

# Teradici Software Development Kit for Mac

The PCoIP Client Software Development Kit (SDK) for Mac is a set of libraries and binaries that enables developers to build custom PCoIP clients. The SDK is provided as part of Teradici Cloud Access Software and the resulting client can connect to Cloud Access Software Standard Edition and Graphics Edition through the Cloud Access and Cloud Access Plus plans. The SDK includes the following components:

- Code Examples
- Programming Guide
- Session Client Source Code
- Session Client Binary Executable
- Session Client Libraries and API Headers
- Session Client API Documentation
- Broker Client Example Code
- Broker Client Libraries and API Headers
- PCoIP Core Libraries and API Headers
- PCoIP Core Binary

## Supported Platforms

The PCoIP Client SDK can be used on the following **Mac** operating systems:

- macOS 10.14 Mojave
- macOS 10.13 High Sierra

## What Can You Build With the PCoIP Client SDK?

The PCoIP Client SDK provides developers the ability to embed a PCoIP session into any program or solution, or create a standalone client with a completely custom user interface and workflow. With complete control over how a PCoIP client is built, you can create clients that incorporate

customizations in both **pre-session** and **in-session** phases of a PCoIP connection. For example, the following customizations are all typical use cases:

### Pre-Session Customizations

- Customizing the client user interface to create a branded, end-to-end solution using company assets such as:
  - Corporate Logos
  - Slogans, trademarks or other text
  - Corporate colors and iconography
- Developing customized authentication workflows, either directly or using a broker.
- Automatically connecting users to specific desktops or applications, based on an identified user type or task.
- Embedding a remote workload into an application.

### In-Session Customizations

- Client branding, including:
  - Menu item labels
  - Window titles
  - Application icons
  - Company logos
- Automatic bridging of USB devices.
- In-session menu bar visibility.
- Disabling hot keys.
- Client display size in windowed and fullscreen mode.
- Configuring resolutions.
- Securely using local and bridged USB devices.

Complete control of the in-session user experience is possible using the PCoIP Core API.

# Who Should Read This Guide?

This guide provides information for software developers and systems integrators with an understanding of SDK integrations and virtualization systems who are developing customized PCoIP clients. Readers should already understand the Teradici Cloud Access Software and how it is used in virtual environments, in both brokered and nonbrokered sessions. This guide will break down how to use the session client executable and how to use the broker client API, session client API and core API. This document is not intended for users who are unfamiliar with SDK integrations, or for Teradici Cloud Access Software users who do not require a customized PCoIP client. In this guide, you will learn about:

- PCoIP Session Components and Considerations
- Customizing the PCoIP Session
- Setting up a PCoIP Agent Test Environment
- Setting up a Development Environment for Windows
- Troubleshooting Issues related to Setting up, Developing and Building the SDK

## Understanding terms and conventions in Teradici guides

For more information on the industry specific terms, abbreviations, text conventions, and graphic symbols used in this guide, see [Using Teradici Product and Component Guides](#) and the [Teradici Glossary](#).

## Session API and Core API Development

If you are looking for development specific instructions and information around using the Session API, see [Session API Development](#).

If you are looking for development specific instructions and information around using the Core Libraries and API, see [Core API Development](#).

# What's New in This Release?

This release introduces the following features and enhancements to the PCoIP Software Client SDK for macOS:

## **Detect Monitors Feature**

**Detect Monitors** is a new menu item that enables users to trigger the client to recognize if there has been a change in the displays connected to the Client. For example, if you are plugging a projector into a laptop. This feature extends the remote workstation to include the projector without having to disconnect or reconnect to the session.

# System Requirements

The following table outlines the system requirements for the PCoIP Client SDK for Mac:

System	Version Required
PCoIP Client SDK Operating Systems	<ul style="list-style-type: none"> <li>• macOS Catalina (10.15)*</li> <li>• macOS Mojave (10.14)</li> </ul>
Compatible PCoIP agents	All PCoIP agents and versions
Compatible Teradici Remote Workstation Cards	TERA22x0 with firmware 5.0.1+
Compatible PCoIP Host Software	<ul style="list-style-type: none"> <li>• Host Software for Windows: 4.3.1+</li> <li>• Host Software for Linux: 4.7.0+</li> </ul>

## \*macOS Catalina Support

There are some steps that need to be carried out to enable macOS Catalina support and functionality is properly configured on the PCoIP Client SDK for macOS 19.11:

- You need to sign the libraries and notarize your custom client with Apple before the application can run on macOS Catalina.
- You need to sign all libraries, as well as the `SessionClient` executable and package and notarize these before your custom application, using the `SessionClient` executable, can run on macOS Catalina.

## Remote Workstation Platforms

PCoIP Host Software must be installed on Remote Workstation machines to enable keyboard and mouse functionality.

## Notarizing Applications to run on macOS Catalina

The PCoIP Client SDK for macOS 19.11 has been compiled with OSX SDK 10.13 and 10.14 which meets the Apple notarization requirements. For information on notarizing applications built with the SDK, see [https://developer.apple.com/documentation/security/notarizing\\_your\\_app\\_before\\_distribution](https://developer.apple.com/documentation/security/notarizing_your_app_before_distribution).

## About Teradici and Third-Party Software Licenses

The Teradici Client SDK is a Software Development Kit for Windows, macOS, and Linux that allows developers to build custom PCoIP clients. The licensee can build a PCoIP client using the kit for the purposes of connecting to Teradici Cloud Access Software agents.

Teradici Client SDK code which is copyrighted by Teradici is licensed by the terms listed [here](#).

The Teradici Client SDK contains open-source and third-party components. The licenses for these Third-party license terms by operating system:

- [Windows open-source and third-party licenses](#)
- [Linux open-source and third-party licenses](#)
- [macOS open-source and third-party licenses](#)

PCoIP Clients that are built from the SDK are dynamically linked to QT libraries licensed under the GNU Lesser GPL v3.0 or later, which can be obtained here:

- [Windows QT libraries](#)
- [macOS QT libraries](#)
- Linux clients use a system library; source code is available from the Ubuntu distribution.

The license terms for these QT libraries are [here](#).

If you want to statically link the QT libraries, you can only do so by purchasing a [QT commercial license](#). While the PCoIP Client SDK uses QT, the licensee can select technologies other than QT.

The PCoIP Client executable can run on macOS, Linux, or Windows. Licenses for each OS are obtained from Apple, Microsoft, or via open source licenses for Linux distributions

# Wacom Tablet Support

The Software Client for Mac supports Wacom tablets in a *bridged* configuration, where peripheral data is sent to the desktop for processing.

## Remote USB Device Support

If a pointer type USB device, for example wacom tablets, mouse, or stylus device, is remoted, you need to grant *PCoIPClient.app* computer control to enable it to render and move the cursor. You can enable this through the **Preferences>Security & Privacy>Privacy>Accessibility** location on your system settings.

## Bridged Wacom Tablets

Bridged Wacom tablets are supported only in low-latency environments. Tablets in network environments with greater than 25ms latency will show reduced responsiveness and are not recommended.

The following Wacom tablet models have been tested and are supported on a PCoIP Software Client for Mac.

### PCoIP client support for *bridged* Wacom tablets and the Software Client for Mac

	PCoIP Standard Agent for Linux	PCoIP Graphics Agent for Linux	PCoIP Standard Agent for Windows	PCoIP Graphics Agent for Windows	PCoIP Remote Workstation Card
Wacom Intuos Pro Medium	✓	✓	✓	✓	—
Wacom Intuos Pro Large	✓	✓	✓	✓	—

	PCoIP Standard Agent for Linux	PCoIP Graphics Agent for Linux	PCoIP Standard Agent for Windows	PCoIP Graphics Agent for Windows	PCoIP Remote Workstation Card
Wacom Cintiq 22	✓	✓	✓	✓	—
Wacom Cintiq 16	✓	✓	✓	✓	—
Wacom Cintiq Pro 13	✓*	✓*	✓*	✓*	—
Wacom Cintiq Pro 16	✓*	✓*	✓*	✓*	—
Wacom Cintiq 22 -Pen Only	—	—	—	—	—
Wacom Cintiq Pro 24 Pen and Touch	✓*	✓*	✓*	✓*	—
Wacom Cintiq Pro 32 Pen and Touch	✓	✓	✓	✓	—

\*These Wacom models have not been tested by Teradici.

