NICE DCV

Web Client SDK Developer Guide

NICE DCV: Web Client SDK Developer Guide

Table of Contents

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

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)

- Touchscreen
- Stylus (on Linux, Windows 10, and Windows Server 2019 servers)
- Multiple monitor support

For more information about these features, see Supported features in the *NICE DCV User Guide*.

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:

Browser	**Version**
Google Chrome	Latest three major versions
Mozilla Firefox	Latest three major versions
Microsoft Edge	Latest three major versions
Apple Safari for macOS	Latest three major versions

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-1.1.3-329.zip.sign nice-dcv-web-client-
   sdk-1.1.3-329.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

modified to also include the content of the corresponding file from NICE DCV Web Client SDK package and possibly any other license information from the libraries used by the consuming user application.

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

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:

```
<!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:

```
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:

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

Define the variables to use to store the Authentication object, Connection object, and the NICE DCV server URL.

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

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

```
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.

```
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.

```
function onPromptCredentials(auth, challenge) {
  // Let's check if in challege 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.

```
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.

```
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";
  form = document.createElement("form");
  fieldSet = document.createElement("fieldset");

  form.onsubmit = submitCredentials;
  form.appendChild(fieldSet);
  form.appendChild(submitButton);

  document.body.appendChild(form);
}

function addInput (name) {
  var type = name === "password" ? "password" : "text";

  var inputField = document.createElement("input");
  inputField.name = name;
  inputField.id = name;
  inputField.placeholder = name;
  inputField.type = type;
  fieldSet.appendChild(inputField);
}

function onPromptCredentials (_, credentialsChallenge) {
  createLoginForm();
  credentialsChallenge.requiredCredentials.forEach(challenge => addInput(challenge.name));
}
```

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 `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.

```
// 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 `DCVViewer` React component from the NICE DCV Web UI SDK.

Topics

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:

```
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:

```
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:

```
<!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.

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

Render the top level container node of your app.

```
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.

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

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.

```
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();
  }
  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(credentials).map((cred) => (
            <input
              key={cred}
              name={cred}
              placeholder={cred}
              type={cred === "password" ? "password" : "text"}
              onChange={updateCredentials}
              value={credentials[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

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. 13)
- 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:

Name	Type	Description
`url`	string	The host name and port of the running NICE DCV server in the following format: `https://dcv_host_address:port`. For example: `https://my-dcv-server:8443`.

Name	Type	Description
`callbacks`	authenticationCallbacks (p. 19)	The callbacks that are available to be called during the authentication process.

Returns:

- The Authentication object.

Type

Authentication (p. 60)

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:

Name	Type	Description
`config`	ConnectionConfig (p. 26)	The ConnectionConfig object.

Returns:

- A Connection object, or an error object.

Type

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

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:

Name	Type	Description
`handler`	function	The custom log handler function. The handler function contains level (number), levelName (string), domain (string), and message (string).

Returns:

Type

    void

setLogLevel(level) → {void}

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

Parameters:

Name	Type	Description
`level`	LogLevel (p. 37)	The log level to use.

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) ConnectionError :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:

Name	Type	Description
`major`	integer	The major version number.
`minor`	integer	The minor version number.
`patch`	integer	The patch version number.
`revision`	integer	The revision number.
`extended`	string	The extended string.
`versionStr`	string	A concatenation of the major, minor, patch, and revision numbers in the

Name	Type	Description
		form `major.minor.patch +build.revision`.

(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

The AudioError code enums available in the DCV module

- `SETTING_AUDIO_FAILED`
- `CHANNEL_NOT_AVAILABLE`

Type:

- number

authenticationCallbacks

Authentication callbacks

Type:

- Object

Properties:

Name	Type	Description
`promptCredentials`	authPromptCredentialsCallback (p. 21)	The callback function to be called when the user is challenged for credentials.
`error`	authErrorCallback (p. 20)	The callback function to be called when authentication fails.
`success`	authSuccessCallback (p. 21)	The callback function to be called when authentication succeeds.

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:

Name	Type	Description
`authentication`	Authentication (p. 60)	The Authentication object.
`error`	Object	The error object raised by the authentication process.

Name	Type	Description

Name	Type	Description
`code`	AuthenticationErrorCode (p. 20)	The error code.
`message`	string	The error message.

