning convention (p. 2)

Prerequisites

Before you start working with the NICE DCV Web Client SDK, ensure that you're familiar with NICE DCV and NICE DCV sessions. For more information, see the NICE DCV Administrator Guide.

The NICE DCV Web Client SDK supports NICE DCV server version 2020 and later.

Supported features

You can build custom web browser client applications that support the following NICE DCV features:
- Connect to Windows NICE DCV servers
- Connect to Linux NICE DCV servers
- Manage streaming modes
- Transfer files
- Print from sessions
- Copy and paste
- Stereo 2.0 audio playback
- Stereo 2.0 audio recording (on Windows servers)
Browser support

The NICE DCV Web Client SDK supports JavaScript (ES6) and it can be used from JavaScript or TypeScript applications.

The NICE DCV Web Client SDK supports the following web browsers:

<table>
<thead>
<tr>
<th>Browser</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Chrome</td>
<td>Latest three major versions</td>
</tr>
<tr>
<td>Mozilla Firefox</td>
<td>Latest three major versions</td>
</tr>
<tr>
<td>Microsoft Edge</td>
<td>Latest three major versions</td>
</tr>
<tr>
<td>Apple Safari for macOS</td>
<td>Latest three major versions</td>
</tr>
</tbody>
</table>

Versioning convention

The NICE DCV Web Client SDK version is defined in the following format: `major.minor.patch`. The versioning convention generally adheres to the semantic versioning model. A change in the major version, such as from `1.x.x` to `2.x.x`, indicates that breaking changes that might require code changes and a planned deployment have been introduced. A change in the minor version, such as from `1.1.x` to `1.2.x`, is backwards compatible, but might include deprecated elements.
Getting started with the NICE DCV Web Client SDK

The NICE DCV Web Client SDK comprises of a main `dcv.js` file and some auxiliary components. All the files are distributed inside a compressed archive that can be downloaded from the NICE website.

To get started with the NICE DCV Web Client SDK

1. The NICE DCV Web Client SDK archive is digitally signed with a secure GPG signature. To verify the archive's signature, you must import the NICE GPG key. To do so, open a terminal window and import the NICE GPG key.

   ```
   $ wget https://d1uj6qtbmh3dt5.cloudfront.net/NICE-GPG-KEY
   $ gpg --import NICE-GPG-KEY
   ```

2. Download the **NICE DCV Web Client SDK archive** and the **NICE DCV Web Client SDK archive signature** from the NICE website.

3. Verify the signature of the NICE DCV Web Client SDK archive using the signature.

   ```
   $ gpg --verify
   signature_filename.zip.sign
   archive_filename.zip
   ```

   For example:

   ```
   $ gpg --verify nice-dcv-web-client-sdk-1.2.1-358.zip.sign nice-dcv-web-client-sdk-1.2.1-358.zip
   ```

4. If the signature verifies successfully, extract the contents of the NICE DCV Web Client SDK archive and place the extracted directory on your web server. For example:

   ```
   $ unzip
   archive_filename.zip
   -d /path_to/server_directory
   ```

**Important**

- You must retain the folder structure when deploying the NICE DCV Web Client SDK on your web server.
- When using NICE DCV Web UI SDK, please beware that the `DCVViewer` React component expects the EULA.txt and third-party-licenses.txt files from this package to be present in the URL path for the embedded web server. The third-party-licenses.txt file should be
Connect to a NICE DCV server and get the first frame

The following tutorial shows you how to prepare your HTML page for your custom web client, how to authenticate and connect to a NICE DCV server, and how to receive the first frame of streamed content from the NICE DCV session.

Topics
- Step 1: Prepare your HTML page (p. 4)
- Step 2: Authenticate, connect, and get the first frame (p. 4)
- Bonus: Automatically create an HTML login form (p. 6)

Step 1: Prepare your HTML page

In your web page, you must load the needed JavaScript modules and you must add a `<div>` HTML element with a valid id where you want the NICE DCV Web Client SDK to draw the content stream from the remote NICE DCV server.

For example:

```html
<!DOCTYPE html>
<html lang="en" style="height: 100%;">
<head>
  <title>DCV first connection</title>
</head>
<body style="height: 100%;">
  <div id="root" style="height: 100%;"></div>
  <div id="dcv-display"></div>
  <script type="module" src="index.js"></script>
</body>
</html>
```

Step 2: Authenticate, connect, and get the first frame

This section shows how to complete the user authentication process, how to connect the NICE DCV server, and how to receive the first frame of content from the NICE DCV server.

First, from the `index.js` file import the NICE DCV Web Client SDK. It can be imported either as a Universal Module Definition (UMD) module, like so:

```javascript
import "./dcvjs/dcv.js"
```

Otherwise, starting from version 1.1.0, it can also be imported as a ECMAScript Module (ESM) from the corresponding package, like so:

```javascript
import dcv from "./dcvjs/dcv.js"
```
Define the variables to use to store the Authentication object, Connection object, and the NICE DCV server URL.

```javascript
let auth,
    connection,
    serverUrl;
```

On script load, log the NICE DCV Web Client SDK version, and on page load, call the `main` function.

```javascript
console.log("Using NICE DCV Web Client SDK version " + dcv.version.versionStr);
document.addEventListener('DOMContentLoaded', main);
```

The `main` function sets the log level and starts the authentication process.

```javascript
function main () {
    console.log("Setting log level to INFO");
    dcv.setLogLevel(dcv.LogLevel.INFO);

    serverUrl = "https://your-dcv-server-url:port/";

    console.log("Starting authentication with", serverUrl);
    auth = dcv.authenticate(
        serverUrl,
        {
            promptCredentials: onPromptCredentials,
            error: onError,
            success: onSuccess
        }
    );
}
```

The `promptCredentials`, `error`, and `success` functions are mandatory callback functions that must be defined in the authentication process.

If the NICE DCV server prompts for credentials, the `promptCredentials` callback function receives the requested credential challenge from the NICE DCV server. If the NICE DCV server is configured to use system authentication, then the credentials must be provided in the form of a user name and a password. The following code samples assume that the user name is `my_dcv_user` and that the password is `my_password`.

If authentication fails, the `error` callback function receives an error object from the NICE DCV server.

If the authentication succeeds, the `success` callback function receives an array of couples that includes the session id (`sessionId`) and authorization tokens (`authToken`) for each session that the `my_dcv_user` user is allowed to connect to on the NICE DCV server. The following code sample calls the connect function and connects to the first session returned in the array.

```javascript
function onPromptCredentials(auth, challenge) {
    // Let's check if in challenge we have a username and password request
    if (challengeHasField(challenge, "username") && challengeHasField(challenge, "password")) {
        auth.sendCredentials({username: "my_dcv_user", password: "my_password"})
    } else {
        // Challenge is requesting something else...
    }
}

function challengeHasField(challenge, field) {
    return challenge.requiredCredentials.some(credential => credential.name === field);
}
```
function onError(auth, error) {
    console.log("Error during the authentication: " + error.message);
}

// We connect to the first session returned
function onSuccess(auth, result) {
    let {sessionId, authToken} = {...result[0]};
    connect(sessionId, authToken);
}

Connect to the NICE DCV server. The `firstFrame` callback method is called when the first frame is received from the NICE DCV server.

```javascript
function connect (sessionId, authToken) {
    console.log(sessionId, authToken);
    dcv.connect({
        url: serverUrl,
        sessionId: sessionId,
        authToken: authToken,
        divId: "dcv-display",
        callbacks: {
            firstFrame: () => console.log("First frame received")
        }
    }).then(function (conn) {
        console.log("Connection established!");
        connection= conn;
    }).catch(function (error) {
        console.log("Connection failed with error " + error.message);
    });
}
```

**Bonus: Automatically create an HTML login form**

The `challenge` object is returned when the `promptCredentials` callback function is called. It includes a property named `requiredCredentials` that is an array of objects - one object per credential that is requested by the NICE DCV server. Each object includes the name and the type of the requested credential. You can use the `challenge` and `requiredCredentials` objects to automatically create an HTML login form.

The following code sample shows you how to do this.

```javascript
let form, fieldSet;

function submitCredentials (e) {  
    var credentials = {};
    fieldSet.childNodes.forEach(input => credentials[input.id] = input.value);
    auth.sendCredentials(credentials);
    e.preventDefault();
}

function createLoginForm () {
    var submitButton = document.createElement("button");
    submitButton.type = "submit";
    submitButton.textContent = "Login";
    ```
Work with NICE DCV features

The availability of NICE DCV features depends on the permissions configured for the NICE DCV session and the capabilities of the client's web browser.

The features that are available in a NICE DCV session are managed by the permissions that have been specified for the session. This means that even if a feature is supported by the NICE DCV Web Client SDK, access to that feature might be prevented based on the permissions defined by the session administrator. For more information, see Configuring NICE DCV Authorization in the NICE DCV Administrator Guide.

Understanding the featuresUpdate callback function

When the availability of a feature in a NICE DCV session changes, the NICE DCV Web Client SDK notifies you using the featuresUpdate callback function that you specify at the time of establishing the connection. For example:

```
featuresUpdate: function (connection, list) {
  ...
},
```

The callback function notifies you only of the features for which the availability has changed. The list parameter is an array of strings, and it includes only the names of the updated features. For example, if the availability of the audio input feature changes for the session, the parameter includes only "audio-in". If at a later point, the availability of the clipboard copy and paste features change for the session, the parameter includes only "clipboard-copy", "clipboard-paste".

Handling feature updates

The featuresUpdate callback function only notifies you that the availability of one or more features has changed. To know which features were updated, you must query the feature using the...
The `connection.queryFeature` method. This can be done at any time after the notification of change has been received. This method returns a Promise that resolves to the requested feature's updated status. The status value is always associated and it has a Boolean (true | false) property called enabled. Some features might have additional properties in the status value. If the feature's availability has not been updated, it's rejected.

The following example code shows how to do this.