Other Wacom tablets may work, but have not been tested.



# About PCoIP Sessions

Establishing a PCoIP session involves a number of key components, including system actors, PCoIP session phases, and connection brokers as discussed next.

## System Actors

There are at least three components that work together to create a PCoIP session:

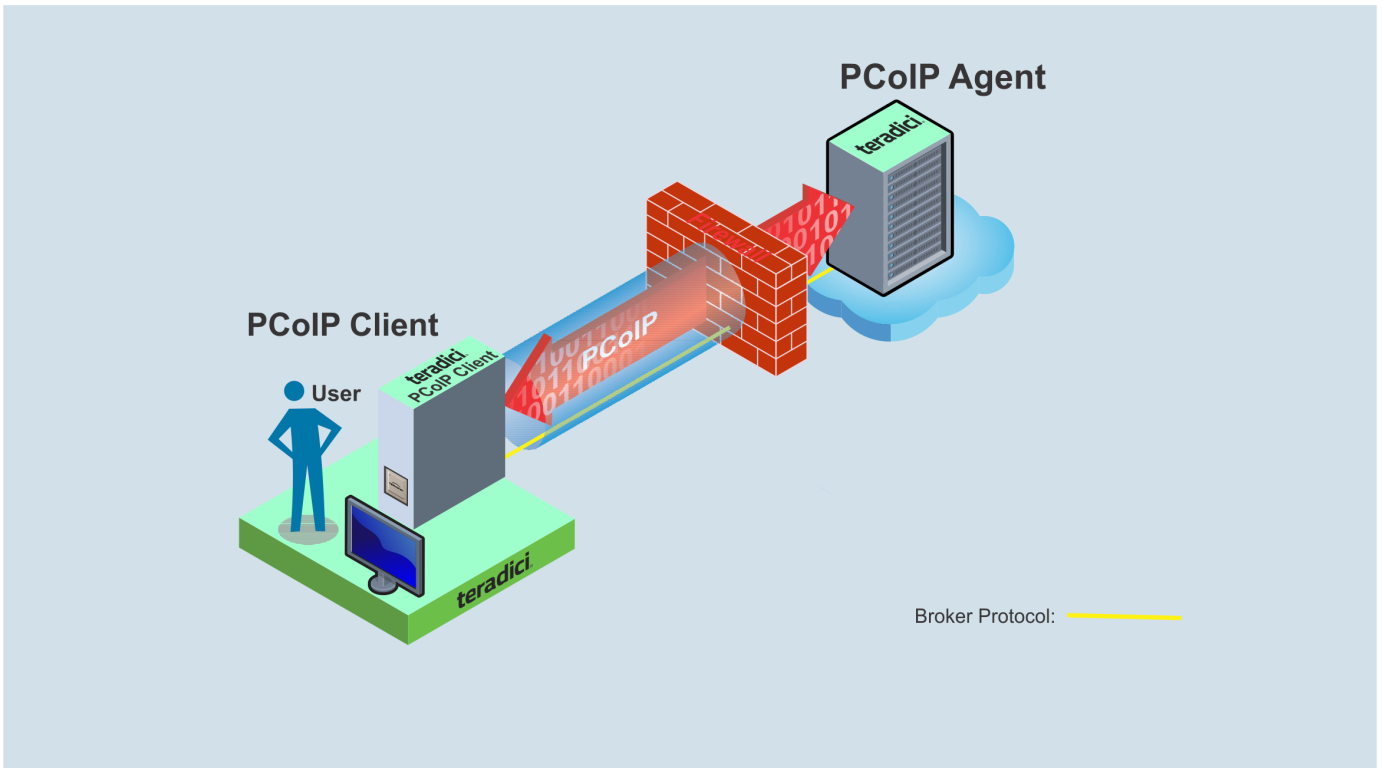
- **PCoIP client:** The hardware or software device, local to the user, which requests and drives the PCoIP session by negotiating with PCoIP brokers and PCoIP agents.
- **PCoIP Broker:** Brokers maintain lists of active users, their authentication information, and the host machines they can connect to. Except for systems using direct connections, all PCoIP sessions are negotiated via third-party brokers.
- **PCoIP agent:** The Teradici extension installed on the host machine. The PCoIP agent is the single point of access for PCoIP clients, and handles all video, audio, and USB exchanges between the client and desktop.

### Terminology: Hosts and Desktops

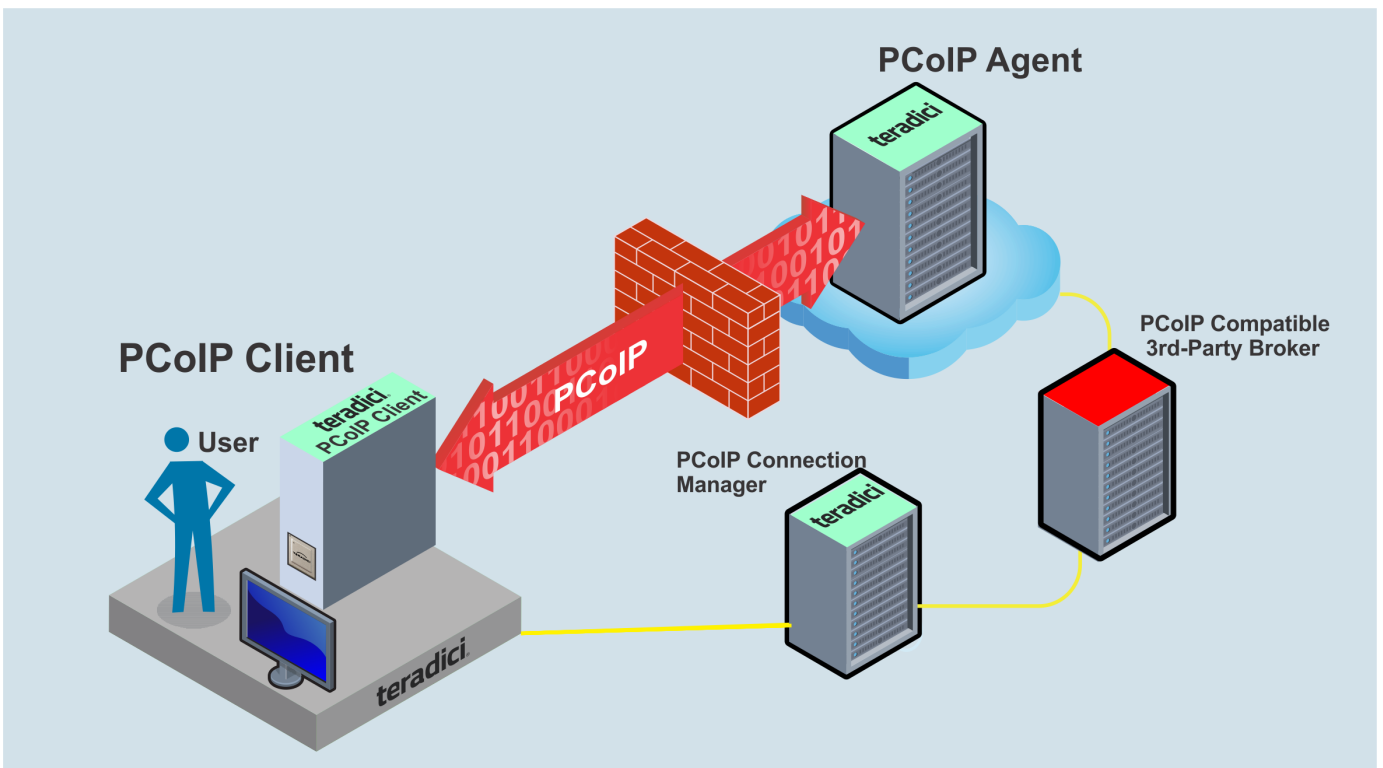
**Host** refers to a Windows, or Linux machine, either virtual or physical, which has a PCoIP agent installed and can serve a remote desktop to a PCoIP Client. **Desktop** refers to an entity which is delivered to the client as a remote workload. This is typically a full Windows or Linux desktop, but it can also be configured to present a single application.