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:

Name	Type	Description
`authentication`	Authentication (p. 60)	The Authentication object.
`challenge`	Object	The challenge.

Name	Type	Description
`required`	`Array<Object>`	An array of requested credential objects.

Name	Type	Description
`name`	string	The name of the requested credential.
`type`	string	The type of the requested credential.

authSuccessCallback(authentication, authenticationData)

The callback function to be called when authentication succeeds.

Parameters:

Name	Type	Description
`authentication`	Authentication (p. 60)	The Authentication object.
`authenticationData`	Array.<Object>	An array of objects that include NICE DCV session IDs and authentication tokens.

Name	Type	Description
`sessionId`	string	The NICE DCV session ID.
`authToken`	string	The authentication token for the NICE DCV session.

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:

Name	Type	Description
`event`	Object	Information about the clipboard event.

Name	Type	Attribu	Description
`name`	established | copy | paste | dataSizeAlert | autoCopyDone | newDataAvailable | autoPasteDone | remoteError | pasteAvailableData		Always present. The name of the event.
`clipboardData`	Object | string		The data in the clipboard.
`autoCopy`	boolean	<optional>	Indicates whether automatic copying from the session clipboard to the local client clipboard is enabled.
`maxDataSize`	number	<optional>	The maximum amount of data that can be placed

Name	Type	Description			
		Name	Type	Attribu	Description
					in the clipboard.
		`error`	string	<optional>	Error information if applicable.

ClosingReasonErrorCode

The ClosingReasonError 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`

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:

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

Name	Type	Description
`featuresUpdate`	featuresUpdateCallback (p. 31)	The callback function to be called when a feature's status changes.
`clipboardEvent`	clipboardEventCallback (p. 22)	The callback function to be called when a `clipboardEvent` is generated.
`deviceChangeEvent`	deviceChangeEventCallback (p. 29)	The callback function to be called when an `deviceChange` event is triggered.
`screenshot`	screenshotCallback (p. 40)	The callback function to be called when a `screenshot` is available.

ConnectionConfig

The configuration for a NICE DCV connection.

Type:

- Object

Properties:

Name	Type	Description
`url`	string	The host name and port of the running NICE DCV server in the following format: `https://dcv_host_address:port`. For example: `https://my-dcv-server:8443`.
`sessionId`	string	The NICE DCV session ID.
`authToken`	string	The authentication token to use when connecting to the server.
`baseUrl`	string	The absolute or relative URL from which to load SDK files.
`resourceBaseUrl`	string	The absolute or relative URL from which to access DCV resources.
`enabledChannels`	Array.<Channel (p. 22)>	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.

Name	Type	Description
`losslessColorspace`	Colorspace (p. 24)	Indicates the colorspace that will be used. If not specified, it defaults to "RGB".
`divId`	string	The ID of the `div` object in the HTML DOM where SDK should create the canvas with the remote stream.
`volumeLevel`	integer	The preferred volume level. The valid range is 0 to 100.
`clipboardAutoSync`	boolean	Indicates whether automatic copying from the NICE DCV session clipboard to the local client clipboard is enabled for compatible web browsers.
`dynamicAudioTuning`	boolean	Indicates whether to dynamically tune the audio based on the NICE DCV server audio settings when a connection is established.
`clientHiDpiScaling`	boolean	Indicates whether to scale the canvas based on the client's DPI.
`highColorAccuracy`	boolean	Indicates whether high color accuracy should be used if available. If not specified, it defaults to `false`.
`enableWebCodecs`	Boolean	Indicates whether WebCodecs should be used if available. Defaults to `false` if not specified.
`observers`	connectionCallbacks (p. 24)	The callback functions to call for events that are related to the connection.
`callbacks`	connectionCallbacks (p. 24)	The same as the `observers` property, but each callback includes the Connection (p. 43) object as the first parameter.

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)

Type:

- number

createDirectory(path)

Parameters:

Name	Type	Description
`path`	string	The absolute path on the server where we want to create a directory. It should also include the name of the target directory.

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:

Name	Type	Description
`info`	Object	Information about the data channel.

Name	Type	Description
`name`	string	The name of

Name	Type	Description

Name	Type	Description
		the data channel.
`token`	string	The authentication token for the data channel.

deleteFile(path)

Parameters:

Name	Type	Description
`path`	string	The absolute path on the server identifying the file we want to delete.