```javascript
// Connection callback called
function featuresUpdate(_, list) {
  if (list.length > 0) {
    list.forEach((feat) => {
      connection.queryFeature(feat).then(status => console.log(feat, "is", status.enabled));
    });
  }
}
```

Use NICE DCV Web UI SDK

The following tutorial shows you how to authenticate against the NICE DCV server, connect to it and render the DCVView React component from the NICE DCV Web UI SDK.

Topics

- Prerequisites (p. 8)
- Step 1: Prepare your HTML page (p. 9)
- Step 2: Authenticate, connect and render the DCVView React component. (p. 9)

Prerequisites

You need to install React, ReactDOM, AWS UI Components React, AWS UI Global Styles and AWS UI Design Tokens.

```
$ npm i react react-dom @awsui/components-react @awsui/global-styles @awsui/design-tokens
```

You would also need to download NICE DCV Web Client SDK. See Getting started with the NICE DCV Web Client SDK (p. 3) to read the step-by-step guide on how to do that.

You must create an alias for importing the dcv module, since it is an external dependency for NICE DCV Web UI SDK. For instance, if you are using webpack to bundle your web app, you can use the resolve.alias option like so:

```javascript
const path = require("path");

module.exports = {
  //...
  resolve: { alias: {
    dcv: path.resolve("path", "to", "dcv.js"),
  },
},
};
```
If you are using rollup for bundling, you can install @rollup/plugin-alias, and use it like so:

```javascript
import alias from '@rollup/plugin-alias';
const path = require('path);
module.exports = {
  //...
  plugins: [
    alias({
      entries: [
        { find: 'dcv', replacement: path.resolve('path', 'to', 'dcv.js') },
      ]
    })
  ]
};
```

Step 1: Prepare your HTML page

In your web page, you must load the required JavaScript modules and you should have a `<div>` HTML element with a valid id where the entry component of your app will be rendered.

For example:

```html
<!DOCTYPE html>
<html lang="en" style="height: 100%;">
<head>
  <title>DCV first connection</title>
</head>
<body style="height: 100%;">
  <div id="root" style="height: 100%;"></div>
  <script type="module" src="index.js"></script>
</body>
</html>
```

Step 2: Authenticate, connect and render the DCVViewer React component.

This section shows how to complete the user authentication process, how to connect the NICE DCV server, and how to render the DCVViewer React component.

First, from the `index.js` file, import React, ReactDOM and your top level App component.

```javascript
import React from "react";
import ReactDOM from 'react-dom';
import App from './App';
```

Render the top level container node of your app.

```javascript
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById("root")
);
```

In the `App.js` file, import the NICE DCV Web Client SDK as a ESM module, the DCVViewer React component from the NICE DCV Web UI SDK, React and the AWS UI Global Styles package.
Step 2: Authenticate, connect and render the DCVViewer React component.

```javascript
import React from "react";
import dcv from "dcv";
import '@awsui/global-styles/index.css';
import {DCVViewer} from "./dcv-ui/dcv-ui.js";

const LOG_LEVEL = dcv.LogLevel.INFO;
const SERVER_URL = "https://your-dcv-server-url:port/";
const BASE_URL = "/static/js/dcvjs";

let auth;

function App() {
  const [authenticated, setAuthenticated] = React.useState(false);
  const [sessionId, setSessionId] = React.useState('');
  const [authToken, setAuthToken] = React.useState('');
  const [credentials, setCredentials] = React.useState({});

  const onSuccess = (__, result) => {
    var { sessionId, authToken } = { ...result[0] };
    console.log("Authentication successful.");
    setSessionId(sessionId);
    setAuthToken(authToken);
    setAuthenticated(true);
    setCredentials({});
  }

  const onPromptCredentials = (__, credentialsChallenge) => {
    let requestedCredentials = {};
    credentialsChallenge.requiredCredentials.forEach(challenge =>
      requestedCredentials[challenge.name] = "";
    )
    setCredentials(requestedCredentials);
  }

  const authenticate = () => {
    dcv.setLogLevel(LOG_LEVEL);
    auth = dcv.authenticate(
      SERVER_URL,
      {
        promptCredentials: onPromptCredentials,
        error: onError,
        success: onSuccess
      }
    );
  }

  const updateCredentials = (e) => {
    const { name, value } = e.target;
    setCredentials({
      ...credentials,
      [name]: value
    });
  }

  const submitCredentials = (e) => {
    auth.sendCredentials(credentials);
    e.preventDefault();
  }
```

Following is an example showing how to authenticate against the NICE DCV Server and render the DCVViewer React component from NICE DCV Web UI SDK, provided the authentication was successful.
React.useEffect(() => {
  if (!authenticated) {
    authenticate();
  }
}, [authenticated]);

const handleDisconnect = (reason) => {
  console.log("Disconnected: " + reason.message + " (code: " + reason.code + ")");
  auth.retry();
  setAuthenticated(false);
}

return {
  authenticated ?
  <DCVViewer
    dcv={{
      sessionId: sessionId,
      authToken: authToken,
      serverUrl: SERVER_URL,
      baseUrl: BASE_URL,
      onDisconnect: handleDisconnect,
      logLevel: LOG_LEVEL
    }}
    uiConfig={{
      toolbar: {
        visible: true,
        fullscreenButton: true,
        multimonitorButton: true,
      },
    }}
  />
  :
  <div
    style={{
      height: window.innerHeight,
      backgroundColor: "#373737",
      display: 'flex',
      alignItems: 'center',
      justifyContent: 'center',
    }}
  >
    <form
      <fieldset>
      {Object.keys(credits).map((cred) => {
        <input
          key={cred}
          name={cred}
          placeholder={cred}
          type={cred === "password" ? "password" : "text"}
          onChange={updateCredentials}
          value={credits[cred]}
        />
      })}
    </fieldset>
    <button
      type="submit"
      onClick={submitCredentials}
    >
      Login
    </button>
  </form>
  </div>
};
}
const onError = (_, error) => {
  console.log("Error during the authentication: " + error.message);
}

export default App;

The promptCredentials, error, and success functions are mandatory callback functions that must be defined in the authentication process.

If the NICE DCV server prompts for credentials, the promptCredentials callback function receives the requested credential challenge from the NICE DCV server. If the NICE DCV server is configured to use system authentication, then the credentials must be provided in the form of a user name and a password.

If authentication fails, the error callback function receives an error object from the NICE DCV server.

If the authentication succeeds, the success callback function receives an array of couples that includes the session id (sessionId) and authorization tokens (authToken) for each session that the user is allowed to connect to on the NICE DCV server. The code sample above updates the React state to render the DCVViewer component on successful authentication.

To know more about the properties accepted by this component, see the NICE DCV Web UI SDK reference.
SDK reference

This section provides descriptions, syntax, and usage examples for the NICE DCV Web Client SDK.

Topics
- DCV module (p. 13)
- Connection Class (p. 43)
- Authentication Class (p. 60)
- Resource Class (p. 61)
- NICE DCV Web UI SDK (p. 61)

DCV module

A module that implements the client side of the DCV protocol.

Exposes
- Methods (p. 13)
- Members (p. 15)
- Type and callback definitions (p. 18)

Methods

List
- authenticate(url, callbacks) → {Authentication (p. 60)}
- connect(config) → {Promise.<Connection>|Promise.<{code: ConnectionErrorCode, message: string}>} (p. 14)
- setLogHandler(handler) → {void} (p. 14)
- setLogLevel(level) → {void} (p. 15)

authenticate(url, callbacks) → {Authentication (p. 60)}

Starts the authentication process for the specified NICE DCV server endpoint.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>string</td>
<td>The host name and port of the running NICE DCV server in the following format: https://dcv_host_address:port. For example: <a href="https://my-dcv-server:8443">https://my-dcv-server:8443</a>.</td>
</tr>
</tbody>
</table>
## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>callbacks</td>
<td>authenticationCallbacks (p. 19)</td>
<td>The callbacks that are available to be called during the authentication process.</td>
</tr>
</tbody>
</table>

**Returns:**

- The Authentication object.

**Type**

Authentication (p. 60)

```javascript
connect(config) → {Promise.<Connection (p. 43)>| Promise.<{code: ConnectionErrorCode (p. 27), message: string}>}
```

Connects to the specified NICE DCV server endpoint. If connection succeeds, it returns a Connection object. If connection fails, it returns an error object.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>config</td>
<td>ConnectionConfig (p. 26)</td>
<td>The ConnectionConfig object.</td>
</tr>
</tbody>
</table>

**Returns:**

- A Connection object, or an error object.

**Type**

Promise.<Connection (p. 43)> | Promise.<{code: ConnectionErrorCode (p. 27), message: string}>