The following diagrams show the actors outlined above in a brokered and direct connection:

### Direct Connection



### Brokered Connection



# PCoIP Session Phases

There are two phases in a PCoIP session:

- **Pre-session** In the pre-session phase, a PCoIP client communicates with a PCoIP broker to authenticate a user and obtain a list of desktops for which that user is authorized. The client then presents this list to the user for selection, and asks the broker to establish a PCoIP session with the selected desktop.
- **Session** In the session phase, the PCoIP session has been successfully launched and the client is connected to the remote desktop. Once the PCoIP connection is established, a session client is invoked by the pre-session client. The session client is primarily a conduit between the host and the client applications. For a list of customizable in-session properties, with examples, see [Customizing the PCoIP Session](#).

# About Brokered and Non-Brokered Connections

PCoIP-compatible brokers are resource managers that authenticate users and dynamically assign authorized host agents to PCoIP clients based on the identity of the user. PCoIP clients can connect to PCoIP agents using a PCoIP-compatible broker, called a **brokered** connection, or directly, called a **non-brokered** or **direct** connection. The broker client library included in the Client SDK is designed to communicate with PCoIP-compatible brokers using the PCoIP Broker Protocol. In direct connections, when no broker is used, the PCoIP agent acts as its own broker. The client makes the same calls to the broker client library in either case.

## Example pre-session Client

The included pre-session client, `broker_client_example`, uses the included broker client library to execute transactions using the PCoIP Broker Protocol. This example client demonstrates how to establish both brokered and non-brokered connections

# Connecting to a USB Device

Remote hosts using the PCoIP Standard Agent or the PCoIP Graphics Agent can use USB devices that are attached to the client. When you connect a local USB device to your remote host it will be disabled on the client machine.

USB device connections do not persist across multiple PCoIP sessions. You must connect your USB device each time you connect.

## **PCoIP Agent needs to be configured to enable USB redirection**

The USB menu will only show up if the PCoIP Agent has been configured to enable USB redirection and a USB device has been detected by the PCoIP Client.

## **One-time PCoIP Client login**

The first time you connect after a Mac installation, you must enter a Mac administrator user name and password. You only need to do this when you install a new client.

## To connect to a USB device:

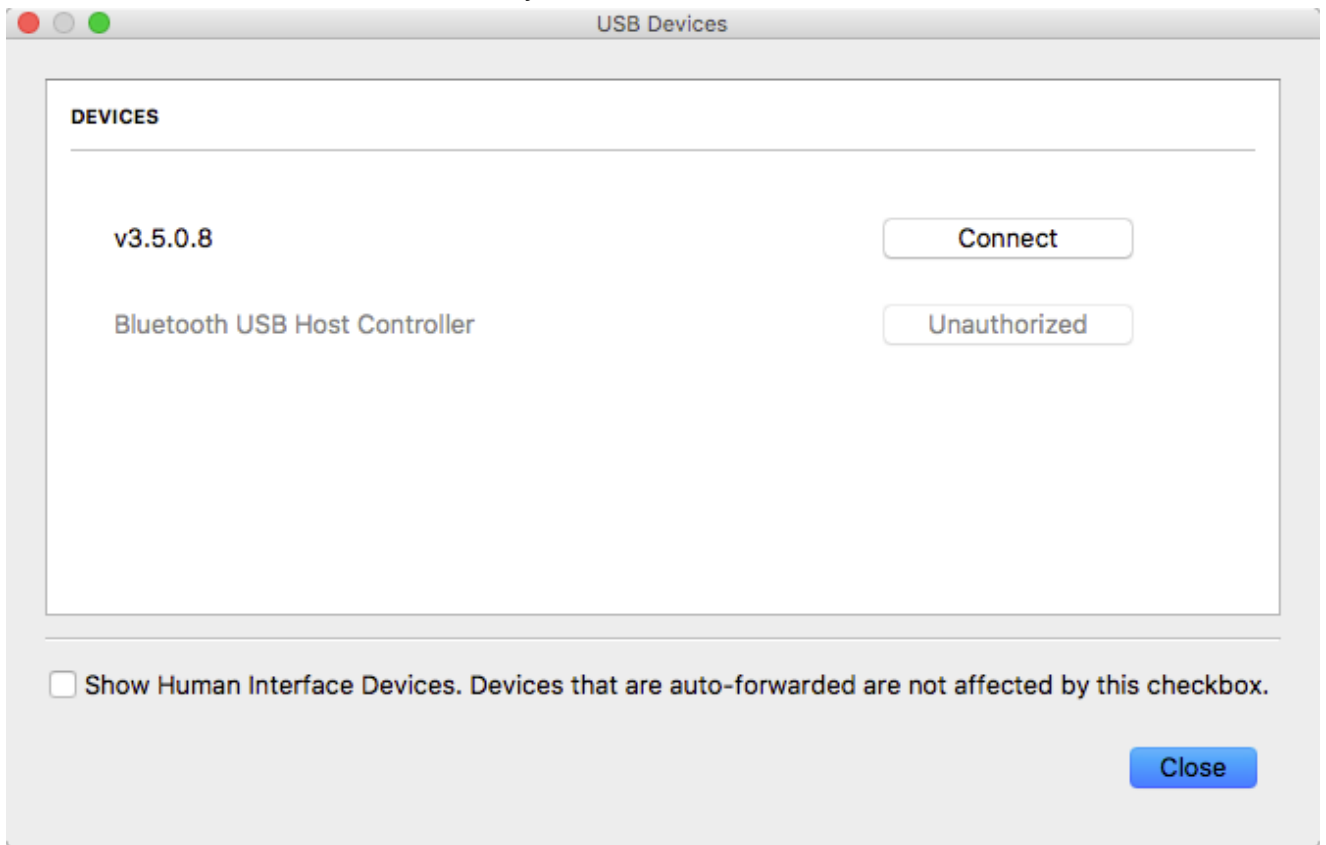
1. Attach the USB device you want to connect.
2. Select **Connection > USB Devices** from the PCoIP Software Client menu.

A list of USB devices connected to your client machine appears. Integrated USB devices, such as built-in cameras on laptops, will appear in this list along with devices you have plugged in yourself.

Some devices will identify themselves only as *USB Device*.

To disconnect a USB device:

3. Click **Connect** beside the USB device you want to use.



### Connecting to Human Interface Devices

Most Human Interface Devices (HIDs), such as keyboards and mice, are automatically handled by the PCoIP Software Client and don't appear on in this list even if they use a USB connection.

If you need to connect a Human Interface Device that can't be locally processed, like a 3D mouse or a Wacom tablet, enable the **Show Human Interface Devices** checkbox to reveal the device in the USB device list and click its Connect button.