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:

Name	Type	Description
`reason`	Object	The reason for the disconnection.

Name	Type	Description
`code`	number	The reason code.
`message`	string	The reason message.

displayAvailabilityCallback(status, displayId)

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

Parameters:

Name	Type	Description
`status`	Object	The status of the display.

Name	Type	Description
`enabled`	boolean	Indicates if the display is enabled.
`closed`	boolean	Indicates if the display is closed.

Name	Type	Description
`displayId`	number	The identifier for the display.

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:

Name	Type	Description
`serverWidth`	number	The width (in pixels) of the primary display.
`serverHeight`	number	The height (in pixels) of the primary display.
`heads`	Array.<Monitor (p. 37)>	The display heads supported by the NICE DCV server.

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:

Name	Type	Description
`featuresList`	Array.<feature (p. 31)>	An array of features that have changed.

fileDownloadCallback(fileResource)

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

Parameters:

Name	Type	Description
`fileResource`	Object	Information about the file that is ready for download.

Name	Type	Description

Name	Type	Description
`id`	string	The identifier for the file.
`url`	string	The URL to use to download the file.
`domain`	string	The resource domain.
`token`	string	The authentication token to use to download the file. The token is also included in the URL.

filePrintedCallback(printResource)

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

Parameters:

Name	Type	Description
`printResource`	Object	Information about the printed file.

Name	Type	Description
`id`	string	The identifier for the printed file.
`url`	string	The URL to use to download the

Name	Type	Description
		<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>printed file.</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></table>

filestorage

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

Type:

- Object

Properties:

Name	Type	Description
`list`	list (p. 37)	Function that allows to list the items (files and directories) present at the supplied path on the server.
`createDirectory`	createDirectory (p. 28)	Function that allows to create a directory at the specified path on the server.
`retrieveFile`	retrieveFile (p. 40)	Function that allows to locally download a file at the specified path on the server.

Name	Type	Description
`deleteFile`	deleteFile (p. 29)	Function that allows to delete a file at the specified path on the server.
`renameFile`	renameFile (p. 40)	Function that allows to rename a file from the specified source path to the specified destination path.
`renameDirectory`	renameDirectory (p. 39)	Function that allows to rename a directory from the specified source path to the absolute destination path.
`storeFile`	storeFile (p. 42)	Function that allows to upload a local file to the supplied path on the server.

filestorageEnabledCallback(enabled)

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

Parameters:

Name	Type	Description
`enabled`	boolean	Indicates whether file storage is enabled.

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:

Name	Type	Description
`resizeEnabled`	boolean	Indicates whether the server supports resizing the client display layout.
`relativeMouseModeEnabled`	boolean	Indicates whether the server supports relative mouse mode.
`displayId`	number	The identifier for the display.

idleWarningNotificationCallback(disconnectionDateTime)

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

Parameters:

Name	Type	Description
`disconnectionDateTime`	Date	The date and time of the disconnection.

collaboratorListCallback(collaborators)

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

Parameters:

Name	Type	Description
`collaborators`	Array.<Object>	A list of objects containing information on collaborators.

Name	Type	Description
`username`	string	The username of the collaborator.
`owner`	boolean	Indicates whether the collaborator

Name	Type	Description

Name	Type	Description
		is the session owner.
connectionId	number	Indicates the ID assigned by the server to the connection.

licenseNotificationCallback(notification)

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

Parameters:

Name	Type	Description
notification	Object	The notification.

Name	Type	Description
product	string	The DCV product.
status	string	The status of the license.
message	string	A message.
leftDays	number	The number of days before the license expires.
isDemo	boolean	Indicates if the license is a demo license.
numUnlicensed	number	The number of

Name	Type	Description

Name	Type	Description
		unlicensed connections.
`licensingMode`	string	The licensing mode.
`documentationUrl`	string	The URL for the documentation.

list(path)

Parameters:

Name	Type	Description
`path`	string	The absolute path on the server of which we want to list the content.

LogLevel

The available SDK log levels.

Type:

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

Monitor

Type:

- Object

Properties:

Name	Type	Description
`name`	string	The name of the display head.
`rect`	Object	Information about the display head.

Name	Type	Description
`x`	number	The initial `x`

Name	Type	Description
		<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>coordinate for the display head.</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></table>
`primary`	boolean	Indicates whether the display head is the primary display head. This is determined from the remote operating system if available.
`dpi`	number	The DPI of the display head.