```javascript
setLogHandler(handler) → {void}
```

Sets a custom log handler function. When overriding the default log handler, the original log entry position will be lost when debugging with the browser console.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler</td>
<td>function</td>
<td>The custom log handler function. The handler function contains level (number), levelName (string), domain (string), and message (string).</td>
</tr>
</tbody>
</table>
**Returns:**

Type

void

**setLogLevel(level) → {void}**

Sets the log level. This is required only if the default log handler is used.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>LogLevel (p. 37)</td>
<td>The log level to use.</td>
</tr>
</tbody>
</table>

**Returns:**

Type

void

---

**Members**

**List**

- (constant) AudioError :AudioErrorCode (p. 15)
- (constant) AuthenticationError :AuthenticationErrorCode (p. 16)
- (constant) ChannelError :ChannelErrorCode (p. 16)
- (constant) ClosingReasonError :ClosingReasonErrorCode (p. 16)
- (constant) ConnectionState :ConnectionErrorCode (p. 16)
- (constant) CustomChannelError :CustomChannelErrorCode (p. 16)
- (constant) DisplayConfigError :DisplayConfigErrorCode (p. 16)
- (constant) FileStorageError :FileStorageErrorCode (p. 17)
- (constant) LogLevel :LogLevel (p. 17)
- (constant) MultiMonitorError :MultiMonitorErrorCode (p. 17)
- (constant) ResolutionError :ResolutionErrorCode (p. 17)
- (constant) version (p. 17)
- (constant) ScreenshotError :ScreenshotErrorCode (p. 18)
- (constant) WebcamError :WebcamErrorCode (p. 18)

**constant) AudioError :AudioErrorCode (p. 19)**

The AudioError codes enum.

**Type:**

- AudioErrorCode (p. 19)
(constant) AuthenticationError :AuthenticationErrorCode (p. 20)

The AuthenticationError codes enum.

Type:
- AuthenticationErrorCode (p. 20)

(constant) ChannelError :ChannelErrorCode (p. 22)

The ChannelError codes enum.

Type:
- ChannelErrorCode (p. 22)

(constant) ClosingReasonError :ClosingReasonErrorCode (p. 24)

The ClosingReasonError codes enum.

Type:
- ClosingReasonErrorCode (p. 24)

(constant) ConnectionError :ConnectionErrorCode (p. 27)

The ConnectionError codes enum.

Type:
- ConnectionErrorCode (p. 27)

(constant) CustomChannelError :CustomChannelErrorCode (p. 28)

The CustomChannelError codes enum.

Type:
- CustomChannelErrorCode (p. 28)

(constant) DisplayConfigError :DisplayConfigErrorCode (p. 30)

The DisplayConfigError codes enum.

Type:
- DisplayConfigErrorCode (p. 30)
(constant) FileStorageError : FileStorageErrorCode (p. 34)

The FileStorageError codes enum.

Type:
- FileStorageErrorCode (p. 34)

(constant) LogLevel : LogLevel (p. 37)

The available SDK log levels.

Type:
- LogLevel (p. 37)

(constant) MultiMonitorError : MultiMonitorErrorCode (p. 38)

The MultiMonitorError codes enum.

Type:
- MultiMonitorErrorCode (p. 38)

(constant) ResolutionError : ResolutionErrorCode (p. 40)

The ResolutionError codes enum.

Type:
- ResolutionErrorCode (p. 40)

(constant) version

The NICE DCV version with major, minor, patch, revision, extended, and versionStr.

Properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>major</td>
<td>integer</td>
<td>The major version number.</td>
</tr>
<tr>
<td>minor</td>
<td>integer</td>
<td>The minor version number.</td>
</tr>
<tr>
<td>patch</td>
<td>integer</td>
<td>The patch version number.</td>
</tr>
<tr>
<td>revision</td>
<td>integer</td>
<td>The revision number.</td>
</tr>
<tr>
<td>extended</td>
<td>string</td>
<td>The extended string.</td>
</tr>
<tr>
<td>versionStr</td>
<td>string</td>
<td>A concatenation of the major, minor, patch, and revision numbers in the</td>
</tr>
</tbody>
</table>
Type and callback definitions

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>form major.minor.patch</td>
<td>+build.revision</td>
<td></td>
</tr>
</tbody>
</table>

(constant) ScreenshotError :ScreenshotErrorCode (p. 40)

The ScreenshotError codes enum.

Type:
- ScreenshotErrorCode (p. 40)

(constant) WebcamError :WebcamErrorCode (p. 42)

The WebcamError codes enum.

Type:
- WebcamErrorCode (p. 42)

Type and callback definitions

List
- AudioErrorCode (p. 19)
- authenticationCallbacks (p. 19)
- AuthenticationErrorCode (p. 20)
- authErrorCallback(authentication, error) (p. 20)
- authPromptCredentialsCallback(authentication, challenge) (p. 21)
- authSuccessCallback(authentication, authenticationData) (p. 21)
- Channel (p. 22)
- ChannelErrorCode (p. 22)
- clipboardEventCallback(event) (p. 22)
- ClosingReasonErrorCode (p. 24)
- Colorspace (p. 24)
- connectionCallbacks (p. 24)
- ConnectionConfig (p. 26)
- ConnectionErrorCode (p. 27)
- createDirectory(path) (p. 28)
- CustomChannelErrorCode (p. 28)
- dataChannelCallback(info) (p. 28)
- deleteFile(path) (p. 29)
- deviceChangeEventCallback() (p. 29)
- disconnectCallback(reason) (p. 29)
- displayAvailabilityCallback(status, displayId) (p. 29)
- DisplayConfigErrorCode (p. 30)
- `displayLayoutCallback(serverWidth, serverHeight, heads)` (p. 30)
- `feature` (p. 31)
- `featuresUpdateCallback(featuresList)` (p. 31)
- `fileDownloadCallback(fileResource)` (p. 31)
- `filePrintedCallback(printResource)` (p. 32)
- `filestorage` (p. 33)
- `filestorageEnabledCallback(enabled)` (p. 34)
- `FileStorageErrorCode` (p. 34)
- `firstFrameCallback(resizeEnabled, relativeMouseModeEnabled, displayId)` (p. 35)
- `idleWarningNotificationCallback(disconnectionDateTime)` (p. 35)
- `collaboratorListCallback(collaborators)` (p. 35)
- `licenseNotificationCallback(notification)` (p. 36)
- `list(path)` (p. 37)
- `LogLevel` (p. 37)
- `Monitor` (p. 37)
- `MultiMonitorErrorCode` (p. 38)
- `qualityIndicatorStateCallback(state)` (p. 39)
- `renameDirectory(src, dest)` (p. 39)
- `renameFile(src, dest)` (p. 40)
- `ResolutionErrorCode` (p. 40)
- `retrieveFile(path)` (p. 40)
- `screenshotCallback(screenshot)` (p. 40)
- `ScreenshotErrorCode` (p. 40)
- `serverInfo` (p. 41)
- `stats` (p. 41)
- `storeFile(file, dir)` (p. 42)
- `WebcamErrorCode` (p. 42)

**AudioErrorCode**

The AudioError code enums available in the DCV module

- `SETTING_AUDIO_FAILED`
- `CHANNEL_NOT_AVAILABLE`

**Type:**

- `number`

**authenticationCallbacks**

Authentication callbacks

**Type:**

- `Object`
Properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>promptCredentials</td>
<td>authPromptCredentialsCallback (p. 21)</td>
<td>The callback function to be called when the user is challenged for credentials.</td>
</tr>
<tr>
<td>error</td>
<td>authErrorCallback (p. 20)</td>
<td>The callback function to be called when authentication fails.</td>
</tr>
<tr>
<td>success</td>
<td>authSuccessCallback (p. 21)</td>
<td>The callback function to be called when authentication succeeds.</td>
</tr>
</tbody>
</table>

AuthenticationErrorCode

The AuthenticationError code enums available in the DCV module

- INVALID_MESSAGE
- UNKNOWN_AUTH_MODE
- SESSION_NOT_AVAILABLE
- NO_SESSIONS
- WRONG_CREDENTIALS
- SASL_CHALLENGE
- SASL_AUTH_MECHANISM
- FAILED_COMMUNICATION
- AUTHENTICATION_REJECTED
- GENERIC_ERROR
- WRONG_CREDENTIALS_FORMAT
- WRONG_CREDENTIALS_TYPE
- UNREQUESTED_CREDENTIALS
- MISSING_CREDENTIAL

Type:

- number

authErrorCallback(authentication, error)

The callback function to be called when authentication fails.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authentication</td>
<td>Authentication (p. 60)</td>
<td>The Authentication object.</td>
</tr>
<tr>
<td>error</td>
<td>Object</td>
<td>The error object raised by the authentication process.</td>
</tr>
</tbody>
</table>
## Type and callback definitions

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>AuthenticationErrorCode (p. 20)</td>
<td>The error code.</td>
</tr>
<tr>
<td>message</td>
<td>string</td>
<td>The error message.</td>
</tr>
</tbody>
</table>

### authPromptCredentialsCallback(authentication, challenge)

The callback function to be called when the user is challenged for credentials. The user must answer the challenge by providing the requested credentials.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authentication</td>
<td>Authentication (p. 60)</td>
<td>The Authentication object.</td>
</tr>
<tr>
<td>challenge</td>
<td>Object</td>
<td>The challenge.</td>
</tr>
<tr>
<td>requiredCredentials</td>
<td>Array&lt;Object&gt;</td>
<td>array of requested credential objects.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the requested credential.</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td>The type of the requested credential.</td>
</tr>
</tbody>
</table>

### authSuccessCallback(authentication, authenticationData)

The callback function to be called when authentication succeeds.
Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authentication</td>
<td>Authentication (p. 60)</td>
<td>The Authentication object.</td>
</tr>
<tr>
<td>authenticationData</td>
<td>Array.&lt;Object&gt;</td>
<td>An array of objects that include NICE DCV session IDs and authentication tokens.</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sessionId</td>
<td>string</td>
<td>The NICE DCV session ID.</td>
</tr>
<tr>
<td>authToken</td>
<td>string</td>
<td>The authentication token for the NICE DCV session.</td>
</tr>
</tbody>
</table>
```

Channel

The available channels that can be specified.

Type:

- "clipboard" | "display" | "input" | "audio" | "filestorage"

ChannelErrorCode

The ChannelError code enums available in the DCV module

- ALREADY_OPEN
- INITIALIZATION_FAILED
- REJECTED

Type:

- number

clipboardEventCallback(event)

The callback function to be called when a clipboardEvent is generated.
Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Object</td>
<td>Information about the clipboard event.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>established</td>
<td>copy</td>
<td>paste</td>
</tr>
<tr>
<td>clipboard</td>
<td>Object</td>
<td>string</td>
<td>The data in the clipboard.</td>
</tr>
<tr>
<td>autoCopy</td>
<td>boolean</td>
<td>optional</td>
<td>Indicates whether automatic copying from the session clipboard to the local client clipboard is enabled.</td>
</tr>
<tr>
<td>maxData</td>
<td>number</td>
<td>optional</td>
<td>The maximum amount of data that can be placed</td>
</tr>
</tbody>
</table>
### ClosingReasonErrorCode

The ClosingReasonErrorCode code enums available in the DCV module

- TRANSPORT_ERROR
- NO_ERROR
- GENERIC_ERROR
- INTERNAL_SERVER_ERROR
- PROTOCOL_ERROR
- AUTHORIZATION_DENIED
- AUTHORIZATION_REVOKED
- ACCESS_REJECTED
- IDLE_TIMEOUT_EXPIRED
- DISCONNECT_BY_OWNER
- DISCONNECT_BY_USER
- EVICTED

**Type:**

- number

### Colorspace

The available colorspaces that can be specified.