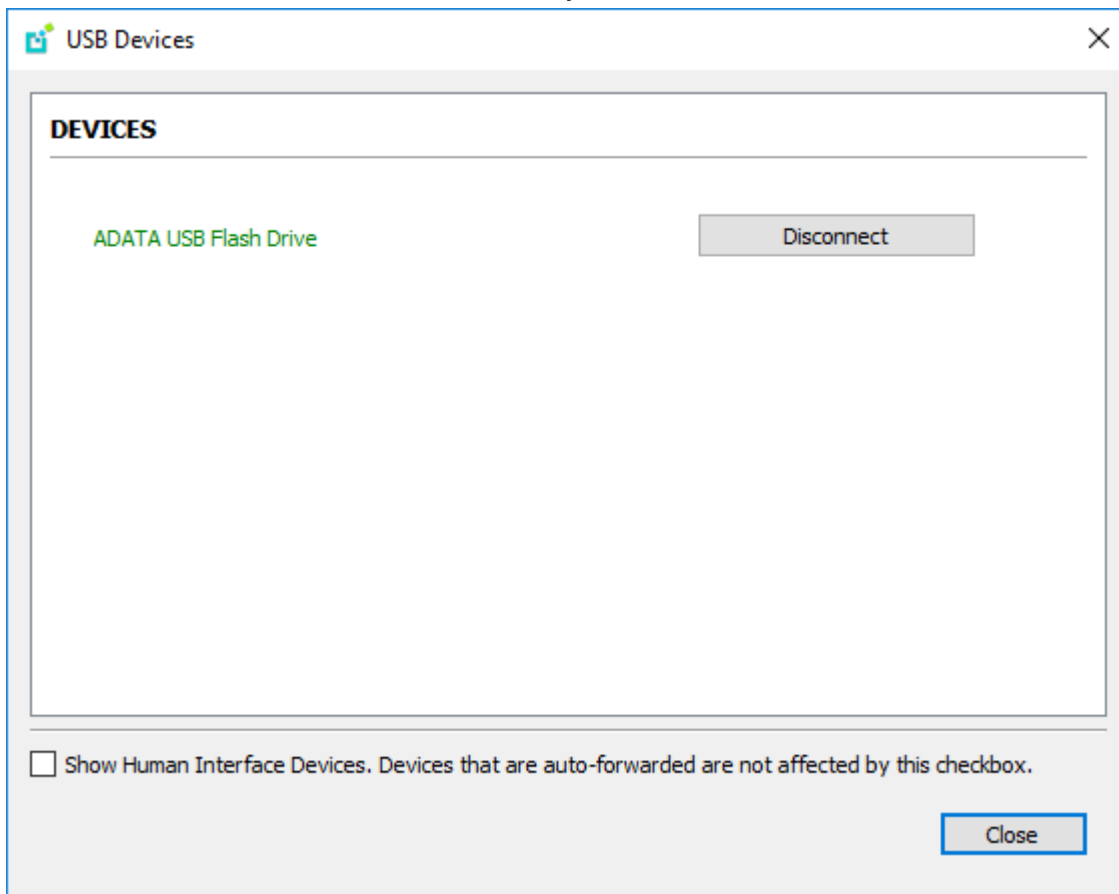
You may also have to complete additional configuration steps or install drivers on the host machine.

## To disconnect a USB device:

1. Select **Connection > USB Devices** from the PCoIP Software Client menu.

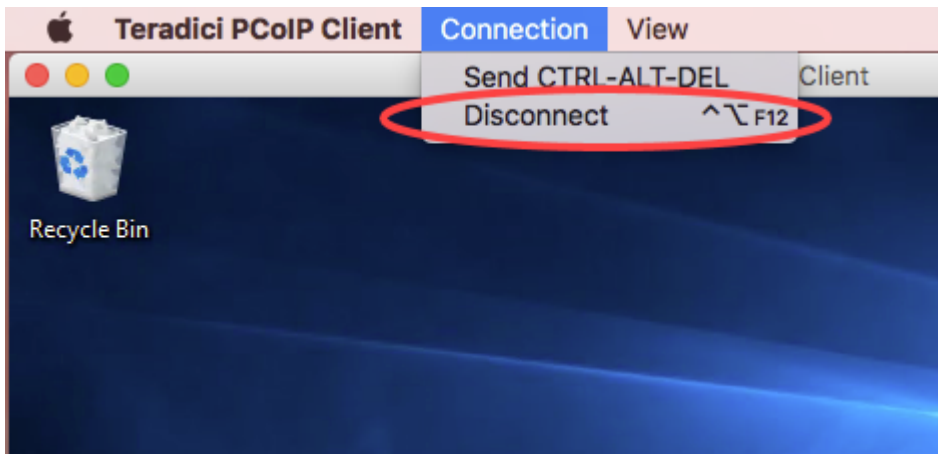
To disconnect a USB device:

2. Click **Disconnect** beside the USB device you want to disconnect.



# Disconnecting a Session

To disconnect a PCoIP session from either client, select the **Connection > Disconnect** menu option.



## Quickly Disconnect from a session

To quickly disconnect from a session, press **CTRL+Option+F12**

Quitting the application will also disconnect the current session.

## Session Reconnection

If a network interruption is detected, the PCoIP session enters a reconnecting phase. In this phase the client will show you the network reconnecting dialog which indicates that there is a network issue and that the client is trying to reconnect and re-establish the PCoIP session. You can click **disconnect** to cancel the attempted reconnect and disconnect the session completely. If the reconnection is successful, the notification dialog will disappear and the session will be restored, if not, the session will be disconnected completely.



# Changing the PCoIP Software Client Window Mode

You can use the PCoIP Software Client in full-screen or windowed mode. Full-screen mode is recommended in most cases.

## Activating Full Screen Modes

The PCoIP Software Client provides two full-screen modes: one monitor and all monitors.

**To switch from windowed mode to full-screen mode:**

- To use one full-screen display, select **View > Show Fullscreen One Monitor**.

All open windows and applications will be moved to a single full-screen display. This is equivalent to disconnecting all but one monitor from a physical host.

- To use all available full-screen displays, select **View > Show Fullscreen All Monitors**.

This mode shows full-screen displays on all client monitors.

### Keyboard shortcut

You can also enter full-screen mode by pressing **Ctrl+Option+Return** while in windowed mode. The shortcut will activate whichever full-screen mode was used last, or *all monitors* if no previous selection was made.

**To switch from Fullscreen One Monitor to Fullscreen All Monitors mode:**

1. Reveal the menu bar by moving the mouse cursor to the top of a client display.
2. Select **View > Show Fullscreen All Monitors**.

**To switch from Fullscreen All Monitors to Fullscreen One Monitor mode:**

1. Reveal the menu bar by moving the mouse cursor to the top of a client display.
2. Select **View > Show Fullscreen One Monitor**.

### Persistent Display Topology

Depending on the display topology mode you have selected, for example **Fullscreen One Monitor**, if you disconnect and then reconnect to the session, it will maintain that same mode upon reconnection. The state and mode will be preserved.

## Minimizing the PCoIP Software Client from a Full-screen Mode

To minimize a software client in full-screen mode:

1. Reveal the menu bar by moving the mouse cursor to the top of any display.
2. Select **View > Minimize Client**.

### Keyboard shortcut

You can also minimize the client by pressing **Ctrl+Option+m** while in any full-screen mode.

## Activating Windowed Mode

To switch from full-screen mode to windowed mode:

1. Reveal the menu bar by moving the mouse cursor to the top of any display.
2. Select **View > Leave Fullscreen**.

### Keyboard shortcut

You can also enter windowed mode by pressing **Ctrl+Option+Return** while in any full-screen mode.

# Enhanced Audio and Video Synchronization

Enhanced Audio and Video Synchronization provides improved full-screen video playback, reducing the difference in delays between the audio and video channels and smoothing frame playback on the client. This improves lip sync and reduces video frame drops for movie playback.

This feature introduces a small lag in user interaction responsiveness when enabled. Using enhanced audio and video synchronization will reduce the maximum frame rate.

Enhanced A/V Sync is enabled on a per-display basis, so you can dedicate individual displays to playback without impacting responsiveness on the others.

The enhanced A/V Sync feature is turned off by default and needs to be turned on for the menu items to appear.