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:

Name	Type	Description
`state`	Array.<Object>	Information about the connection quality.

Name	Type	Description
`name`	string	The name of the indicator.
`status`	NORMAL | WARNING | CRITICAL	Description of the status.
`changed`	boolean	Indicates whether the status changed.

renameDirectory(src, dest)

Parameters:

Name	Type	Description
`src`	string	The absolute source path on the server identifying the directory we want to rename.
`dest`	string	The absolute destination path on the server specifying the target path and directory name.

renameFile(src, dest)

Parameters:

Name	Type	Description
`src`	string	The absolute source path on the server identifying the file we want to rename.
`dest`	string	The absolute destination path on the server specifying the target path and file name.

ResolutionErrorCode

The ResolutionError code enums available in the DCV module

- `INVALID_ARGUMENT`
- `NO_CHANNEL`

Type:

- number

retrieveFile(path)

Parameters:

Name	Type	Description
`path`	string	The absolute path on the server identifying the file we want to download locally.

screenshotCallback(screenshot)

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

Parameters:

Name	Type	Description
`screenshot`	byte[]	Screenshot buffer in PNG format, or `null` if screenshot retrieval failed.

ScreenshotErrorCode

The ScreenshotError code enums available in the DCV module

- `NO_CHANNEL`
- `GENERIC_ERROR`

Type:

- number

serverInfo

Type:

- Object

Properties:

Name	Type	Description
`name`	string	The name of the software.
`version`	Object	The software version number.

Name	Type	Description
`major`	number	The major version number.
`minor`	number	The minor version number.
`revision`	number	The revision version number.

Name	Type	Description
`os`	string	The OS.
`arch`	string	The architecture.
`hostname`	string	The hostname.

stats

Type:

- Object

Properties:

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

storeFile(file, dir)

Parameters:

Name	Type	Description
`file`	File	The file object (for more information see https://developer.mozilla.org/en-US/docs/Web/API/File) we want to upload to the server.
`dir`	string	The absolute path on the server where we want to upload the file.

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)
- 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) → {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. 55)
- 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. 58)
- 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:

Name	Type	Description
`win`	Object	The window to which the display must be attached.
`displayConf`	Object	The configuration of the display.

Name	Type	Attribu	Description
`displayId`	number	<optional>	The ID of the display.
`displayDivName`			The name of the display div.

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:

Name	Type	Attributes	Description
`enabled`	boolean		To start listening to events, specify `true`. To stop listening to events, specify `false`.
`win`	Object	<optional>	The window in which to listen for events. If omitted, the default window is used.
`displayId`	number	<optional>	The ID of the display that should listen the events. If omitted, the default display of the window is used.

Returns:

Type

void

detachDisplay(displayId) → {void}

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

Parameters:

Name	Type	Description
`displayId`	number	The ID of the display to detach.

Returns:

Type

void

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:

Name	Type	Description
`connectionId`	boolean	The id of the connection that will be disconnected.

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:

Name	Type	Description
`enable`	boolean	To enable display quality updates, specify `true`. To disable display quality updates, specify `false`.

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:

Name	Type	Description
`key`	Control | Alt | AltGraph | Meta | OS | Shift	The key to send.
`location`	KeyboardEvent.location	The key's location. For more information, see https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/location.
`isDown`	boolean	If the key event to inject is a keydown (`true`) or a keyup (`false`).

Returns:

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

Type

boolean

openChannel(name, authToken, callbacks) → {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:

Name	Type	Description
`name`	string	The name of the channel.

Name	Type	Description
`authToken`	string	The authentication token to use to connect to the channel.
`callbacks`	Object	The onMessage and onClose callbacks functions to call.

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:

Name	Type	Description
`featureName`	feature (p. 31)	The name of the feature to query.

Returns:

Promise. If resolved, the function returns a `status` object that always containes 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:

Name	Type	Description
`shortcuts`	Array.<Object>	The array of keys and mappings to register.

Name	Type	Description

Name	Type	Description
`sequence`	Array.<Object>	The keyboard shortcut to register.

Na	Ty	Description
`key`	KeyboardEvent.key	The value of the key pressed by the user. For more information, see https://developer.mozilla en-US/docs/Web/API/KeyboardEvent/key.
`loc`	KeyboardEvent.locat	The array of keys to send. The location of the key on the keyboard. For more information, see https://developer.mozilla en-

Name	Type	Description

Name	Type	Description
output	Array.<Object>	The intended action to be performed by the shortcut.