**Type:**

- "RGB" | "YUV_REC601" | "YUV_REC709"

### connectionCallbacks

The callbacks that are available to be called in the event of a connection error.

**Type:**

- Object
### Properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disconnect</td>
<td>disconnectCallback (p. 29)</td>
<td>The callback function to be called when the connection ends.</td>
</tr>
<tr>
<td>displayLayout</td>
<td>displayLayoutCallback (p. 30)</td>
<td>The callback function to be called when the display layout or resolution is changed.</td>
</tr>
<tr>
<td>displayAvailability</td>
<td>displayAvailabilityCallback (p. 29)</td>
<td>The callback function to be called when a display's availability changes.</td>
</tr>
<tr>
<td>firstFrame</td>
<td>firstFrameCallback (p. 35)</td>
<td>The callback function to be called when the first frame is received from the NICE DCV server.</td>
</tr>
<tr>
<td>filePrinted</td>
<td>filePrintedCallback (p. 32)</td>
<td>The callback function to be called when a file is printed on the NICE DCV server.</td>
</tr>
<tr>
<td>fileDownload</td>
<td>fileDownloadCallback (p. 31)</td>
<td>The callback function to be called when a file is ready to be downloaded from the NICE DCV server.</td>
</tr>
<tr>
<td>dataChannel</td>
<td>dataChannelCallback (p. 28)</td>
<td>The callback function to be called when the NICE DCV server sends a notification about the availability of a data channel.</td>
</tr>
<tr>
<td>licenseNotification</td>
<td>licenseNotificationCallback (p. 36)</td>
<td>The callback function to be called when the NICE DCV server sends a notification about the license state.</td>
</tr>
<tr>
<td>idleWarningNotification</td>
<td>idleWarningNotificationCallback (p. 35)</td>
<td>The callback function to be called when the NICE DCV server sends an idle timeout warning.</td>
</tr>
<tr>
<td>collaboratorList</td>
<td>collaboratorListCallback (p. 35)</td>
<td>The callback function to be called when the NICE DCV server sends the list of collaborators (since NICE DCV Web Client SDK version 1.1.0).</td>
</tr>
<tr>
<td>qualityIndicatorState</td>
<td>qualityIndicatorStateCallback (p. 39)</td>
<td>The callback function to be called when the connection quality indicator changes state.</td>
</tr>
<tr>
<td>filestorageEnabled</td>
<td>filestorageEnabledCallback (p. 34)</td>
<td>The callback function to be called when file storage is enabled or disabled.</td>
</tr>
</tbody>
</table>
## ConnectionConfig

The configuration for a NICE DCV connection.

**Type:**
- Object

**Properties:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>string</td>
<td>The host name and port of the running NICE DCV server in the following format: <code>https://dcv_host_address:port</code>. For example: <code>https://my-dcv-server:8443</code>.</td>
</tr>
<tr>
<td>sessionId</td>
<td>string</td>
<td>The NICE DCV session ID.</td>
</tr>
<tr>
<td>authToken</td>
<td>string</td>
<td>The authentication token to use when connecting to the server.</td>
</tr>
<tr>
<td>baseUrl</td>
<td>string</td>
<td>The absolute or relative URL from which to load SDK files.</td>
</tr>
<tr>
<td>resourceBaseUrl</td>
<td>string</td>
<td>The absolute or relative URL from which to access DCV resources.</td>
</tr>
<tr>
<td>enabledChannels</td>
<td>Array.&lt;Channel (p. 22)&gt;</td>
<td>Indicates the list of channels that can be enabled. If not specified or an empty array is provided, it defaults to all the available channels.</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>losslessColorspace</td>
<td>Colorspace (p. 24)</td>
<td>Indicates the colorspace that will be used. If not specified, it defaults to &quot;RGB&quot;.</td>
</tr>
<tr>
<td>divId</td>
<td>string</td>
<td>The ID of the div object in the HTML DOM where SDK should create the canvas with the remote stream.</td>
</tr>
<tr>
<td>volumeLevel</td>
<td>integer</td>
<td>The preferred volume level. The valid range is 0 to 100.</td>
</tr>
<tr>
<td>clipboardAutoSync</td>
<td>boolean</td>
<td>Indicates whether automatic copying from the NICE DCV session clipboard to the local client clipboard is enabled for compatible web browsers.</td>
</tr>
<tr>
<td>dynamicAudioTuning</td>
<td>boolean</td>
<td>Indicates whether to dynamically tune the audio based on the NICE DCV server audio settings when a connection is established.</td>
</tr>
<tr>
<td>clientHiDpiScaling</td>
<td>boolean</td>
<td>Indicates whether to scale the canvas based on the client's DPI.</td>
</tr>
<tr>
<td>highColorAccuracy</td>
<td>boolean</td>
<td>Indicates whether high color accuracy should be used if available. If not specified, it defaults to false.</td>
</tr>
<tr>
<td>enableWebCodecs</td>
<td>Boolean</td>
<td>Indicates whether WebCodecs should be used if available. Defaults to false if not specified.</td>
</tr>
<tr>
<td>observers</td>
<td>connectionCallbacks (p. 24)</td>
<td>The callback functions to call for events that are related to the connection.</td>
</tr>
<tr>
<td>callbacks</td>
<td>connectionCallbacks (p. 24)</td>
<td>The same as the observers property, but each callback includes the Connection (p. 43) object as the first parameter.</td>
</tr>
</tbody>
</table>

**ConnectionErrorCode**

The ConnectionError code enums available in the DCV module

- ALREADY_OPEN
- INVALID_CONFIG
- INITIALIZATION_FAILED
- REJECTED
- MAIN_CHANNEL_ALREADY_OPEN
• GENERIC_ERROR (since DCV Server 2021.0)
• INTERNAL_SERVER_ERROR (since DCV Server 2021.0)
• AUTHENTICATION_FAILED (since DCV Server 2021.0)
• PROTOCOL_ERROR (since DCV Server 2021.0)
• INVALID_SESSION_ID (since DCV Server 2021.0)
• INVALID_CONNECTION_ID (since DCV Server 2021.0)
• CONNECTION_LIMIT_REACHED (since DCV Server 2021.0)
• SERVER_UNREACHABLE (since DCV Server 2022.1)

Type:

• number

createDirectory(path)

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>string</td>
<td>The absolute path on the server where we want to create a directory. It should also include the name of the target directory.</td>
</tr>
</tbody>
</table>

CustomChannelErrorCode

The CustomChannelError code enums available in the DCV module

• TRANSPORT_ERROR

Type:

• number

dataChannelCallback(info)

The callback function to be called when the NICE DCV server sends a notification about the availability of a data channel.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>Object</td>
<td>Information about the data channel.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The name of</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>the data channel.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>token</td>
<td>string</td>
<td>The authentication token for the data channel.</td>
</tr>
</tbody>
</table>

### deleteFile(path)

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>string</td>
<td>The absolute path on the server identifying the file we want to delete.</td>
</tr>
</tbody>
</table>

### deviceChangeEventCallback()

The callback function to be called when an deviceChange event is triggered.

### disconnectCallback(reason)

The callback function to be called when the connection ends.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason</td>
<td>Object</td>
<td>The reason for the disconnection.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>number</td>
<td>The reason code.</td>
</tr>
<tr>
<td>message</td>
<td>string</td>
<td>The reason message.</td>
</tr>
</tbody>
</table>

### displayAvailabilityCallback(status, displayId)

The callback function to be called when a display's availability changes.
Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Object</td>
<td>The status of the display.</td>
</tr>
<tr>
<td>enabled</td>
<td>boolean</td>
<td>Indicates if the display is enabled.</td>
</tr>
<tr>
<td>closed</td>
<td>boolean</td>
<td>Indicates if the display is closed.</td>
</tr>
<tr>
<td>displayId</td>
<td>number</td>
<td>The identifier for the display.</td>
</tr>
</tbody>
</table>

DisplayConfigErrorCode

The DisplayConfigError code enums available in the DCV module

- INVALID_ARGUMENT
- UNSUPPORTED_OPERATION
- NO_CHANNEL

Type:

- number

displayLayoutCallback(serverWidth, serverHeight, heads)

The callback function to be called when the display layout or resolution is changed.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serverWidth</td>
<td>number</td>
<td>The width (in pixels) of the primary display.</td>
</tr>
<tr>
<td>serverHeight</td>
<td>number</td>
<td>The height (in pixels) of the primary display.</td>
</tr>
<tr>
<td>heads</td>
<td>Array.&lt;Monitor (p. 37)&gt;</td>
<td>The display heads supported by the NICE DCV server.</td>
</tr>
</tbody>
</table>
feature

The feature values.

- **display** - Indicates the availability of a single-display video stream.
- **display-multi** - Indicates the availability of a multi-display video stream.
- **high-color-accuracy** - Indicates the availability of high color accuracy (since NICE DCV Web Client SDK version 1.1.0).
- **mouse** - Indicates the availability of mouse functionality.
- **keyboard** - Indicates the availability of keyboard functionality.
- **keyboard-sas** - Indicates the availability of SAS sequence (Control + Alt + Delete) functionality.
- **relative-mouse** - Indicates the availability of relative mouse mode.
- **clipboard-copy** - Indicates the availability of clipboard copy functionality from NICE DCV server to the client.
- **clipboard-paste** - Indicates the availability of clipboard paste functionality from the client to the NICE DCV server.
- **audio-in** - Indicates the availability of audio input functionality using the microphone.
- **audio-out** - Indicates the availability of audio playback functionality.
- **webcam** - Indicates the availability of webcam streaming functionality.
- **file-download** - Indicates availability of file download functionality from the NICE DCV server to the client.
- **file-upload** - Indicates availability of file upload functionality from the client to the NICE DCV server.