## To turn on A/V Sync:

1. Execute the following command:

```
sudo defaults write "com.teradici.Teradici PCoIP Client.plist"  
enable_enhanced_avsync 1
```

The above text is OS specific. Note the setting changed from 0 to 1.

When enabled, a check mark will appear beside the enhanced A/V Sync menu item on that display. It is disabled on all displays by default.

## To use enhanced A/V Sync:

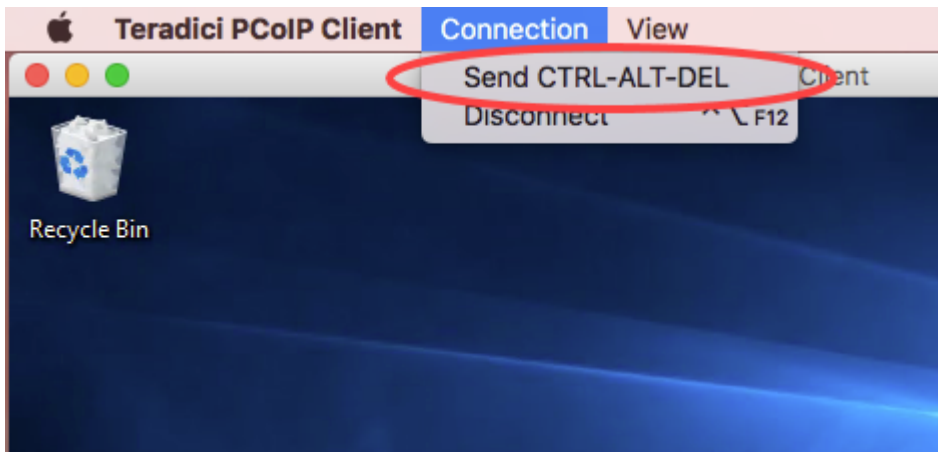
1. If you are in full-screen mode, reveal the menu bar on the display you want to enhance by moving the mouse cursor to the top of the screen.
2. On the display you want to enhance select **View>Enhanced A/V Sync** to toggle the enhanced sync mode.

**i Persistent Display Topology**

The Enhanced Audio and Video Synchronization feature is persistent across sessions from the same client, provided that the display topology has not changed.

# Sending a Ctrl-Alt-Del Command

To send the Ctrl-Alt-Del keyboard command to a remote workstation, select the **Connection > Send CTRL-ALT-DEL** menu option.



# Changing the Language

In addition to English, the PCoIP Software Client also supports these languages: Chinese (Simplified), French, German, Italian, Japanese, Korean, Portuguese (Brazil), Portuguese (Europe), Russian, Spanish, and Turkish. During installation, you can select one of the supported languages.

**To change the language after installation:**

- In **System Preferences > Language & Region > Advanced > General** tab, select another language in the **Format language** list.

# PCoIP Ultra

The PCoIP Client provides support for PCoIP Ultra, the latest protocol enhancements from Teradici. PCoIP Ultra is optimized for truly lossless support with bit-exact color accuracy and preservation of content detail at the highest frame rates.

PCoIP Ultra protocol enhancements propels our industry-recognized performance into the future of remote computing, with faster, more interactive experience for users of remote workstations working with high-resolution content.

PCoIP Ultra is disabled by default. To enable it, see [Enabling PCoIP Ultra](#).

## PCoIP Ultra Enhancements

In version 19.11, PCoIP Ultra provides the following benefits:

- Support for 4K/UHD high frame rate content.
- Efficient scaling across multicore CPUs leveraging AVX2 instruction sets.
- Highest image quality with efficient build-to-lossless and color-accurate.
- Dynamic network adaption and network resilience.

The Software Client for MacOS 19.11 with PCoIP Ultra contains certain limitations around USB and printer plugin redirection. The latest beta version of PCoIP Ultra is available via our Technology Preview version, as outlined below.

### PCoIP Ultra Technology Preview

PCoIP Ultra is an evolving technology, and new capabilities and enhancements are introduced frequently.

If you would like to test unreleased versions of PCoIP Ultra, we invite you to join the PCoIP Ultra Technology Preview program. Technology Preview users receive pre-release versions of Teradici software for use in non-production environments, and provide feedback to our engineering teams.

To learn more and to join the technology preview, visit the [Teradici Support site](#).

# Requirements

To take advantage of PCoIP Ultra, you need to meet these requirements:

- A PCoIP agent (any type) 19.05.0 or later.
- A PCoIP Software Client for macOS, PCoIP Software Client for Windows or PCoIP Software Client for Linux, 19.05.0 or later.

## **PCoIP Software Client for Mac users**

Support for PCoIP Ultra on the PCoIP Software Client for Mac is provided via the PCoIP Ultra Technology Preview as described above.

- The CPUs on both the agent and the client machines must support the AVX2 instruction set.



# Session Client Binary

The SDK is bundled with a session client binary, which can be invoked via command line or programmatically from a pre-session client. The session client for macOS is located in the SDK distribution on the following path:

- `"path-to-unzipped-sdk-package"/sdk/usr/bin/ClientSession.app`

For an example of programmatically invoking the in-session client, search for `launch_session` in `"path-to-unzipped-sdk-package"/sdk/usr/lib/pcoip-client/examples/broker_client_example/main.cpp`. After the PCoIP connection is established, several command line options are available from the in-session client, as documented in Customizing the PCoIP Session above.

# Customizable Session Features

The following PCoIP session features can be customized:

- Session Menu bar Visibility
- Disable Hot Keys
- Windowed or Fullscreen Mode
- Set Host Resolution
- Custom Client Branding
- Image Scaling
- Maintain Aspect Ratio
- USB Auto Forward
- USB VID/PID Auto Forward
- Disable USB
- Locale
- Session Log ID
- Log Level
- Log File Name
- Force Native Resolution

## Examples show command-line usage

The examples shown here invoke the session client via the command line. You can also set these properties when invoking the session client programmatically.

## Disable Session Menu Bar Visibility

To enhance the user experience the PCoIP Session Client enables the menu bar by default, however some use cases may require that it be disabled, or hidden, in order to prevent the user

from accessing menu functionality. To disable the menu bar feature use the parameter `disable-menubar`.

## Disable Hot Keys

To improve usability, session hot keys, such as **Ctrl+Option+F12**(which disconnects a PCoIP session)are available to users by default. The parameter for this feature is `disable-hotkeys`.

## Windowed or Fullscreen Mode

Depending on your application needs, you can display the PCoIP session in either windowed or fullscreen mode. Fullscreen mode allows the display topology to support multiple monitors as an extended desktop; windowed mode gives you the flexibility to display multiple application windows in parallel and switch between them quickly. Windowed mode improves the user experience, as well as resulting in an increase in performance. Windowed mode is the default mode, and to activate fullscreen mode use the `full-screen` parameter.

## Set Host Resolution

Normally, the session client opens with arbitrary window dimensions. In some cases, you may wish to lock the resolution of your host application displays. This ensures the user's viewing experience is consistent across different monitors and their native resolutions. The parameter for this feature is `set-host-resolution`.

- **Host Resolution Limitations:** It is only possible to specify one target resolution for all displays. The host resolution will not perform to its optimal capability if you have monitors with different resolutions.

## Custom Client Branding

You can customize the branding of your custom session client in several ways by creating a client branding package. These customizations affect the user's experience once they have connected to

their PCoIP session. The parameters for this feature are `branding-package` and `branding-hash`. The following elements can be customized in the session client:

- The OS application title and logo
- The session client toolbar title and logo
- The logo displayed in the OS taskbar
- The following default menu item label:
  - *About PCoIP Client*
  - *Quit PCoIP Client*
- The content shown in the **About** dialog:
  - Replace the dialog text
  - Provide hyperlinks to corporate resources and product information
  - Add a custom logo
- Customize a client alert and message window titles.

## Image Scaling

The image scaling feature enables scaling on the client without having to specify the desktop resolution. You can apply image scaling when the resolution of the client monitor is not the same as the resolution provided by the host. This feature provides a smoother process for image scaling on the client. The parameter for this feature is `enable-scaling`.