Na	Ty	Description
		US/docs/Web/API/KeyboardEvent/location.

Na	Ty	Description
key	KeyboardEvent.key	The value of the key pressed by the user. For more information, see https://developer.mozilla en-US/docs/Web/API/KeyboardEvent/key.
loca	KeyboardEvent.locat	The array of keys to send. The location of the key

Name	Type	Description				
		Name	Type	Description		
				Na	Ty	Description
						on the keyboard. For more information, see https://developer.mozilla en-US/docs/Web/API/KeyboardEvent/location.

Returns:

Type

void

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

Requests 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:

Name	Type	Description
`highColorAccuracy`	boolean	Whether or not high color accuracy should be requested.

Returns:

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

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:

Name	Type	Description
`layout`	Array.<Monitor (p. 37)>	The requested displays in the layout.

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:

Name	Type	Description
`width`	number	The width to request in pixels. The minimum allowed value is `0`.
`height`	number	The height to request in pixels. The minimum allowed value is `0`.

Returns:

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

Type

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

sendKeyboardEvent(event) → {boolean}

Sends a keyboard shortcut event. For more information about keyboard events, see https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent. Valid Keyboard events include: `keydown`, `keypress`, and `keyup`. For more information about these events, see https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent#events.

Parameters:

Name	Type	Description
`event`	KeyboardEvent	The keyboard event to send.

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:

Name	Type	Description
`shortcut`	Array.<Object>	The array of keys to send.

Name	Type	Description
`key`	KeyboardEvent.key	The value of the key pressed by the user. For more information, see https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/key.
`location`	KeyboardEvent.location	The array of keys to send. The location of the key

Name	Type	Description
		<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>on the keyboard. For more information, see https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/location.</td></tr></table>

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:

Name	Type	Attributes	Description
`min`	number		The minimum image quality.
`max`	number	<optional>	The maximum image quality.

Returns:

Type

void

setDisplayScale(scaleRatio, displayId) → {Promise\|Promise.<{code: ResolutionErrorCode (p. 40), message: string}>}

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:

Name	Type	Description
`scaleRatio`	float	The scaling ratio to use. Must be a strictly positive number.
`displayId`	number	The ID of the display to scale.

Returns:

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

Type

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

setKeyboardQuirks(quirks) → {void}

Sets keyboard quirks for the client computer.

Parameters:

Name	Type	Description
`quirks`	Object	The keyboard quirks to enable or disable.

Name	Type	Description
`macOptionToAlt`	boolean	To map the Option key to Alt for macOS, specify `true`. Otherwise, specify `false`.
`macCommandToControl`	boolean	To map the Command key to Ctrl for macOS, specify `true`. Otherwise, specify `false`.

Returns:

Type

void

setMaxDisplayResolution(maxWidth, maxHeight) → {void}

Sets the maximum display resolution to use for the connection.

Parameters:

Name	Type	Description
`maxWidth`	number	The maximum display width in pixels. The minimum allowed value is `0`.
`maxHeight`	number	The maximum display height in pixels. The minimum allowed value is `0`.

Returns:

Type

void

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

Enables or disables the microphone.

Parameters:

Name	Type	Description
`enable`	boolean	To enable the microphone, specify `true`. To disable the microphone, specify `false`.

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:

Name	Type	Description
`minWidth`	number	The minimum display width in pixels. The minimum allowed value is `0`.
`minHeight`	number	The minimum display height in pixels. The minimum allowed value is `0`.

Returns:

Type

void

setUploadBandwidth(value) → {number}

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

Parameters:

Name	Type	Description
`value`	number	The maximum bandwidth limit in kbps. Valid range is 1024 kbps to 102400 kbps.

Returns:

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

Type

number

setVolume(volume) → {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:

Name	Type	Description
`volume`	number	The volume level to use.

Returns:

Type

void

setWebcam(enable, deviceId) → {Promise|Promise.<{code: WebcamErrorCode (p. 42), message: string}>}

Enables or disables the webcam.

Parameters:

Name	Type	Description
`enable`	boolean	To enable the webcam, specify `true`. To disable the webcam, specify `false`.
`deviceId`	string	The device ID of the webcam.