**Type:**

- **string**

featuresUpdateCallback(featuresList)

The callback function to be called when a feature's status changes.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>featuresList</td>
<td>Array.&lt;feature (p. 31)&gt;</td>
<td>An array of features that have changed.</td>
</tr>
</tbody>
</table>

fileDownloadCallback(fileResource)

The callback function to be called when a file is ready to be downloaded from the NICE DCV server.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileResource</td>
<td>Object</td>
<td>Information about the file that is ready for download.</td>
</tr>
</tbody>
</table>
filePrintedCallback(printResource)

The callback function to be called when a file is printed on the NICE DCV server.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printResource</td>
<td>Object</td>
<td>Information about the printed file.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>The identifier for the printed file.</td>
</tr>
<tr>
<td>url</td>
<td>string</td>
<td>The URL to use to download the file.</td>
</tr>
<tr>
<td>domain</td>
<td>string</td>
<td>The resource domain.</td>
</tr>
<tr>
<td>token</td>
<td>string</td>
<td>The authentication token to use to download the file. The token is also included in the URL.</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>domain</td>
<td>string</td>
<td>The resource domain. In this case, printer.</td>
</tr>
<tr>
<td>token</td>
<td>string</td>
<td>The authentication token to use to download the printed file. The token is also included in the URL.</td>
</tr>
</tbody>
</table>

**filestorage**

Object that allows for exploring and performing actions on the file system.

**Type:**
- Object

**Properties:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>list (p. 37)</td>
<td>Function that allows to list the items (files and directories) present at the supplied path on the server.</td>
</tr>
<tr>
<td>createDirectory</td>
<td>createDirectory (p. 28)</td>
<td>Function that allows to create a directory at the specified path on the server.</td>
</tr>
<tr>
<td>retrieveFile</td>
<td>retrieveFile (p. 40)</td>
<td>Function that allows to locally download a file at the specified path on the server.</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>deleteFile</td>
<td>deleteFile (p. 29)</td>
<td>Function that allows to delete a file at the specified path on the server.</td>
</tr>
<tr>
<td>renameFile</td>
<td>renameFile (p. 40)</td>
<td>Function that allows to rename a file from the specified source path to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specified destination path.</td>
</tr>
<tr>
<td>renameDirectory</td>
<td>renameDirectory (p. 39)</td>
<td>Function that allows to rename a directory from the specified source path to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the absolute destination path.</td>
</tr>
<tr>
<td>storeFile</td>
<td>storeFile (p. 42)</td>
<td>Function that allows to upload a local file to the supplied path on the server.</td>
</tr>
</tbody>
</table>

**filestorageEnabledCallback(enabled)**

The callback function to be called when file storage is enabled. Lazy channel on Internet Explorer 11 only.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>boolean</td>
<td>Indicates whether file storage is enabled.</td>
</tr>
</tbody>
</table>

**FileStorageErrorCode**

The FileStorageError code enums available in the DCV module

- CANCELLED
- ABORTED
- INVALID_ARGUMENT
- NOT_IMPLEMENTED
- ERROR
- ALREADY_EXIST
- NOT_FOUND

**Type:**

- number
firstFrameCallback(resizeEnabled, relativeMouseModeEnabled, displayId)

The callback function to be called when the first frame is received from the NICE DCV server. Emitted for each display.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resizeEnabled</td>
<td>boolean</td>
<td>Indicates whether the server supports resizing the client display layout.</td>
</tr>
<tr>
<td>relativeMouseModeEnabled</td>
<td>boolean</td>
<td>Indicates whether the server supports relative mouse mode.</td>
</tr>
<tr>
<td>displayId</td>
<td>number</td>
<td>The identifier for the display.</td>
</tr>
</tbody>
</table>

idleWarningNotificationCallback(disconnectionDateTime)

The callback function to be called when the NICE DCV server sends an idle timeout warning.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disconnectionDateTime</td>
<td>Date</td>
<td>The date and time of the disconnection.</td>
</tr>
</tbody>
</table>

collaboratorListCallback(collaborators)

The callback function to be called when the NICE DCV server sends the list of collaborators.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collaborators</td>
<td>Array.&lt;Object&gt;</td>
<td>A list of objects containing information on collaborators.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>username</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>owner</td>
<td>boolean</td>
</tr>
</tbody>
</table>

The username of the collaborator.

Indicates whether the collaborator is the owner.
### licenseNotificationCallback(notification)

The callback function to be called when the NICE DCV server sends a notification about the license state.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notification</td>
<td>Object</td>
<td>The notification.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>product</td>
<td>string</td>
<td>The DCV product.</td>
</tr>
<tr>
<td>status</td>
<td>string</td>
<td>The status of the license.</td>
</tr>
<tr>
<td>message</td>
<td>string</td>
<td>A message.</td>
</tr>
<tr>
<td>leftDays</td>
<td>number</td>
<td>The number of days before the license expires.</td>
</tr>
<tr>
<td>isDemo</td>
<td>boolean</td>
<td>Indicates if the license is a demo license.</td>
</tr>
<tr>
<td>numUnlic</td>
<td>number</td>
<td>The number of</td>
</tr>
</tbody>
</table>
### Name Type Description

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unlicensed</td>
<td></td>
<td>unlicensed connections.</td>
</tr>
<tr>
<td>licensing</td>
<td>string</td>
<td>The licensing mode.</td>
</tr>
<tr>
<td>documentationUrl</td>
<td>string</td>
<td>The URL for the documentation.</td>
</tr>
</tbody>
</table>

#### list(path)

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>string</td>
<td>The absolute path on the server of which we want to list the content.</td>
</tr>
</tbody>
</table>

#### LogLevel

The available SDK log levels.

**Type:**

- TRACE | DEBUG | INFO | WARN | ERROR | SILENT

#### Monitor

**Type:**

- Object

**Properties:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the display head.</td>
</tr>
<tr>
<td>rect</td>
<td>Object</td>
<td>Information about the display head.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>number</td>
<td>The initial x</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>coordinate</td>
<td>for the display head.</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>number</td>
<td>The initial y coordinate for the display head.</td>
</tr>
<tr>
<td>width</td>
<td>number</td>
<td>The width (in pixels) of the display head.</td>
</tr>
<tr>
<td>height</td>
<td>number</td>
<td>The height (in pixels) of the display head.</td>
</tr>
<tr>
<td>primary</td>
<td>boolean</td>
<td>Indicates whether the display head is the primary display head. This is determined from the remote operating system if available.</td>
</tr>
<tr>
<td>dpi</td>
<td>number</td>
<td>The DPI of the display head.</td>
</tr>
</tbody>
</table>

**MultiMonitorErrorCode**

The MultiMonitorError code enums available in the DCV module

- NO_DISPLAY_CHANNEL
- MAX_DISPLAY_NUMBER_REACHED
- INVALID_ARGUMENT
- DISPLAY_NOT_OPENED_BY_SERVER
- REQUEST_TIMEOUT
- GENERIC_ERROR
- NO_ERROR

**Type:**

- number
**qualityIndicatorStateCallback(state)**

The callback function to be called when the connection quality indicator changes state.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>Array.&lt;Object&gt;</td>
<td>Information about the connection quality.</td>
</tr>
</tbody>
</table>

- **name**
  - Type: string
  - Description: The name of the indicator.

- **status**
  - Type: NORMAL | WARNING | CRITICAL
  - Description: Description of the status.

- **changed**
  - Type: boolean
  - Description: Indicates whether the status changed.

**renameDirectory(src, dest)**

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>string</td>
<td>The absolute source path on the server identifying the directory we want to rename.</td>
</tr>
<tr>
<td>dest</td>
<td>string</td>
<td>The absolute destination path on the server specifying the target path and directory name.</td>
</tr>
</tbody>
</table>
renameFile(src, dest)

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>string</td>
<td>The absolute source path on the server identifying the file we want to rename.</td>
</tr>
<tr>
<td>dest</td>
<td>string</td>
<td>The absolute destination path on the server specifying the target path and file name.</td>
</tr>
</tbody>
</table>

ResolutionErrorCode

The ResolutionError code enums available in the DCV module

- INVALID_ARGUMENT
- NO_CHANNEL

Type:

- number

retrieveFile(path)

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>string</td>
<td>The absolute path on the server identifying the file we want to download locally.</td>
</tr>
</tbody>
</table>

screenshotCallback(screenshot)

The callback function to be called when a screenshot is available.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>screenshot</td>
<td>byte[]</td>
<td>Screenshot buffer in PNG format, or null if screenshot retrieval failed.</td>
</tr>
</tbody>
</table>

ScreenshotErrorCode

The ScreenshotError code enums available in the DCV module
• **NO_CHANNEL**
• **GENERIC_ERROR**

**Type:**
- number

**serverInfo**

**Type:**
- Object

**Properties:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the software.</td>
</tr>
<tr>
<td>version</td>
<td>Object</td>
<td>The software version number.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>major</td>
<td>number</td>
<td>The major version number.</td>
</tr>
<tr>
<td>minor</td>
<td>number</td>
<td>The minor version number.</td>
</tr>
<tr>
<td>revision</td>
<td>number</td>
<td>The revision version number.</td>
</tr>
<tr>
<td>os</td>
<td>string</td>
<td>The OS.</td>
</tr>
<tr>
<td>arch</td>
<td>string</td>
<td>The architecture.</td>
</tr>
<tr>
<td>hostname</td>
<td>string</td>
<td>The hostname.</td>
</tr>
</tbody>
</table>

**stats**

**Type:**
- Object
Properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fps</td>
<td>number</td>
<td>The current frames per second.</td>
</tr>
<tr>
<td>traffic</td>
<td>number</td>
<td>The current traffic in bit/s.</td>
</tr>
<tr>
<td>peakTraffic</td>
<td>number</td>
<td>The peak of traffic in bit/s since the connection was established.</td>
</tr>
<tr>
<td>latency</td>
<td>number</td>
<td>The current latency in ms.</td>
</tr>
<tr>
<td>currentChannels</td>
<td>number</td>
<td>The number of channels that have been opened since the connection was established.</td>
</tr>
<tr>
<td>openedChannels</td>
<td>number</td>
<td>The number of currently opened channels.</td>
</tr>
<tr>
<td>channelErrors</td>
<td>number</td>
<td>The number of channels which have reported an error.</td>
</tr>
</tbody>
</table>

storeFile(file, dir)

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>File</td>
<td>The file object (for more information see <a href="https://developer.mozilla.org/en-US/docs/Web/API/File">https://developer.mozilla.org/en-US/docs/Web/API/File</a>) we want to upload to the server.</td>
</tr>
<tr>
<td>dir</td>
<td>string</td>
<td>The absolute path on the server where we want to upload the file.</td>
</tr>
</tbody>
</table>

WebcamErrorCode

The WebcamError code enums available in the DCV module

- SETTING_WEBCAM_FAILED
- CHANNEL_NOT_AVAILABLE

Type:

- number
Connection Class

The Connection Class obtained by calling the `connect` method (p. 14) of the `dcv` module. For an example showing how to use it, see the Getting started (p. 4) section.

