aws

Web Client SDK Developer Guide

Amazon DCV

Amazon DCV: Web Client SDK Developer Guide

Table of Contents

What is the Amazon DCV Web Client SDK?

> **Note**
>
> Amazon DCV was previously known as NICE DCV.

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

Using the Amazon 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 Amazon DCV Web Client SDK to build your custom web browser client applications to interact with Amazon DCV sessions within your workflows.

Topics

Prerequisites

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

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

Supported features

You can build custom web browser client applications that support the following Amazon DCV features:

- Connect to Windows Amazon DCV servers
- Connect to Linux Amazon 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 *Amazon DCV User Guide*.

Browser support

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

The Amazon 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

Browser	Version
Apple Safari for macOS	Latest three major versions

Versioning convention

The Amazon 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 Amazon DCV Web Client SDK

The Amazon 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 Amazon DCV website .

To get started with the Amazon DCV Web Client SDK

1. The Amazon 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 **Amazon DCV Web Client SDK archive** and the **Amazon DCV Web Client SDK archive signature** from the Amazon DCV website .
3. Verify the signature of the Amazon 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-1.8.7-858.zip.sign nice-dcv-web-client-
   sdk-1.8.7-858.zip
   ```

4. If the signature verifies successfully, extract the contents of the Amazon 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 Amazon DCV Web Client SDK on your web server.
> - When using Amazon 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 Amazon DCV Web Client SDK package and possibly any other license information from the libraries used by the consuming user application.

Connect to a Amazon 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 Amazon DCV server, and how to receive the first frame of streamed content from the Amazon 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 Amazon DCV Web Client SDK to draw the content stream from the remote Amazon 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 Amazon DCV server, and how to receive the first frame of content from the Amazon DCV server.

First, from the `index.js` file import the Amazon 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 Amazon DCV server URL.

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

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

```
console.log("Using Amazon 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 Amazon DCV server prompts for credentials, the `promptCredentials` callback function receives the requested credential challenge from the Amazon DCV server. If the Amazon DCV server is configured to use system authentication, then the sign-in credentials must be provided. The following code samples assume that the username is `my_dcv_user` and that the password is `my_password`.

If authentication fails, the `error` callback function receives an error object from the Amazon 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 Amazon DCV server. The following code sample calls the connect function and connects to the first session returned in the array.