## Maintain Aspect Ratio

If the host and client aspect ratios do not match, and this parameter is not used then the display will be stretched to fit. The parameter for this feature is `maintain-aspect-ratio`. If the native aspect ratios of the host's display and the client's display do not match, the host's aspect ratio will be preserved and will appear in the client with black bars either on the sides or top and bottom of the display.

## USB Auto-Forward

Automatic bridging enables you to auto bridge all non-HID USB devices. Use the `usb-auto-forward` parameter.

## USB Vendor ID/Product ID Auto-Forward

You can automatically forward up to 20 USB devices to the host at the start of the session by calling the session client executable with `vidpid-auto-forward` and the required VID/PID pairs. Devices that are auto-forwarded will appear in the USB Devices dialog box, enabling users to connect or disconnect them from the host.

## Disable USB

You can disable USB functionality in the client with the `disable-usb` parameter.

## Locale

The Local feature enables you to use the appropriate localized user interface for the client session. This feature will make the session GUI more flexible to accommodate a wide range of languages. You can choose the language translation you require by setting the `locale` parameter. The following table states the available language translations and codes:

Locale Code	Language
de	German
es	Spanish
fr	French
it	Italian
ja	Japanese

Locale Code	Language
ko	Korean
pt	Portuguese (EU)
pt_BR	Portuguese (Brazil)
ru	Russian
tr	Turkish
zh_CN	Chinese (Simplified)
zh_TW	Chinese (Traditional)

### Default Language

By default, the language is set to English.

## Session Log-ID

`log-id` is a UUID that uniquely identifies the session in all PCoIP log files.

## Log Level

`log-level` is the log level parameter. It is possible to over-ride the default log-level, which is 2, by specifying a different log-level parameter. All messages at the specified level or lower will be logged. The following parameters apply:

- 0 = Critical
- 1 = Error
- 2 = Info
- 3 = Debug

 **Troubleshooting and Support**

When reproducing issues for the purposes of troubleshooting and support, set the log level to **Debug**. This will enable you to capture a log of all information messages and errors.

## Log File Name

The default location of the log file may be overridden using the `logfile` parameter to specify a full path and file name for the log-file.

 **Custom Location**

If a custom location is used for the log file then the support bundler script should be updated to capture the logs from that location.

## Force Native Resolution

The resolution of the client monitor can be set to the native resolution when the session client is launched using the `force-native-monitor-resolution` parameter

# Branding Your Session Client

You can customize the branding of your custom session client in several ways by creating a client branding package. These customizations affect the user's experience once they have connected to their PCoIP session. The following elements can be customized in the session client:

- The OS application title and logo
- The session client toolbar title and logo
- The logo displayed in the OS task bar
- The following default menu item label:
  - *About PCoIP Client*
- The content shown in the **About** dialog:
  - Replace the dialog text
  - Provide hyperlinks to corporate resources and product information
  - Add a custom logo
- Customize client alert and message window titles.



# Supporting USB Devices

Transferring non-HID USB devices from the client to the host is called bridging. Both the PCoIP agent on the host machine and the PCoIP client must enable bridging before devices can be transferred. Administrators can globally disable USB bridging support, or enforce device whitelists or blacklists, using GPO variables on the host machine. Clients cannot bridge devices that are disallowed by the agent.

There are two methods of providing USB support from your PCoIP client:

- **Automatic:** Automatically bridged devices are passed from the pre-session client to the session client executable, which forwards them to the host agent. No user interaction is required.
- **Manual:** Manually bridged devices are selected by the user, during a PCoIP session, from the session client UI.

## Remote USB Device Support

If a pointer type USB device, for example wacom tablets, mouse, or stylus device, is remoted, you need to grant *PCoIPClient.app* computer control to enable it to render and move the cursor. You can enable this through the **Preferences>Security & Privacy>Privacy>Accessibility** location on your system settings.

# System Precedence

The following section outlines the scope precedence commands between the **System Scope** and **User Scope**. If you are updating individual user settings then the user scope locations and parameters can be followed. Due to this order of precedence where by the system scope setting takes precedence over the user scope setting, a change in the user settings may not take effect if the system scope setting has been updated.

## System Scope

The *.plist* files are located in `/Library/Preferences/`. The following commands detail the read, delete and write sudo functions:

```
sudo defaults read com.teradici.Teradici PCoIP Client.plist <Key>
```

```
sudo defaults delete com.teradici.Teradici PCoIP Client.plist <Key>
```

```
sudo defaults write com.teradici.Teradici PCoIP Client.plist <Key> <value>
```

If **sudo** is not used in the command, it will automatically go to the user settings.

## User Scope

Within the user scope the *.plist* files are located in `~/Library/Preferences/`. The following commands detail the read, delete and write functions:

```
defaults read com.teradici.Teradici PCoIP Client.plist <Key>
```

```
defaults com.teradici.Teradici PCoIP Client.plist <Key>
```

```
defaults write com.teradici.Teradici PCoIP Client.plist <Key> <value>
```

If **sudo** is used, it will go to the system settings.

 **Re-boot Requirement**

The macOS machine may require a re-boot for the system configuration to take effect.

# Session API Change Log

This section outlines API call updates and changes for the session API from the different versions of the PCoIP Client SDK for macOS.

**20.01** - Removed invalid session num windows configuration item: See **i\_config\_provider** for more details.

**19.11** - No updates or changes.

**19.08** - Character arrays changes to standard C++ strings: See **i\_session** for more details.

# How to Establish a PCoIP Session

## Brokered Session Connection

If you are using a brokered session, this is handled by the broker libraries automatically.

Before you can establish a PCoIP session with a host desktop, gather the following host desktop details:

- IP address
- Port number
- Session ID
- Server name indication (SNI)
- Connection tag

This information can then be passed to the provided in-session client to establish a PCoIP session programmatically. See the example code for specific call syntax. In terms of programming interface, there are two ways that the connection and security information can be presented to `ClientSession.app`:

**Pass the pieces of information individually to the executable.** The following command invokes `ClientSession.app` to establish a PCoIP session and passes the connection and session information as parameters, where:

- **Connection tag:**

```
SCS1WsopFJ3iz1l48PTJMXFkcD4b6M9aiakHXH3e1lLhUR0BceWAifSSn%2b4AV1FC
8IihWVmsISmYFKeA25AtzFrdMpdaCtqlc0zfxAA
```

- **IP address:** `10.64.60.115`
- **Session ID:** `2305843009213693961`

```
open /Applications/ClientSession.app -i connect-tag=
SCS1WsopFJ3iz1l48PTJMXFkcD4b6M9aiakHXH3e1lLhUR0BceWAifSSn%2b4AV1FC
```

```
8IihWVmsISmYFKeA25AtzFrdMpdaCtqli0zfxAA address=10.64.60.115
session-id=2305843009213693961
```

Encode all information into a string container (URI) and then pass to the executable. The following command invokes `client_session` to establish a PCoIP session and passes the connection tag as a parameter and a URI encapsulating the IP address and Session ID in a string container, where:

- **Connection tag:**

```
SCS1WsopFJ3iz1l48PTJMXFkcD4b6M9aiakHXH3e1lLhUR0BceWAifSSn%2b4AV1FC
8IihWVmsISmYFKeA25AtzFrdMpdaCtqli0zfxAA
```

- **URI:** "teradici-pcoip://10.64.60.115:4172?sessionid=230584300921369396"

#### URI Format Documentation

There is a document describing the URI format in the root of the SDK.

```
open /Applications/ClientSession.app connecttag=
SCS1WsopFJ3iz1l48PTJMXFkcD4b6M9aiakHXH3e1lLhUR0BceWAifSSn%2b4A
V1FC8IihWVmsISmYFKeA25AtzFrdMpdaCtqli0zfxAA "teradicipcoip://
10.64.60.115:4172?session-id=230584300921369396"
```