**Exposes**

- Methods (p. 13)

**Methods**

**List**

- `attachDisplay(win, displayConf) → {Promise.<number>|Promise.<{code: MultiMonitorErrorCode, message: string}>} (p. 44)
- `captureClipboardEvents(enabled, win, displayId) → {void} (p. 44)
- `detachDisplay(displayId) → {void} (p. 45)
- `disconnect() → {void} (p. 45)
- `disconnectCollaborator(connectionId) → {void} (p. 46)
- `enableDisplayQualityUpdates(enable) → {void} (p. 46)
- `enterRelativeMouseMode() → {void} (p. 46)
- `getConnectedDevices() → {Promise.<Array.<MediaDeviceInfo>>|Promise.<{message: string}>} (p. 47)
- `getClipboardEvents() → {void} (p. 44)
- `getFileExplorer() → {Promise.<filestorage>|Promise.<{code: ChannelErrorCode, message: string}>} (p. 47)
- `getServerInfo() → {serverInfo} (p. 47)
- `getScreenshot() → {Promise|Promise.<{code: ScreenshotErrorCode, message: string}>} (p. 47)
- `getStats() → {stats} (p. 48)
- `latchModifierKey(key, location, isDown) → {boolean} (p. 48)
- `openChannel(name, authToken, callbacks, namespace) → {Promise|Promise.<{code: ChannelErrorCode, message: string}>} (p. 48)
- `queryFeature(featureName) → {Promise.<{enabled: boolean, remote?: string, autoCopy?: boolean, autoPaste?: boolean, serviceStatus?: string, available?: boolean}>|Promise.<{message: string}>} (p. 49)
- `registerKeyboardShortcuts(shortcuts) → {void} (p. 49)
- `requestDisplayConfig(highColorAccuracy) → {Promise|Promise.<{code: DisplayConfigErrorCode, message: string}>} (p. 52)
- `requestDisplayLayout(layout) → {Promise|Promise.<{code: ResolutionErrorCode, message: string}>} (p. 53)
- `requestResolution(width, height) → {Promise|Promise.<{code: ResolutionErrorCode, message: string}>} (p. 53)
- `sendKeyboardEvent(event) → {boolean} (p. 53)
- `sendKeyboardShortcut(shortcut) → {void} (p. 54)
- `setDisplayQuality(min, maxopt) → {void} (p. 55)
- `setDisplayScale(scaleRatio, displayId) → {Promise|Promise.<{code: ResolutionErrorCode, message: string}>} (p. 56)
- `setKeyboardQuirks(quirks) → {void} (p. 56)
- `setMaxDisplayResolution(maxWidth, maxHeight) → {void} (p. 57)
• setMicrophone(enable) → {Promise|Promise.<{code: AudioErrorCode, message: string}>} (p. 57)
• setMinDisplayResolution(minWidth, minHeight) → {void} (p. 58)
• setUpLoadBandwidth(value) → {number} (p. 58)
• setVolume(volume) → {void} (p. 59)
• setWebcam(enable, deviceId) → {Promise|Promise.<{code: WebcamErrorCode, message: string}>} (p. 59)
• syncClipboards() → {boolean} (p. 59)

attachDisplay(win, displayConf) → {Promise.<number>|Promise.<{code: MultiMonitorErrorCode (p. 38), message: string}>}

Attaches a specific display to a window. You can't attach the main display. If successful, the function returns the displayId.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>win</td>
<td>Object</td>
<td>The window to which the display must be attached.</td>
</tr>
<tr>
<td>displayConf</td>
<td>Object</td>
<td>The configuration of the display.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>displayId</td>
<td>number</td>
<td>&lt;optional&gt;</td>
<td>The ID of the display.</td>
</tr>
<tr>
<td>displayDivName</td>
<td></td>
<td></td>
<td>The name of the display div.</td>
</tr>
</tbody>
</table>

Returns:

Promise. If rejected, the promise returns an error object.

Type

Promise.<number> | Promise.<{code: MultiMonitorErrorCode (p. 38), message: string}>

captureClipboardEvents(enabled, win, displayId) → {void}

Starts or stops listening to copy-paste events. In the case of interactive clipboards (always in the case of paste) we need to start listening to the copy/paste events. It could be useful to start and stop listening only when it is needed, for example, when a modal is shown.
Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>boolean</td>
<td></td>
<td>To start listening to events, specify true. To stop listening to events, specify false.</td>
</tr>
<tr>
<td>win</td>
<td>Object</td>
<td>&lt;optional&gt;</td>
<td>The window in which to listen for events. If omitted, the default window is used.</td>
</tr>
<tr>
<td>displayId</td>
<td>number</td>
<td>&lt;optional&gt;</td>
<td>The ID of the display that should listen the events. If omitted, the default display of the window is used.</td>
</tr>
</tbody>
</table>

Returns:
Type

`detachDisplay(displayId) → {void}`

Detaches a specific display. The main display cannot be detached.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>displayId</td>
<td>number</td>
<td>The ID of the display to detach.</td>
</tr>
</tbody>
</table>

Returns:
Type

`disconnect() → {void}`

Disconnects from the NICE DCV server and closes the connection.

Returns:
Type

`void`
**disconnectCollaborator(connectionId) → {void}**

Requests disconnect of collaborator connected with the provided connection id (since NICE DCV Web Client SDK version 1.1.0).

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectionId</td>
<td>boolean</td>
<td>The id of the connection that will be disconnected.</td>
</tr>
</tbody>
</table>

**Returns:**

Type

void

**enableDisplayQualityUpdates(enable) → {void}**

Enables or disables display quality updates for streaming areas that do not receive updates. Disabling display quality updates reduces bandwidth usage, but it also decreases the display quality.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>boolean</td>
<td>To enable display quality updates, specify true. To disable display quality updates, specify false.</td>
</tr>
</tbody>
</table>

**Returns:**

Type

void

**enterRelativeMouseMode() → {void}**

Enables relative mouse mode.

**Returns:**

Type

void
**getConnectedDevices() → {Promise.<Array.<MediaDeviceInfo>>|Promise.<{message: string}>}**

Requests a list of the media devices connected to the client computer.

**Returns:**

If successful, it returns a Promise that resolves to an array of MediaDeviceInfo objects. For more information, see https://developer.mozilla.org/en-US/docs/Web/API/MediaDeviceInfo. If rejected, the promise returns an error object.

**Type**

Promise.<Array.<MediaDeviceInfo>> | Promise.<{message: string}>

**getFileExplorer() → {Promise.<filestorage (p. 33)>|Promise.<{code: ChannelErrorCode (p. 22), message: string}>}**

Gets an object to manage the NICE DCV server's file storage.

**Returns:**

Promise. Resolves to the file explorer object if fulfilled, or an error object if rejected.

**Type**

Promise.<filestorage (p. 33)> | Promise.<{code: ChannelErrorCode (p. 22), message: string}>

**getServerInfo() → {serverInfo (p. 41)}**

Gets information about the NICE DCV server.

**Returns:**

Information about the server software.

**Type**

serverInfo (p. 41)

**getScreenshot() → {Promise|Promise.<{code: ScreenshotErrorCode (p. 40), message: string}>}**

Retrieves the screenshot of the remote desktop in PNG format. The screenshot will be returned in the screenshotCallback (p. 40) observer. null will be returned instead in case of failures.

**Returns:**

Promise that resolves if the request is processed. If rejected we receive an error object.

**Type**

Promise | Promise.<{code: ScreenshotErrorCode (p. 40), message: string}>
getStats() → {stats (p. 41)}

Gets statistics about the NICE DCV server.

**Returns:**

Information about the streaming statistics.

**Type**

stats (p. 41)

latchModifierKey(key, location, isDown) → {boolean}

Sends a single keyboard `keydown` or `keyup` event for an allowed modifier.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>Control</td>
<td>Alt</td>
</tr>
<tr>
<td>location</td>
<td>KeyboardEvent.location</td>
<td>The key's location. For more information, see <a href="https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/location">https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/location</a>.</td>
</tr>
<tr>
<td>isDown</td>
<td>boolean</td>
<td>If the key event to inject is a keydown (true) or a keyup (false).</td>
</tr>
</tbody>
</table>

**Returns:**

If the requested combination is valid, the function returns `true`, otherwise it returns `false`.

**Type**

boolean

openChannel(name, authToken, callbacks, namespace) → {Promise|Promise.<{code: ChannelErrorCode (p. 22), message: string}>}

Opens a custom data channel on the connection if it was created on the NICE DCV Server.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the channel.</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>authToken</td>
<td>string</td>
<td>The authentication token to use to connect to the channel.</td>
</tr>
<tr>
<td>callbacks</td>
<td>Object</td>
<td>The onMessage and onClose callbacks functions to call.</td>
</tr>
<tr>
<td>namespace</td>
<td>string</td>
<td>The namespace of the channel. Available since NICE DCV Web Client SDK 1.2.0 and NICE DCV Server 2022.1.</td>
</tr>
</tbody>
</table>

Returns:

Promise. If rejected we receive an error object.

Type

Promise | Promise.<{code: ChannelErrorCode (p. 22), message: string}>

queryFeature(featureName) → {Promise.<{enabled: boolean, remote?: string, autoCopy?: boolean, autoPaste?: boolean, serviceStatus?: string, available?: boolean}>|Promise.<{message: string}>

Queries the status of a specific NICE DCV server feature.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>featureName</td>
<td>feature (p. 31)</td>
<td>The name of the feature to query.</td>
</tr>
</tbody>
</table>

Returns:

Promise. If resolved, the function returns a status object that always contains an enabled property, and possibly also other properties. If rejected, the function returns an error object.