> **Note**
>
> In the following code example, replace `MY_DCV_USER` with your own username and `MY_PASSWORD` with your own password.

```
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 Amazon DCV server. The `firstFrame` callback method is called when the first frame is received from the Amazon 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 Amazon 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 Amazon DCV features

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

The features that are available in a Amazon 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 Amazon 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 Amazon DCV Authorization in the *Amazon DCV Administrator Guide* .

Understanding the featuresUpdate callback function

When the availability of a feature in a Amazon DCV session changes, the Amazon 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 Amazon DCV Web UI SDK

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

Topics

Prerequisites

You need to install `React`, `ReactDOM`, `Cloudscape Design Components React`, `Cloudscape Design Global Styles` and `Cloudscape Design Design Tokens`.

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

You would also need to download `Amazon DCV Web Client SDK`. See Getting started with the Amazon DCV Web Client SDK 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 Amazon 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 Amazon 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 Amazon DCV Web Client SDK as a ESM module, the `DCVViewer` React component from the Amazon DCV Web UI SDK, `React` and the `Cloudscape Design Global Styles` package.

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

Following is an example showing how to authenticate against the Amazon DCV Server and render the `DCVViewer` React component from Amazon 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 Amazon DCV server prompts for credentials, the `promptCredentials` callback function receives the requested credential challenge from the Amazon DCV server. If the Amazon 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 Amazon 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 Amazon 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 Amazon DCV Web UI SDK reference.

To know more about self-signed certificates, see the Redirection clarifications with self-signed certificates.

Updating from AWS-UI to Cloudscape Design System

Starting SDK version 1.3.0 we updated our `DCVViewer` component from AWS-UI to its evolution: Cloudscape Design.

Cloudscape uses a different visual theme than AWS-UI, but the underlying code base remains the same. Thus, migrating your application based on the `DCVViewer` should be easy. To migrate,

replace the AWS-UI related NPM packages you've installed with the associated Cloudscape packages:

AWS-UI package name	Cloudscape package name
@awsui/components-react	@cloudscape-design/components
@awsui/global-styles	@cloudscape-design/global-styles
@awsui/collection-hooks	@cloudscape-design/collection-hooks
@awsui/design-tokens	@cloudscape-design/design-tokens

For further details on the migration please refear to the AWS-UI GitHub documentation page.

SDK reference

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

Topics

DCV module

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

Exposes

Methods

List

authenticate(url, callbacks) → {Authentication}

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

Parameters:

Name	Type	Description
`url`	string	The host name and port of the running Amazon DCV server in the following format: `https://dcv_host_address:port` . For example: `https://my-dcv-server:8443` .
`callbacks`	authenticationCallbacks	The callbacks that are available to be called during the authentication process.

Returns:

- The Authentication object.

Type

Authentication

connect(config) → {Promise.<Connection>|Promise.<{code: ConnectionErrorCode, message: string}>}

Connects to the specified Amazon 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	The ConnectionConfig object.

Returns:

- A Connection object, or an error object.

Type

Promise.<Connection> | Promise.<{code: ConnectionErrorCode, 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	The log level to use.

Returns:

Type

void

Members

List

- (constant) AudioError :AudioErrorCode
- (constant) AuthenticationError :AuthenticationErrorCode
- (constant) ChannelError :ChannelErrorCode
- (constant) ClosingReasonError :ClosingReasonErrorCode
- (constant) ConnectionError :ConnectionErrorCode
- (constant) CustomChannelError :CustomChannelErrorCode
- (constant) DisplayConfigError :DisplayConfigErrorCode
- (constant) FileStorageError :FileStorageErrorCode
- (constant) LogLevel :LogLevel
- (constant) MultiMonitorError :MultiMonitorErrorCode
- (constant) ResolutionError :ResolutionErrorCode
- (constant) TimezoneRedirectionError :TimezoneRedirectionErrorCode
- (constant) TimezoneRedirectionSetting :TimezoneRedirectionSettingCode
- (constant) TimezoneRedirectionStatus :TimezoneRedirectionStatusCode
- (constant) version
- (constant) ScreenshotError :ScreenshotErrorCode
- (constant) WebcamError :WebcamErrorCode

(constant) AudioError :AudioErrorCode

The AudioError codes enum.

Type:

- AudioErrorCode

(constant) AuthenticationError :AuthenticationErrorCode

The AuthenticationError codes enum.

Type:

- AuthenticationErrorCode

(constant) ChannelError :ChannelErrorCode

The ChannelError codes enum.

Type:

- ChannelErrorCode

(constant) ClosingReasonError :ClosingReasonErrorCode