Returns:

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

Type

Promise | Promise.<{code: WebcamErrorCode (p. 42), message: string}>

syncClipboards() → {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 sycnhronized, or if the browser does not support autocopy, the function returns `false`.

Type

boolean

Authentication Class

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:

Name	Type	Description
`credentials`	Object	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.

Returns:

Type

void

NICE DCV Web UI SDK

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

Exposes

- Components (p. 61)

Components

List

- DCVViewer (p. 61)

DCVViewer

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

Properties:

List

- dcv (p. 61)
- uiConfig (p. 63)

dcv

Name	Type	Required	Description
`dcv`	Object	Yes	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.

Nam	Type	Requ	Description
`sessionId`	String	Yes	The NICE DCV session ID.
`authToken`	String	Yes	The authentication token to use when connecting to the server.

Name	Type	Required	Description

Name	Type	Required	Description
`serverUrl`	String	Yes	The host name and port of the running NICE DCV server in the following format: https:// dcv_host_address:p For example: https:// my-dcv-server:8443.
`baseUrl`	String	Yes	The absolute or relative URL from which to load SDK files.
`resourceBaseUrl`	String	No (default `""`)	The absolute or relative URL from which to access DCV resources.
`onDisconnect`	function	No (default `() => {}`)	The callback function invoked when disconnecting

Name	Type	Required	Description

Name	Type	Required	Description
			from the NICE DCV server, and the connection is closed.
`logLevel`	LogLevel (p. 37)	No (default: `LogLevel.INFO`)	The log level to use in the viewer.

uiConfig

Name	Type	Required	Description
`uiConfig`	Object	No (default: `{}`)	The object defining the properties to configure whether the toolbar is visible and whether to display the fullscreen and multimonitor buttons on it.

Name	Type	Required	Description
`toolbar`	Object	No (default: `{}`)	The Object defining the configuration options for the toolbar.

Name	Type	Required	Description
`visible`	Boolean	No (default: `true`)	To define whether

Name	Type	Required	Description

Name	Type	Required	Description

Name	Type	Required	Description
			to show or hide the toolbar.
`fullscreenButt`	Boolean (default: `true`)	Not	The option to define whether to show or hide the fullscreen button on the toolbar.
`multimonitorBu`	Boolean (default: `true`)	Not	The option to define whether to show or hide the multimonitor button on the toolbar.

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

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

Topics

1.1.3 — May 23, 2022

Version	Release notes
• Semantic version: 1.1.3 • Build: 329	Changes and bug fixes • Fixed a problem preventing successful connection when specifying the web-url-path option.

1.1.2 — May 19, 2022

Version	Release notes
• Semantic version: 1.1.2 • Build: 322	Changes and bug fixes • Fixed a problem that could cause input to not work correctly after connection.

Version	Release notes
	• Fixed mouse coordinates when scale ratio is greater than 1.

1.1.1 — March 23, 2022

Version	Release notes
• 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

Version	Release notes
• Semantic version: 1.1.0 • Build: 295	New features • 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.

1.0.4 — December 20, 2021

Version	Release notes
• Semantic version: 1.0.4 • Build: 249	New features • Support opening multiple connections from the same page. • Support loading the SDK from a CDN.

1.0.3 — September 01, 2021

Version	Release notes
• 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

Version	Release notes
• Semantic version: 1.0.2 • Build: 167	• Fixed pressure detection for stylus events. • Improved support for Korean keyboard layout on Chrome.

1.0.1 — May 31, 2021

Version	Release notes
• Semantic version: 1.0.1 • Build: 141	• Fixed propagation of connection errors and close reasons • Fixed filestorage chunk progress update • Improved webcam handling • Improved audio-in processing

1.0.0 — March 24, 2021

Version	Release notes
• Semantic version: 1.0.0 • Build: 81	Initial release of the NICE DCV Web Client SDK.

Document History

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

Change	Description	Date
NICE DCV Web Client SDK version 1.1.0	NICE DCV Web Client SDK 1.1.0 is now available. For more information, see SDK v.1.1.0 (p. 66).	February 23, 2022
NICE DCV Web Client SDK version 1.0.1	Fixed some typos. Minor bugs fixed, see SDK v.1.0.1 (p. 67).	May 31, 2021
Initial release	First publication of this content.	March 24, 2021