Type

{Promise.<{enabled: boolean, remote?: string, autoCopy?: boolean, autoPaste?: boolean, serviceStatus?: string, available?: boolean}> | Promise.<{message: string}>

registerKeyboardShortcuts(shortcuts) → {void}

Registers keyboard shortcuts.
### Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shortcuts</td>
<td>Array.&lt;Object&gt;</td>
<td>The array of keys and mappings to register.</td>
</tr>
<tr>
<td>sequence</td>
<td>Array.&lt;Object&gt;</td>
<td>The keyboard shortcut to register.</td>
</tr>
<tr>
<td>key</td>
<td>KeyboardEvent.key</td>
<td>The value of the key pressed by the user. For more information, see <a href="https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/key">https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/key</a>.</td>
</tr>
<tr>
<td>location</td>
<td>KeyboardEvent.location</td>
<td>The array of keys to send. The location of the key on the keyboard. For</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>output</td>
<td>Array.&lt;Object&gt;</td>
<td>The intended action to be performed by the shortcut.</td>
</tr>
</tbody>
</table>

**More information:** [see](https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/)

### `output`  
- **Array.<Object>**
  - **intended action**
    - **to be performed by the shortcut.**

### `key`  
- **KeyboardEvent.key**
  - **value**
    - of the key pressed by the user.
    - For more information, see [see](https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/key).

### `location`  
- **KeyboardEvent.location**
  - **array**
    - of keys to
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>

Returns:

Type

void

requestDisplayConfig(highColorAccuracy) → {Promise|Promise.<{code: DisplayConfigErrorCode (p. 30), message: string}>}

Asks for an updated display config from the NICE DCV Server. Available since NICE DCV Web Client SDK 1.1.0 and NICE DCV Server 2022.0.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| highColorAccuracy  | boolean  | Whether or not high color accuracy should be requested.

Returns:

Promise. If rejected, the promise returns an error object.
**Methods**

**Type**

`Promise | Promise.{code: DisplayConfigErrorCode (p. 30), message: string}`

**requestDisplayLayout(layout)**  

→ `{Promise|Promise.{code: ResolutionErrorCode (p. 40), message: string}}`

Requests an updated display layout for the connection.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>

**Returns:**

Promise. If rejected we receive an error object.

**Type**

`Promise | Promise.{code: ResolutionErrorCode (p. 40), message: string}`

**requestResolution(width, height)**  

→ `{Promise|Promise.{code: ResolutionErrorCode (p. 40), message: string}}`

Requests an updated display resolution from the NICE DCV server.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>number</td>
<td>The width to request in pixels. The minimum allowed value is 0.</td>
</tr>
<tr>
<td>height</td>
<td>number</td>
<td>The height to request in pixels. The minimum allowed value is 0.</td>
</tr>
</tbody>
</table>

**Returns:**

Promise. If rejected, the promise returns an error object.

**Type**

`Promise | Promise.{code: ResolutionErrorCode (p. 40), message: string}`

**sendKeyboardEvent(event)**  

→ `{boolean}`

keypress, and keyup. For more information about these events, see https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent#events.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>KeyboardEvent</td>
<td>The keyboard event to send.</td>
</tr>
</tbody>
</table>

**Returns:**

If the event is not valid, the function returns false. If the event is valid, the function returns true.

Type

    boolean

`sendKeyboardShortcut(shortcut) → {void}`

Sends a keyboard shortcut. Use this function to send a full `keydown` or `keyup` sequence. For example, sending Ctrl + Alt + Del sends the `keydown` events for all the keys followed by the `keyup` events. Use this function even if you want to send a single key.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shortcut</td>
<td>Array.&lt;Object&gt;</td>
<td>The array of keys to send.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>KeyboardEvent.key</td>
<td>key value of the key pressed by the user. For more information, see <a href="https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/key">https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/key</a>.</td>
</tr>
<tr>
<td>location</td>
<td>KeyboardEvent.location</td>
<td>array of keys</td>
</tr>
</tbody>
</table>
### send

The location of the key on the keyboard. For more information, see https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/location.

**Returns:**

Type

```
void
```

### setDisplayQuality(min, maxopt) → {void}

Sets the image quality to use for the connection. Valid range is 0 to 100, with 1 being the lowest image quality and 100 being the highest image quality. Specify 0 to retain the current value.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>number</td>
<td></td>
<td>The minimum image quality.</td>
</tr>
<tr>
<td>max</td>
<td>number</td>
<td>&lt;optional&gt;</td>
<td>The maximum image quality.</td>
</tr>
</tbody>
</table>

**Returns:**

Type

```
void
```
setDisplayScale(scaleRatio, displayId) \to Promise\langle\{code: ResolutionErrorCode (p. 40), message: string}\rangle

Notifies the NICE DCV that the display is scaled on the client side. Use this to notify the server that it needs to scale mouse events to match the client's display ratio.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scaleRatio</td>
<td>float</td>
<td>The scaling ratio to use. Must be a strictly positive number.</td>
</tr>
<tr>
<td>displayId</td>
<td>number</td>
<td>The ID of the display to scale.</td>
</tr>
</tbody>
</table>

**Returns:**

Promise. If rejected, the promise returns an error object.

**Type**

Promise | Promise.\langle\{code: ResolutionErrorCode (p. 40), message: string\}\rangle

setKeyboardQuirks(quirks) \to \{void\}

Sets keyboard quirks for the client computer.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>quirks</td>
<td>Object</td>
<td>The keyboard quirks to enable or disable.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>macOptionToAlt</td>
<td>boolean</td>
<td>To map the Option key to Alt for macOS, specify true. Otherwise, specify false.</td>
</tr>
<tr>
<td>macCommandToControl</td>
<td>boolean</td>
<td>To map the Command key to }</td>
</tr>
</tbody>
</table>
### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl for macOS, specify true, otherwise, specify false.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Returns:**

Type

void

---

**setMaxDisplayResolution**(maxWidth, maxHeight) → {void}

Sets the maximum display resolution to use for the connection.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxWidth</td>
<td>number</td>
<td>The maximum display width in pixels. The minimum allowed value is 0.</td>
</tr>
<tr>
<td>maxHeight</td>
<td>number</td>
<td>The maximum display height in pixels. The minimum allowed value is 0.</td>
</tr>
</tbody>
</table>

**Returns:**

Type

void

---

**setMicrophone**(enable) → {Promise|Promise.<{code: AudioErrorCode (p. 19), message: string}>}

Enables or disables the microphone.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>boolean</td>
<td>To enable the microphone, specify true. To disable the microphone, specify false.</td>
</tr>
</tbody>
</table>
Returns:

Promise. If rejected, the promise returns an error object.

Type

Promise | Promise.<{code: AudioErrorCode (p. 19), message: string}>

**setMinDisplayResolution(minWidth, minHeight) → {void}**

Sets the minimum display resolution to use for the connection. Some applications might require a minimum display resolution. If the minimum required resolution is larger than the maximum resolution supported by the client, a resize strategy is used. Use this function carefully. The resize strategy could cause a less precise mouse and touch input system.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minWidth</td>
<td>number</td>
<td>The minimum display width in pixels. The minimum allowed value is 0.</td>
</tr>
<tr>
<td>minHeight</td>
<td>number</td>
<td>The minimum display height in pixels. The minimum allowed value is 0.</td>
</tr>
</tbody>
</table>

Returns:

Type

void

**setUploadBandwidth(value) → {number}**

Sets the maximum bandwidth to use for uploading files to the NICE DCV server.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>number</td>
<td>The maximum bandwidth limit in kbps. Valid range is 1024 kbps to 102400 kbps.</td>
</tr>
</tbody>
</table>

Returns:

- The set bandwidth limit. `null` if the file storage feature is disabled on the server.

Type

number
setVolume(volume) \rightarrow \{\text{void}\}

Sets the volume level to use for audio. Valid range is 0 to 100, with 0 being the lowest volume and 100 being the highest volume.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>volume</td>
<td>number</td>
<td>The volume level to use.</td>
</tr>
</tbody>
</table>

**Returns:**

Type

void

setWebcam(enable, deviceId) \rightarrow \{\text{Promise} | \text{Promise}.\{\text{code: WebcamErrorCode (p. 42), message: string}\}\}

Enables or disables the webcam.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>boolean</td>
<td>To enable the webcam, specify true. To disable the webcam, specify false.</td>
</tr>
<tr>
<td>deviceId</td>
<td>string</td>
<td>The device ID of the webcam.</td>
</tr>
</tbody>
</table>

**Returns:**

Promise that, if successful, resolves to the attached/detached webcam deviceId. If rejected, the promise returns an error object.

Type

Promise | Promise.\{\text{code: WebcamErrorCode (p. 42), message: string}\}\n
syncClipboards() \rightarrow \{\text{boolean}\}

Synchronizes the local client clipboard with the remote NICE DCV server clipboard. Autocopy must be supported by the browser.

**Returns:**

If the clipboards have been synchronized, the function returns true. If the clipboards have not been synchronized, or if the browser does not support autocopy, the function returns false.
The Authentication Class must be used to obtain an authentication token by calling the `authenticate` method (p. 13) of the `dcv` module. For an example showing how to use it, see the Getting started (p. 4) section.

Exposes
- Methods (p. 13)

Methods

List
- `retry() → {void} (p. 60)
- `sendCredentials(credentials) → {void} (p. 60)

`retry() → {void}`

Retries the authentication process.

Returns:
Type

`void`

`sendCredentials(credentials) → {void}`

Sends the authentication credentials provided by the client to the NICE DCV server.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentials</td>
<td>Object</td>
<td>The object containing the supplied credentials. The credentials must have the same name and be of the same type that is specified in the challenge.</td>
</tr>
</tbody>
</table>

Returns:
Type

`void`
Resource Class

The Resource Class can fetch or discard the corresponding file that was just printed or downloaded. When performing these actions, the corresponding observer functions \texttt{filePrinted} (p. 32) and \texttt{fileDownload} (p. 31) would respectively be invoked with the resource object as their only argument. Such resource can be accepted or declined in order to fetch or discard the file they reference.