The ClosingReasonError codes enum.

Type:

- ClosingReasonErrorCode

(constant) ConnectionError :ConnectionErrorCode

The ConnectionError codes enum.

Type:

- ConnectionErrorCode

(constant) CustomChannelError :CustomChannelErrorCode

The CustomChannelError codes enum.

Type:

- CustomChannelErrorCode

(constant) DisplayConfigError :DisplayConfigErrorCode

The DisplayConfigError codes enum.

Type:

- DisplayConfigErrorCode

(constant) FileStorageError :FileStorageErrorCode

The FileStorageError codes enum.

Type:

- FileStorageErrorCode

(constant) LogLevel :LogLevel

The available SDK log levels.

Type:

- LogLevel

(constant) MultiMonitorError :MultiMonitorErrorCode

The MultiMonitorError codes enum.

Type:

- MultiMonitorErrorCode

(constant) ResolutionError :ResolutionErrorCode

The ResolutionError codes enum.

Type:

- ResolutionErrorCode

(constant) TimezoneRedirectionError :TimezoneRedirectionErrorCode

The TimezoneRedirectionError codes enum.

Type:

- TimezoneRedirectionErrorCode

(constant) TimezoneRedirectionSetting :TimezoneRedirectionSettingCode

The TimezoneRedirectionSetting codes enum.

Type:

- TimezoneRedirectionSettingCode

(constant) TimezoneRedirectionStatus :TimezoneRedirectionStatusCode

The TimezoneRedirectionStatus codes enum.

Type:

- TimezoneRedirectionStatusCode

(constant) version

The Amazon 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.

Name	Type	Description
`extended`	string	The extended string.
`versionStr`	string	A concatenation of the major, minor, patch, and revision numbers in the form `major.minor.patch+build.revision` .

(constant) ScreenshotError :ScreenshotErrorCode

The ScreenshotError codes enum.

Type:

- ScreenshotErrorCode

(constant) WebcamError :WebcamErrorCode

The WebcamError codes enum.

Type:

- WebcamErrorCode

Type and callback definitions

List

- AudioErrorCode
- authenticationCallbacks
- AuthenticationErrorCode
- authErrorCallback(authentication, error)
- authPromptCredentialsCallback(authentication, challenge)
- authSuccessCallback(authentication, authenticationData)
- Channel

- LogLevel
- Monitor
- MultiMonitorErrorCode
- qualityIndicatorStateCallback(state)
- renameDirectory(src, dest)
- renameFile(src, dest)
- ResolutionErrorCode
- retrieveFile(path)
- screenshotCallback(screenshot)
- ScreenshotErrorCode
- serverInfo
- stats
- storeFile(file, dir)
- TimezoneRedirectionErrorCode
- TimezoneRedirectionSettingCode
- TimezoneRedirectionStatusCode
- WebcamErrorCode

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	The callback function to be called when the user is challenged for credentials.
`error`	authErrorCallback	The callback function to be called when authentication fails.
`success`	authSuccessCallback	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	The Authentication object.
`error`	Object	The error object raised by the authentication process. **Name** / **Type** / **Description** `code` / AuthenticationErrorCode / 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	The Authentication object.
`challenge`	Object	The challenge. **Name** / **Type** / **Description** `requiredCredentials` / Array.<Object> / An array of requested credential objects. **N** / **T** / **Description** `name` / String / The name of the requested credential. `type` / String / The type of the requested

Name	Type	Description
		Name: Type: Description: N T Description: credential.

authSuccessCallback(authentication, authenticationData)

The callback function to be called when authentication succeeds.

Parameters:

Name	Type	Description
`authentication`	Authentication	The Authentication object.
`authenticationData`	Array.<Object>	An array of objects that include Amazon DCV session IDs and authentication tokens. Name: Type: Description: `sessior` string The Amazon DCV session ID. `authTok` string The authentic

Name	Type	Description
		Name: Type: Description: ation token for the Amazon 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** / **Attribs** / **Description** `name` — established | copy | paste | dataSilert | autoCone | newDailable | autoPaDone | remotor | pasteAlableD — Always present. The name of the event. `clipbData` — Object | string — The data in the

Name	Type	Description
		Name: Type: Attribs: Description: clipboard. autoC: boolea: <optic>: Indicates whether automatic copying from the session clipboard to the local client clipboard is enabled. maxDa ze: numb: <optic>: The maximum amount of data that can be placed in the clipboard.

Name	Type	Description
		Name: error Type: string Attribs: <optic> Description: 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
- EVICTED
- EXTERNAL_PROTOCOL_CONNECTION_EVICTED
- DISCONNECTION_REQUESTED

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	The callback function to be called when the connection ends.
`displayLayout`	displayLayoutCallback	The callback function to be called when the display layout or resolution is changed.
`displayAvailability`	displayAvailabilityCallback	The callback function to be called when a display's availability changes.
`firstFrame`	firstFrameCallback	The callback function to be called when the first frame is received from the Amazon DCV server.

Name	Type	Description
`filePrinted`	filePrintedCallback	The callback function to be called when a file is printed on the Amazon DCV server.
`fileDownload`	fileDownloadCallback	The callback function to be called when a file is ready to be downloaded from the Amazon DCV server.
`dataChannel`	dataChannelCallback	The callback function to be called when the Amazon DCV server sends a notification about the availability of a data channel.
`licenseNotification`	licenseNotificationCallback	The callback function to be called when the Amazon DCV server sends a notification about the license state.
`idleWarningNotification`	idleWarningNotificationCallback	The callback function to be called when the Amazon DCV server sends an idle timeout warning.
`collaboratorList`	collaboratorListCallback	The callback function to be called when the Amazon DCV server sends the list of collaborators (since Amazon DCV Web Client SDK version 1.1.0).
`qualityIndicatorState`	qualityIndicatorStateCallback	The callback function to be called when the connection quality indicator changes state.

Name	Type	Description
`filestorageEnabled`	filestorageEnabledCallback	The callback function to be called when file storage is enabled or disabled.
`featuresUpdate`	featuresUpdateCallback	The callback function to be called when a feature's status changes.
`clipboardEvent`	clipboardEventCallback	The callback function to be called when a `clipboardEvent` is generated.
`deviceChangeEvent`	deviceChangeEventCallback	The callback function to be called when an `deviceChange` event is triggered.
`screenshot`	screenshotCallback	The callback function to be called when a `screenshot` is available.

ConnectionConfig

The configuration for a Amazon DCV connection.

Type:

- Object

Properties:

Name	Type	Description
`url`	string	The host name and port of the running Amazon DCV server in the following format: `https://d`

Name	Type	Description
		`cv_host_address:port` . For example: `https://my-dcv-server:8443` .
`sessionId`	string	The Amazon 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>	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.
`losslessColorspace`	Colorspace	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.

Name	Type	Description
`clipboardAutoSync`	boolean	Indicates whether automatic copying from the Amazon 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 Amazon 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	The callback functions to call for events that are related to the connection.
`callbacks`	connectionCallbacks	The same as the `observers` property, but each callback includes the Connection 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)
- `SERVER_UNREACHABLE` (since DCV Server 2022.1)
- `GATEWAY_BUSY`
- `UNSUPPORTED_CREDENTIAL` (since DCV Server 2022.2)
- `TRANSPORT_ERROR`

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 Amazon 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 the data channel. `token` / string / The authentication token for the

Name	Type	Description
		Name: Type: Description: 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
		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

Name	Type	Description
		Name / Type / Description: is closed.
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>	The display heads supported by the Amazon 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 Amazon 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 Amazon DCV server to the client.
- `clipboard-paste` - Indicates the availability of clipboard paste functionality from the client to the Amazon 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 Amazon DCV server to the client.
- `file-upload` - Indicates availability of file upload functionality from the client to the Amazon DCV server.
- `timezone-redirection` - Indicates the availability of timezone redirection functionality (since Amazon DCV Web Client SDK version 1.3.0).

Type:

- string

featuresUpdateCallback(featuresList)

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

Parameters:

Name	Type	Description
`featuresList`	Array.<feature>	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 Amazon DCV server.

Parameters:

Name	Type	Description
`fileResource`	Object	Information about the file that is ready for download. **Name** / **Type** / **Description** `id` / string / The identifier for the file. `url` / string / The URL to use to download the file.

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

filePrintedCallback(printResource)

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

Parameters:

Name	Type	Description
printResource	Object	Information about the printed file.

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

Name	Type	Description
		Name: — Type: — Description: The token is also included in the URL.

filestorage

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

Type:

- Object

Properties:

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

Name	Type	Description
`deleteFile`	deleteFile	Function that allows to delete a file at the specified path on the server.
`renameFile`	renameFile	Function that allows to rename a file from the specified source path to the specified destination path.
`renameDirectory`	renameDirectory	Function that allows to rename a directory from the specified source path to the absolute destination path.
`storeFile`	storeFile	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 Amazon 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 Amazon 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 Amazon 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
`usernam`	string	The username of the collaborator.
`owner`	boolean	Indicates whether the collaborator is the session owner.

Name	Type	Description
		Name / **Type** / **Description** `connectionId` / number / Indicates the ID assigned by the server to the connection.