Exposes
- Methods (p. 13)

Methods

List
- \texttt{accept(urlParameters)} $\rightarrow \{\text{void}\}$ (p. 61)
- \texttt{decline()} $\rightarrow \{\text{void}\}$ (p. 61)

\texttt{accept(urlParameters)} $\rightarrow \{\text{void}\}$

Locally downloads the resource.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>urlParameters</td>
<td>Object</td>
<td>The optional object containing the key/value pairs of the URL search parameters passed to the request to fetch the resource.</td>
</tr>
</tbody>
</table>

Returns:

Type

void

\texttt{decline()} $\rightarrow \{\text{void}\}$

Discards the resource.

Returns:

Type

void

NICE DCV Web UI SDK

A JavaScript React component library, currently exporting a single React component called \texttt{DCVViewer} which connects to the NICE DCV Server and renders the toolbar to interact with the remote stream.
Exposes

• Components (p. 62)

Components

List

• DCVViewer (p. 62)

DCVViewer

The React component rendering the toolbar with all of its functionalities useful to interact with the remote stream.

Properties:

List

• dcv (p. 62)
• uiConfig (p. 64)

dcv

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcv</td>
<td>Object</td>
<td>Yes</td>
<td>The object defining the properties necessary to establish the connection to the NICE DCV Server, setting the log level and the URL from where to load the NICE DCV Web Client SDK assets and access the DCV resources.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>String</td>
<td>Yes</td>
<td>The NICE DCV session ID.</td>
</tr>
<tr>
<td>authToken</td>
<td>String</td>
<td>Yes</td>
<td>The authentication token to use when connecting to the server.</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Required</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------</td>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>serverUrl</td>
<td>String</td>
<td>Yes</td>
<td>The host name and port of the running NICE DCV server in the following format: https://dcv_host_address:port. For example: <a href="https://my-dcv-server:8443">https://my-dcv-server:8443</a>.</td>
</tr>
<tr>
<td>baseUrl</td>
<td>String</td>
<td>Yes</td>
<td>The absolute or relative URL from which to load SDK files.</td>
</tr>
<tr>
<td>resourceBaseUrl</td>
<td>String</td>
<td>No</td>
<td>The absolute or relative URL from which to access DCV resources.</td>
</tr>
<tr>
<td>onDisconnect</td>
<td>function</td>
<td>No</td>
<td>The callback function invoked when disconnecting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(default: () =&gt; {})</td>
</tr>
</tbody>
</table>
### Components

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>from the NICE DCV server, and the connection is closed.</td>
</tr>
<tr>
<td>logLevel</td>
<td>LogLevel (p. 37)</td>
<td>No (default: LogLevel.INFO)</td>
<td>The log level to use in the viewer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**uiConfig**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uiConfig</td>
<td>Object</td>
<td>No (default: {})</td>
<td>The object defining the properties to configure whether the toolbar is visible and whether to display the fullscreen and multimonitor buttons on it.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>toolbar</td>
<td>Object</td>
<td>No (default: {})</td>
<td>The object defining the configuration options for the toolbar.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visible</td>
<td>Boolean</td>
<td>No</td>
<td>The option to define whether</td>
</tr>
</tbody>
</table>

64
### Components

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullscreenButton</td>
<td>Boolean</td>
<td>No</td>
<td>(default: true) Define whether to show or hide the fullscreen button on the toolbar.</td>
</tr>
<tr>
<td>multimonitorButton</td>
<td>Boolean</td>
<td>No</td>
<td>(default: true) Define whether to show or hide the multimonitor button on the toolbar.</td>
</tr>
</tbody>
</table>
Release Notes and Document History for NICE DCV Web Client SDK

This page provides the release notes and document history for NICE DCV Web Client SDK.

Topics
- NICE DCV Web Client SDK Release Notes (p. 66)
- Document History (p. 69)

NICE DCV Web Client SDK Release Notes

This section provides release notes for the NICE DCV Web Client SDK by release date.

Topics
- 1.2.1 — July 21, 2022 (p. 66)
- 1.2.0 — June 29, 2022 (p. 66)
- 1.1.3 — May 23, 2022 (p. 67)
- 1.1.2 — May 19, 2022 (p. 67)
- 1.1.1 — March 23, 2022 (p. 67)
- 1.1.0 — February 23, 2022 (p. 67)
- 1.0.4 — December 20, 2021 (p. 68)
- 1.0.3 — September 01, 2021 (p. 68)
- 1.0.2 — July 30, 2021 (p. 69)
- 1.0.1 — May 31, 2021 (p. 69)
- 1.0.0 — March 24, 2021 (p. 69)

1.2.1 — July 21, 2022

<table>
<thead>
<tr>
<th>Version</th>
<th>Release notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantic version: 1.2.1</td>
<td>Changes and bug fixes</td>
</tr>
<tr>
<td>Build: 358</td>
<td>• Fixed a problem that resulted in a failure to connect to NICE DCV server 2019.1 and older.</td>
</tr>
</tbody>
</table>

1.2.0 — June 29, 2022

<table>
<thead>
<tr>
<th>Version</th>
<th>Release notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantic version: 1.2.0</td>
<td>Changes and bug fixes</td>
</tr>
<tr>
<td>Build: 352</td>
<td>• Fixed crashing bug when the frames received are larger than the maximum supported resolution (4096x2160).</td>
</tr>
<tr>
<td>Version</td>
<td>Release notes</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 1.1.3 — May 23, 2022 | • Resource objects (passed as arguments to fileDownload and filePrinted observers) now have the accept and decline methods that can be called on the object to download and discard the resource respectively.  
• Minor bug fix on automatic clipboard synchronization when disconnecting. |
| 1.1.2 — May 19, 2022 | • Semantic version: 1.1.2  
• Build: 322  

• Changes and bug fixes  
  • Fixed a problem preventing successful connection when specifying the web-url-path option.  
  • Fixed a problem that could cause input to not work correctly after connection.  
  • Fixed mouse coordinates when scale ratio is greater than 1. |
| 1.1.1 — March 23, 2022 | • Semantic version: 1.1.1  
• Build: 309  

• Changes and bug fixes  
  • Report Transport Error when communication with the server times out.  
  • Fixed a recurring decoding error when streaming large resolutions. |
| 1.1.0 — February 23, 2022 | • Semantic version: 1.1.0  

• New features |
## 1.0.4 — December 20, 2021

<table>
<thead>
<tr>
<th>Version</th>
<th>Release notes</th>
</tr>
</thead>
</table>
| • Build: 295 | • Release NICE DCV Web UI SDK library with DCVViewer React component.  
• Export NICE DCV Web Client SDK both as UMD and ES modules.  
• Added high color accuracy support.  
• Added the ability to list and interact with clients connected to a session. Added notifications for connection and disconnection. |

### Changes and bug fixes

• Improved webcodecs decoding support.  
• Various keyboard improvements.  
• Fix a bug that was preventing to open a second screen when the clipboard was disabled.

## 1.0.3 — September 01, 2021

<table>
<thead>
<tr>
<th>Version</th>
<th>Release notes</th>
</tr>
</thead>
</table>
| • Semantic version: 1.0.3  
• Build: 202 | New features  
• Experimental support for WebCodecs. This is disabled by default and must be enabled via the ConnectionConfig object using the new property enableWebCodecs.  
• Clipboard: added support for image/png data type on Chromium based browsers.  
• Added observer/callback to get the server’s screenshot as a PNG image (requires NICE DCV server 2021.2). |

### Changes and bug fixes

• Improved handling of keyboard modifiers.
1.0.2 — July 30, 2021

<table>
<thead>
<tr>
<th>Version</th>
<th>Release notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Semantic version: 1.0.2</td>
<td></td>
</tr>
<tr>
<td>• Build: 167</td>
<td></td>
</tr>
<tr>
<td>• Fixed pressure detection for stylus events.</td>
<td></td>
</tr>
<tr>
<td>• Improved support for Korean keyboard layout on Chrome.</td>
<td></td>
</tr>
</tbody>
</table>

1.0.1 — May 31, 2021

<table>
<thead>
<tr>
<th>Version</th>
<th>Release notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Semantic version: 1.0.1</td>
<td></td>
</tr>
<tr>
<td>• Build: 141</td>
<td></td>
</tr>
<tr>
<td>• Fixed propagation of connection errors and close reasons</td>
<td></td>
</tr>
<tr>
<td>• Fixed filestorage chunk progress update</td>
<td></td>
</tr>
<tr>
<td>• Improved webcam handling</td>
<td></td>
</tr>
<tr>
<td>• Improved audio-in processing</td>
<td></td>
</tr>
</tbody>
</table>

1.0.0 — March 24, 2021

<table>
<thead>
<tr>
<th>Version</th>
<th>Release notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Semantic version: 1.0.0</td>
<td></td>
</tr>
<tr>
<td>• Build: 81</td>
<td></td>
</tr>
<tr>
<td>Initial release of the NICE DCV Web Client SDK.</td>
<td></td>
</tr>
</tbody>
</table>

Document History

The following table describes the documentation for this release of NICE DCV Web Client SDK.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>NICE DCV Web Client SDK version 1.2.0</td>
<td>NICE DCV Web Client SDK 1.2.0 is now available. For more information, see SDK v.1.2.0 (p. 66).</td>
<td>June 29, 2022</td>
</tr>
<tr>
<td>NICE DCV Web Client SDK version 1.1.0</td>
<td>NICE DCV Web Client SDK 1.1.0 is now available. For more information, see SDK v.1.1.0 (p. 67).</td>
<td>February 23, 2022</td>
</tr>
<tr>
<td>NICE DCV Web Client SDK version 1.0.1</td>
<td>Fixed some typos. Minor bugs fixed, see SDK v.1.0.1 (p. 69).</td>
<td>May 31, 2021</td>
</tr>
<tr>
<td>Initial release</td>
<td>First publication of this content.</td>
<td>March 24, 2021</td>
</tr>
</tbody>
</table>