licenseNotificationCallback(notification)

The callback function to be called when the Amazon 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.

Name	Type	Description

Name	Type	Description
message	string	A message.
leftDay	number	The number of days before the license expires.
isDemo	boolean	Indicates if the license is a demo license.
numUnlicensed	number	The number of unlicensed connections.
licensingMode	string	The licensing mode.
documentationUrl	string	The URL for the

Name	Type	Description
		<table><tr><th>Name</th><th>Type</th><th>Description</th></tr><tr><td></td><td></td><td>documentation.</td></tr></table>

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.

Name	Type	Description
`rect`	Object	Information about the display head. **Name** / **Type** / **Description** `x` / number / The initial x coordinate for the display head. `y` / number / The initial y coordinate for the display head. `width` / number / The width (in pixels) of the display head. `height` / number / The height (in

Name	Type	Description
		Name / Type / Description: pixels) of the display head.
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 | WARNIN(| CRITICAL / Description of the status. `changec` / 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`
- `NOT_IMPLEMENTED`

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. revisic / number / The revision version number.
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://d

Name	Type	Description
		eveloper.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.

TimezoneRedirectionErrorCode

The TimezoneRedirectionError code enums available in the DCV module

- `INVALID_ARGUMENT`
- `NO_CHANNEL`
- `USER_CANNOT_CHANGE`

Type:

- number

TimezoneRedirectionSettingCode

The TimezoneRedirectionSetting code enums available in the DCV module

- `ALWAYS_OFF`
- `ALWAYS_ON`
- `CLIENT_DECIDES`

Type:

- number

TimezoneRedirectionStatusCode

The TimezoneRedirectionStatus code enums available in the DCV module

- SUCCESS
- PERMISSION_ERROR
- GENERIC_ERROR

Type:

- number

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 of the dcv module. For an example showing how to use it, see the Getting started section.

Exposes

- Methods

Methods

List

- attachDisplay(win, displayConf) → {Promise.<number>|Promise.<{code: MultiMonitorErrorCode, message: string}>}
- captureClipboardEvents(enabled, win, displayId) → {void}
- detachDisplay(displayId) → {void}
- disconnect() → {void}

- disconnectCollaborator(connectionId) → {void}
- enableDisplayQualityUpdates(enable) → {void}
- enableHighPixelDensity(enable) → {void}
- enableTimezoneRedirection(enable) → {Promise|Promise.<{code: TimezoneRedirectionErrorCode, message: string}>}
- enterRelativeMouseMode() → {void}
- getConnectedDevices() → {Promise.<Array.<MediaDeviceInfo>>|Promise.<{message: string}>}
- getFileExplorer() → {Promise.<filestorage>|Promise.<{code: ChannelErrorCode, message: string}>}
- getServerInfo() → {serverInfo}
- getScreenshot() → {Promise|Promise.<{code: ScreenshotErrorCode, message: string}>}
- getStats() → {stats}
- latchModifierKey(key, location, isDown) → {boolean}
- openChannel(name, authToken, callbacks, namespace) → {Promise|Promise.<{code: ChannelErrorCode, message: string}>}
- queryFeature(featureName) → {Promise.<{enabled: boolean, remote?: string, autoCopy?: boolean, autoPaste?: boolean, serviceStatus?: string, available?: boolean}>|Promise.<{message: string}>}
- registerKeyboardShortcuts(shortcuts) → {void}
- requestDisplayConfig(highColorAccuracy) → {Promise|Promise.<{code: DisplayConfigErrorCode, message: string}>}
- requestDisplayLayout(layout) → {Promise|Promise.<{code: ResolutionErrorCode, message: string}>}
- requestResolution(width, height) → {Promise|Promise.<{code: ResolutionErrorCode, message: string}>}
- sendKeyboardEvent(event) → {boolean}
- sendKeyboardShortcut(shortcut) → {void}
- setDisplayQuality(min, maxopt) → {void}
- setDisplayScale(scaleRatio, displayId) → {Promise|Promise.<{code: ResolutionErrorCode, message: string}>} (DEPRECATED)
- setKeyboardQuirks(quirks) → {void}
- setMaxDisplayResolution(maxWidth, maxHeight) → {void}

- setMicrophone(enable) → {Promise|Promise.<{code: AudioErrorCode, message: string}>}
- setMinDisplayResolution(minWidth, minHeight) → {void}
- setUploadBandwidth(value) → {number}
- setVolume(volume) → {void}
- setMicrophone(enable, deviceId) → {Promise|Promise.<{code: AudioErrorCode, message: string}>}
- setWebcam(enable, deviceId) → {Promise|Promise.<{code: WebcamErrorCode, message: string}>}
- syncClipboards() → {boolean}

attachDisplay(win, displayConf) → {Promise.<number>|Promise.<{code: MultiMonitorErrorCode, 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** / **Attribs** / **Description** `displ` / numb / <optic> / The ID of the display. `displ` `vName` / / / The name of

Name	Type	Description
		Name: — Type: — Attribs: — Description: the display div.

Returns:

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

Type

Promise.<number> | Promise.<{code: MultiMonitorErrorCode, 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.

Name	Type	Attributes	Description
`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 Amazon DCV server and closes the connection.

Returns:

Type

void

disconnectCollaborator(connectionId) → {void}

Requests disconnect of collaborator connected with the provided connection id (since Amazon 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

enableHighPixelDensity(enable) → {void}

Enables or disables high pixel density on the client.

Parameters:

Name	Type	Description
`enable`	boolean	Whether or not high pixel density should be enabled.

Returns:

Type

void

enableTimezoneRedirection(enable) → {Promise|Promise.<{code: TimezoneRedirectionErrorCode, message: string}>}

Enables or disables timezone redirection. Once it is enabled, the client requests the server to make the server desktop timezone match the client timezone.

Parameters:

Name	Type	Description
`enable`	boolean	To enable timezone redirecti on, specify `true`. To disable timezone redirection, specify `false`.

Returns:

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

Type

Promise.<number> | Promise.<{code: TimezoneRedirectionErrorCode, message: string}>

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>|Promise.<{code: ChannelErrorCode, message: string}>}

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

Returns:

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

Type

Promise.<filestorage> | Promise.<{code: ChannelErrorCode, message: string}>

getServerInfo() → {serverInfo}

Gets information about the Amazon DCV server.

Returns:

Information about the server software.

Type

serverInfo

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

Retrieves the screenshot of the remote desktop in PNG format. The screenshot will be returned in the screenshotCallback 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, message: string}>

getStats() → {stats}

Gets statistics about the Amazon DCV server.

Returns:

Information about the streaming statistics.

Type

stats

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, namespace) → {Promise| Promise.<{code: ChannelErrorCode, message: string}>}

Opens a custom data channel on the connection if it was created on the Amazon 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.
`namespace`	string	The namespace of the channel. Available since Amazon DCV Web Client SDK 1.2.0 and Amazon DCV Server 2022.1.

Returns:

Promise. If rejected we receive an error object.

Type

Promise | Promise.<{code: ChannelErrorCode, 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 Amazon DCV server feature.

Parameters:

Name	Type	Description
`featureName`	feature	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** `sequenc` / Array.<Object> / The keyboard shortcut to register. **N** / **T** / **Description** k / K v / The value of the key

Name	Type	Description
		Name: Type: Description: N T Description: pressed by the user. For more information, see https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/key. l K v t This array of keys to send. The

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>N</th><th>T</th><th>Description</th></tr>
<tr><td></td><td></td><td>location of the key on the keyboard. For more information, see https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/location.</td></tr>
</table>
</td></tr>
<tr><td><code>output</code></td><td>Array.<Object></td><td>The intended action</td></tr>
</table>
</td></tr>
</table>

Name	Type	Description
		Name: Type: Description: to be performed by the shortcut. Name: k Type: K v Description: The value of the key pressed by the user. For more information, see https://developer.mozilla.org/en-US/docs/Web/API/

Name	Type	Description
		Name: Type: Description:
		N T Description: Keyboard Event/key.
		l K v t This array of keys to send. The location of the key on the keyboard. For more information, see https://developer.mozilla.org/en-US/

Name	Type	Description
		Name: Type: Description: docs/Web/API/KeyboardEvent/location.

Returns:

Type

void

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

Requests an updated display config from the Amazon DCV Server. Available since Amazon DCV Web Client SDK 1.1.0 and Amazon 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, message: string}>

requestDisplayLayout(layout) → {Promise|Promise.<{code: ResolutionErrorCode, message: string}>}

Requests an updated display layout for the connection.

Parameters:

Name	Type	Description
`layout`	Array.<Monitor>	The requested displays in the layout.

Returns:

Promise. If rejected we receive an error object.

Type

Promise | Promise.<{code: ResolutionErrorCode, message: string}>

requestResolution(width, height) → {Promise|Promise.<{code: ResolutionErrorCode, message: string}>}

Requests an updated display resolution from the Amazon DCV server.

Parameters:

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

Name	Type	Description
`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, 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:

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>shortcut</td><td>Array.<Object></td><td>The array of keys to send.
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>key</td><td>KeyboardEvent.key</td><td>The value of the key pressed by the user. For more information, see https://developer.mozilla.org/en-US/docs/Web/API/Keybo</td></tr>
</table>
</td></tr>
</table>

Name	Type	Description
		Name / **Type** / **Description** ardEvent/key. `locatic` / Keyboarc vent.loca tion / The array of keys to 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:

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, message: string}>} (DEPRECATED)

Deprecated since version 1.3.0. There is no need to set the display scale anymore. Mouse coordinates will be managed automatically internally.

Notifies the Amazon 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, 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

Name	Type	Description
		Name / **Type** / **Description** macOS, specify `true`. Otherwise, specify `false`. `macCommandToCtrl` / 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, 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, 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 Amazon 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

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

[Experimental - might change in the future] Enables or disables the microphone.

Parameters:

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

Name	Type	Description
`deviceId`	string	The device ID of the microphone. If no deviceId is provided, the `default` deviceId is used.

Returns:

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

Type

Promise | Promise.<{code: AudioErrorCode, message: string}>

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

syncClipboards() → {boolean}

Synchronizes the local client clipboard with the remote Amazon 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 of the `dcv` module. For an example showing how to use it, see the Getting started section.

Exposes

- Methods

Methods

List

- retry() → {void}
- sendCredentials(credentials) → {void}

retry() → {void}

Retries the authentication process.

Returns:

Type

void

sendCredentials(credentials) → {void}

Sends the authentication credentials provided by the client to the Amazon 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

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 `filePrinted` and `fileDownload` 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

Methods

List

- accept(urlParameters) → {void}
- decline() → {void}

accept(urlParameters) → {void}

Locally downloads the resource.

Parameters:

Name	Type	Description
`urlParameters`	Object	The optional object containing the key/value pairs of the URL search parameters passed to the request to fetch the resource.

Returns:

Type

void

decline() → {void}

Discards the resource.

Returns:

Type

void

Amazon DCV Web UI SDK

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

Exposes

Components

List

DCVViewer

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

Properties:

List

- dcv
- uiConfig

dcv

Name	Type	Required	Description
dcv	Object	Yes	The object defining the properties necessary to establish the connection to the Amazon DCV Server, setting the log level and the URL from where to load the Amazon DCV Web Client SDK assets and access the DCV resources.

Name	Type	Required	Description
			<table><tr><th>Nam</th><th>Typ</th><th>Req</th><th>Descripti on</th></tr><tr><td>ses</td><td>Strir</td><td>Yes</td><td>The Amazon DCV session ID.</td></tr><tr><td>aut</td><td>Strir</td><td>Yes</td><td>The authentic ation token to use when connectin g to the server.</td></tr><tr><td>ser</td><td>Strir</td><td>Yes</td><td>The host name and port of the running Amazon DCV server in the</td></tr></table>

Name	Type	Required	Description
			Nam Typ Req Description following format: https://dcv_host_address:port. For example: https://my-dcv-server:8443. bas Strir Yes The absolute or relative URL from which to load SDK files. res ase Strir No (def "") The absolute or relative URL

Name	Type	Required	Description
			Nam **Type** **Req** **Description**
			from which to access DCV resources.
			`onDisconnect` func No (def `()=>{}`) The callback function invoked when disconnecting from the Amazon DCV server, and the connection is closed.
			`log` Logl No (def `LogINF` The log level to use in

Name	Type	Required	Description
			Nam Typ Req Descripti on 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 fullscree n and multimonitor buttons on it. Nam Typ Req Descripti on `too` Obj No (def {}) The Object defining the configura tion options for the toolbar.

Name	Type	Required	Description
			Nam Typ Req Descripti on
			Boolean (default: true) define whether to show or hide the toolbar.
			Boolean (default: true) define whether to show or hide the fullscreen button on the toolbar.

Name	Type	Required	Description
			Nam Typ Req Descripti on Description (optional, default to true) define whether to show or hide the multimonitor button on the toolbar.

Release Notes and Document History for Amazon DCV Web Client SDK

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

Topics

Amazon DCV Web Client SDK Release Notes

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

Topics

1.8.7 — October 31, 2024

Build numbers	Changes and bug fixes	
• Semantic version: 1.8.7 • Build: 858	• Fixed rendering on Firefox 130 and newer	

1.8.4 — October 1, 2024

Build numbers	New features	Changes and bug fixes
• Semantic version: 1.8.4 • Build: 840	The following features were added: • Renamed to “Amazon DCV Web Client SDK” • Added new API enableHighPixelDensity for high dpi displays • Added an experimental API setMicrophone to select the microphone in compatible browsers • Added new Connection Errors GATEWAY_BUSY, UNSUPPORTED_CREDENTIAL, and TRANSPORT_ERROR •	• Improved Webcam handling • Improved audio playback handling • Improved WebCodecs handling • Improved plug and unplug of microphone and webcam • Improved remote window dragging when multimonitor •

Build numbers	New features	Changes and bug fixes
	Added new Closing Reasons EXTERNAL_PROTOCOL_CONNECTION_EVICTED, and DISCONNECTION_REQUESTED	File storage upload and download permissions are now correctly propagated • Minor fixes on rendering

1.5.10 — December 19, 2023

Version	Release notes
• Semantic version: 1.5.10 • Build: 684	Changes and bug fixes • Fix stream decoding errors

1.5.6 — November 9, 2023

Version	Release notes
• Semantic version: 1.5.6 • Build: 659	Changes and bug fixes • Performance improvements in the stream decoding and rendering • Removed support for Internet Explorer 11

1.4.4 — June 29, 2023

Version	Release notes

Version	Release notes
• Semantic version: 1.4.4 • Build: 573	Changes and bug fixes • The viewer UI compoenent now uses the `navigator.keyboard.lock` API on browsers that support it to handle special keys in full screen. • Fixed a problem which could cause wrong colors when using Chrome 114 or newer. • Improved WebCodecs detection. • Fixed a problem with the mouse button state when entering the window. • Fixed a problem which could cause the modifier keys to remain pressed on macOS. • Improved audio robustness to degraded network conditions. • Fixed memory leaks. • Improved logs to include time and level.

1.4.0 — March 28, 2023

Version	Release notes
• Semantic version: 1.4.0 • Build: 476	New features •

Version	Release notes
	Added a new `uploadFiles` method to the `FileStorage` object to upload multiple files. • The viewer UI component now supports drag and drop to initiate file upload. • The WebCodecs browser API is now used also for audio and webcam. Changes and bug fixes • Fixed memory leaks related to repeated connections from the same page. • `setUploadBandwidth` now allows values up to 1 Gbps. • Optimized rendering of UI components. • Fixed support for animated cursors on Windows. • Fixed a problem with clipboard support when both text and image data are present for the same operation. • Improved robustness of the Webcam API: settings cannot be changed while a request is already in progress, `webcam.setEnabled` now keeps track of device ID for the request is in progress and returns a

Version	Release notes
	Promise. The viewer UI component shows notification in case of error.

1.3.1 — December 9, 2022

Version	Release notes
• Semantic version: 1.3.1 • Build: 413	Changes and bug fixes • Fixed a problem which could cause the Time Zone redirection UI to go out of synchroni zation with the server. • Fixed a memory leak after multiple reconnections. • Fixed a problem which caused a blank page on disconnetion. • Fixed a bug causing console warnings on audio decoder close.

1.3.0 — November 11, 2022

Version	Release notes
• Semantic version: 1.3.0 • Build: 407	New features • Adopted Cloudscape (https://cloudscap e.design) for the UI Viewer component. • Added support for Time Zone redirection.

Version	Release notes
	Changes and bug fixes • Fixed missing update on asynchronous clipboard when the DCV viewer is focused. • The `setDisplayScale` function is not needed anymore when scaling the display on client side. • The `DCVViewer` component now automatically calls `disconnect()` when it is unmounted.

1.2.1 — July 21, 2022

Version	Release notes
• Semantic version: 1.2.1 • Build: 358	Changes and bug fixes • Fixed a problem that resulted in a failure to connect to Amazon DCV server 2019.1 and older.

1.2.0 — June 29, 2022

Version	Release notes
• Semantic version: 1.2.0 • Build: 352	Changes and bug fixes •

Version	Release notes
	Fixed crashing bug when the frames received are larger than the maximum supported resolution (4096x2160). • 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.

1.1.3 — 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 •	Changes and bug fixes •

Version	Release notes
Build: 322	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

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 Amazon DCV Web UI SDK library with `DCVViewer` React component. • Export Amazon DCV Web Client SDK both as UMD and ES modules. • Added high color accuracy support.

Version	Release notes
	• Added the ability to list and interact with clients connected to a session. Added notifications for connection and disconnec tion. 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

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 Amazon 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

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 Amazon DCV Web Client SDK.

Document History

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

Change	Description	Date
Amazon DCV Web Client SDK version 1.8.4	Amazon DCV Web Client SDK 1.8.4 is now available. For more information, see SDK v.1.8.4.	October 1, 2024
Amazon DCV Web Client SDK version 1.5.6	Amazon DCV Web Client SDK 1.5.6 is now available. For	November 9, 2023

Change	Description	Date
	more information, see SDK v.1.5.6.	
Amazon DCV Web Client SDK version 1.4.4	Amazon DCV Web Client SDK 1.4.4 is now available. For more information, see SDK v.1.4.4.	June 29, 2023
Amazon DCV Web Client SDK version 1.4.0	Amazon DCV Web Client SDK 1.4.0 is now available. For more information, see SDK v.1.4.0.	March 28, 2023
Amazon DCV Web Client SDK version 1.3.1	Amazon DCV Web Client SDK 1.3.1 is now available. For more information, see SDK v.1.3.1.	December 9, 2022
Amazon DCV Web Client SDK version 1.3.0	Amazon DCV Web Client SDK 1.3.0 is now available. For more information, see SDK v.1.3.0.	November 11, 2022
Amazon DCV Web Client SDK version 1.2.0	Amazon DCV Web Client SDK 1.2.0 is now available. For more information, see SDK v.1.2.0.	June 29, 2022
Amazon DCV Web Client SDK version 1.1.0	Amazon DCV Web Client SDK 1.1.0 is now available. For more information, see SDK v.1.1.0.	February 23, 2022
Amazon DCV Web Client SDK version 1.0.1	Fixed some typos. Minor bugs fixed, see SDK v.1.0.1.	May 31, 2021

Change	Description	Date
Initial release	First publication of this content.	March 24, 2021