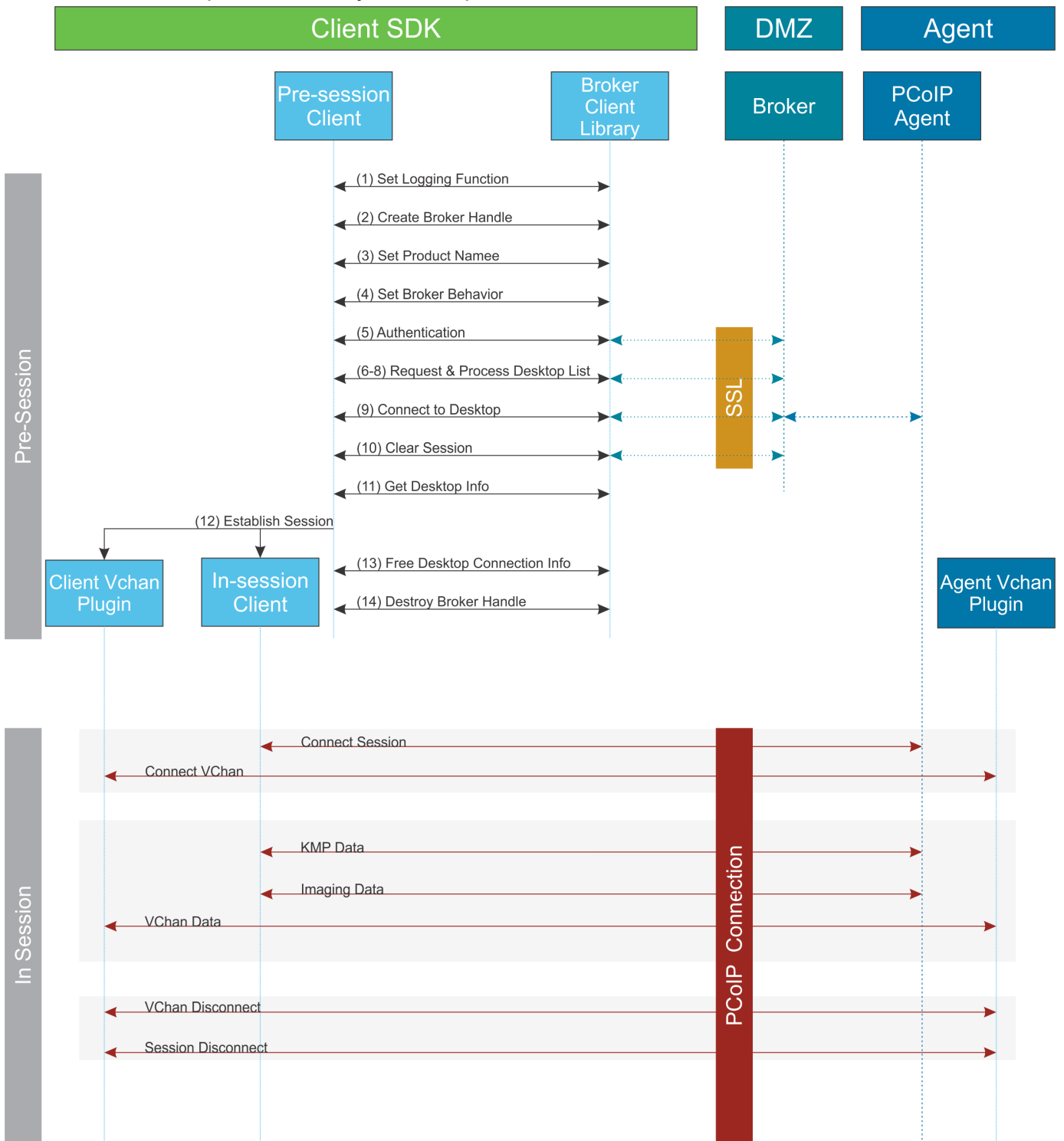
# PCoIP Session-Creation Steps and Actors

The steps indicated below are used and documented in the bundled sample code. Refer to the code for specific function calls, expected return values and error-handling requirements. The example C++ code can be found in the SDK package located here:

## **Direct (non-brokered) connections**

When there is no PCoIP broker in a system, as in direct connections, the PCoIP agent acts as its own broker. Clients make the same calls to the Broker Client Library whether there is a PCoIP broker inline or not.

### PCoIP Session Sequence used by the Sample Client



Each of these steps is used in the sample code, with a comment identifying the step number.



### Custom Log Implementations

You can design and implement your own logging functionality, so long as it follows the same callback signature of the log function template that is required by the PCoIP Client SDK API.

1. **Set a logging function** The `broker_client_library` requires users to provide a log function as part of the logging mechanism. A log function template is provided in the example code.
2. **Create a broker handle** Create a handle for the broker instance.
3. **Set client information** This information identifies your client to the broker. It should include the client name, client version, and client platform.
4. **Set broker address and behavior on unverified certification** This step identifies the address of the broker you want to connect to, and specifies error handling in the event the broker identity cannot be verified.
5. **Authentication between the broker and the client** This step requests an authentication method from the broker, and then submits the user's authentication information to the broker using the supplied authentication method. The client must implement all the authentication methods required by the broker.
6. **Request desktop list** Once the client is successfully authenticated by the broker, request a list of host servers (desktops) that the authenticated user is allowed to access.
7. **Retrieve desktop info** Loop through each desktop in the list acquired in step 6, requesting the name and ID of each desktop.
8. **Process the desktop list** Perform any processing required on the desktop list, and provide it to the user interface for selection.

### Desktop Selection Presentation Customization

At this stage, you can also customize the dialogue and interface the user will use to select a desktop.

9. **Connect to selected host server** This step asks the broker to set up the PCoIP session. The broker then contacts the agent, which supplies the necessary information (most notably the session tag) the client will need to establish the connection later. The PCoIP session is not established yet at this stage.
10. **Clear session with broker** On a successful connection, clear the broker session. This effectively disconnects the client from the broker.

11. **Get desktop connection information and launch the session** Request the connection and security properties from the desktop (for example, its IP address, its port number, or a session ID), and handle errors if any of the required properties are not returned.
12. **Proceed with established session** This step invokes `client_session`, and implements the actual PCoIP connection. For specific instructions regarding establishing PCoIP connections, see [How to Establish a PCoIP Session](#).
13. **Free desktop connection information** When the in-session client has been invoked, dispose of the collected desktop information.
14. **Destroy the broker client handle** Destroy the broker handle.

# Session Client Integration

There are two methods of integrating the in-session phase, integrating using the session client binary or using the session client API. The following diagrams show these methods in relation to integrating the SDK into a custom application:

## Session Client Binary Integration

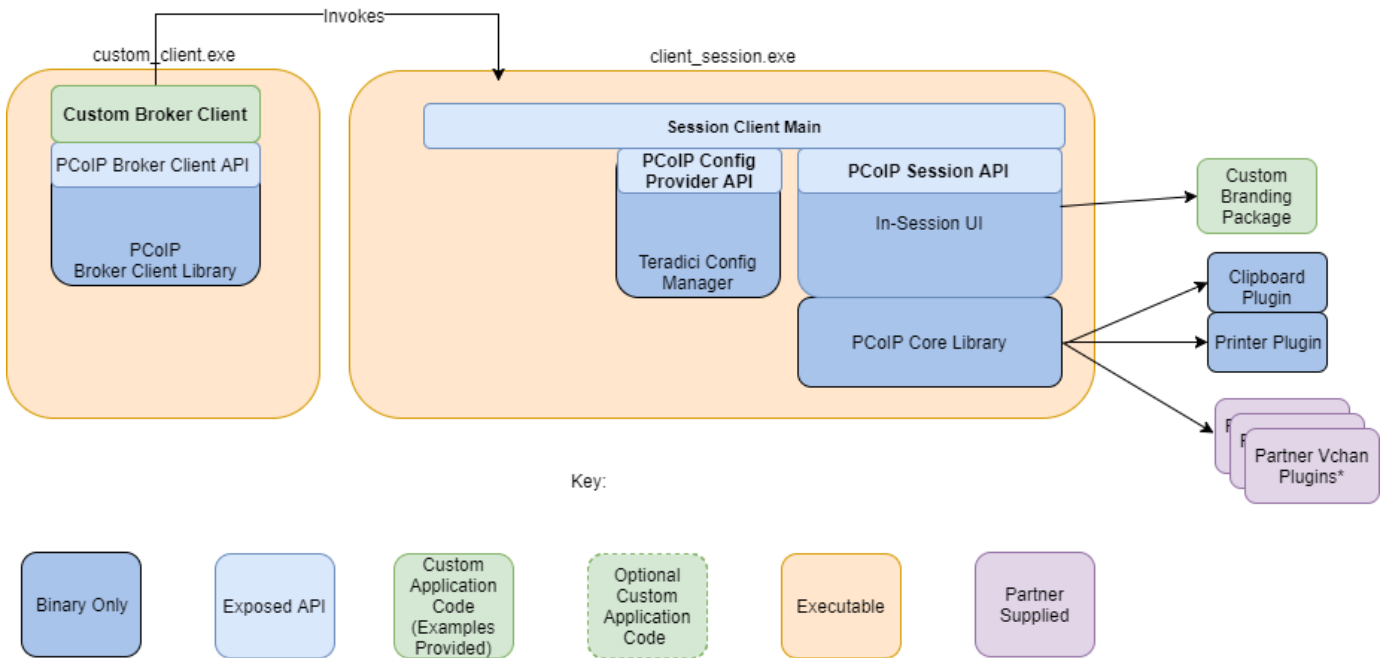
This method uses the session client binary as a separate in-session application. This is the simplest way to use the client SDK and it is recommended as the initial mode to use for developing your own client. In this mode, the pre-session and session phases are handled by separate executables:

- Write a custom pre-session executable according to your workflow and needs using the `broker_client_example` source code as a starting point. For more information see [Session Client API Integration](#).
- Use the stock `ClientSession.app` executable, which is located within the SDK at "`path-to-unzipped-sdk-package`"/`MacOS/ClientSession.app`, to establish the connection.

### Security Considerations

The values passed to the `ClientSession` app executable include sensitive information required to establish a session with the host. In particular, the connection tag is a single-use time-limited (60 seconds) token that allows the `ClientSession` app executable to connect to the host as an authenticated user. If an attacker is able to gain visibility to command line parameters as they are passed to client session, it is possible that they could use them before `ClientSession` app does and gain access to the host as that user. It is vital to ensure good security practices are applied to the client machine to prevent it from being compromised. This type of attack can be avoided completely by integrating the pre-session and `ClientSession` app into a single executable as described in the following section.

## Binary Integration



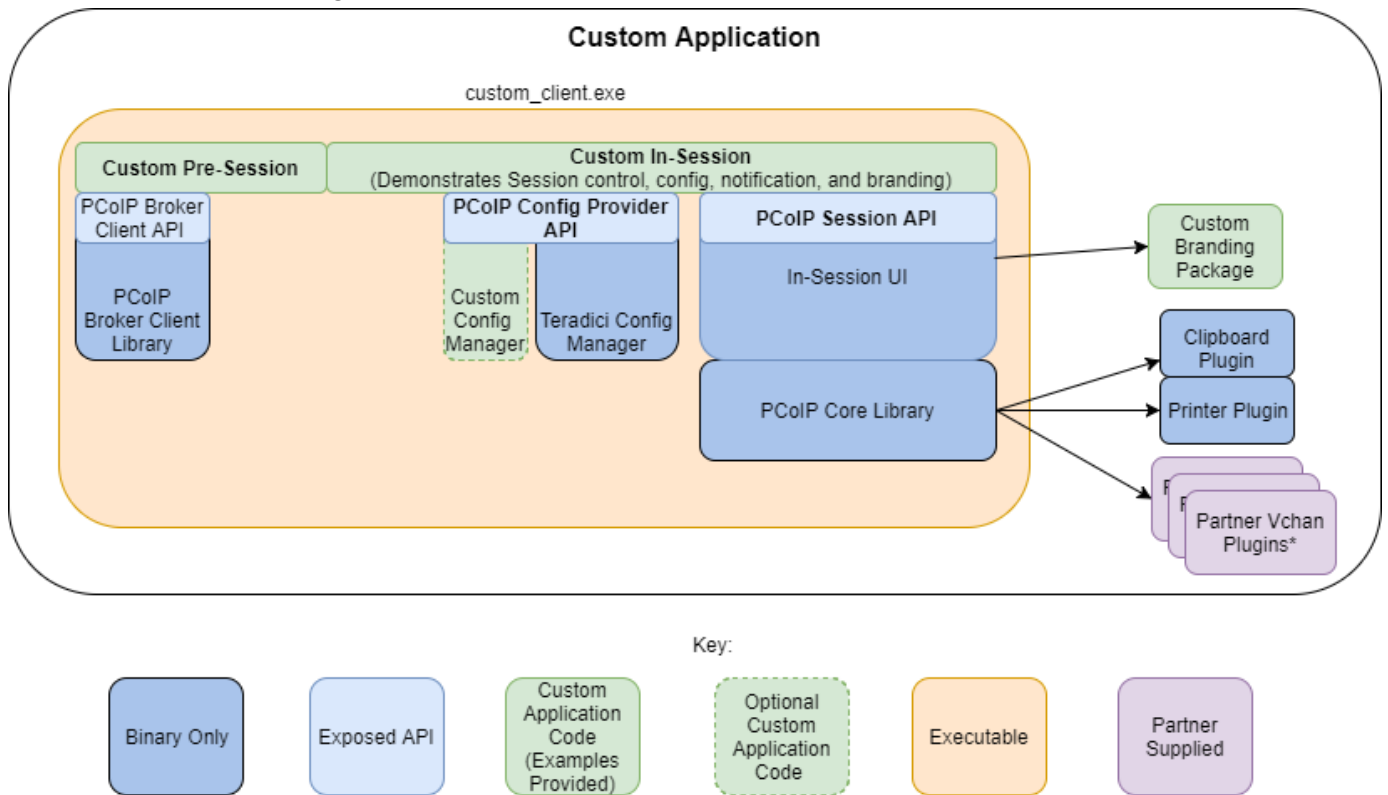
## Session Client API Integration

This method uses the session client API to integrate the in-session functionality into a custom application. This method is necessary if you wish to modify the behavior of the ClientSession app executable beyond that which is possible via its command line interface, or if you need to integrate the pre-session and ClientSession app into a single executable. In this mode you will use:

- The broker\_client\_example source code as a starting point for writing custom pre-session functionality
- The ClientSession app source code as a starting point for integrating with the Session Client API.

The Session Client API provides a simple high level C++ interface for configuring, starting, and stopping a session using the values obtained from a broker.

## Session Client API Integration



\*Partner Virtual Channel Plugins can be developed using the Virtual Channel SDK to enable this you can combine the PCoIP Client SDK with the PCoIP Virtual Channel SDK. For more information, see the [Virtual Channel SDK](#).

# Minimal Client Example

The following is an example of how to use the Teradici broker libraries and the PCoIP Session libraries to connect to an agent and launch a PCoIP session.

The following steps outline how to build this example:

1. Install the SDK.
2. Create a build/directory inside *minimal\_client/*
3. From *build/*, run cmake configure and build commands.
4. Update the login\_info.txt with the address and credentials of an authentic agent so that the built client can talk to the agent.
5. Run the built minimal\_client application. It should be able to launch a brokered PCoIP Session to the agent.

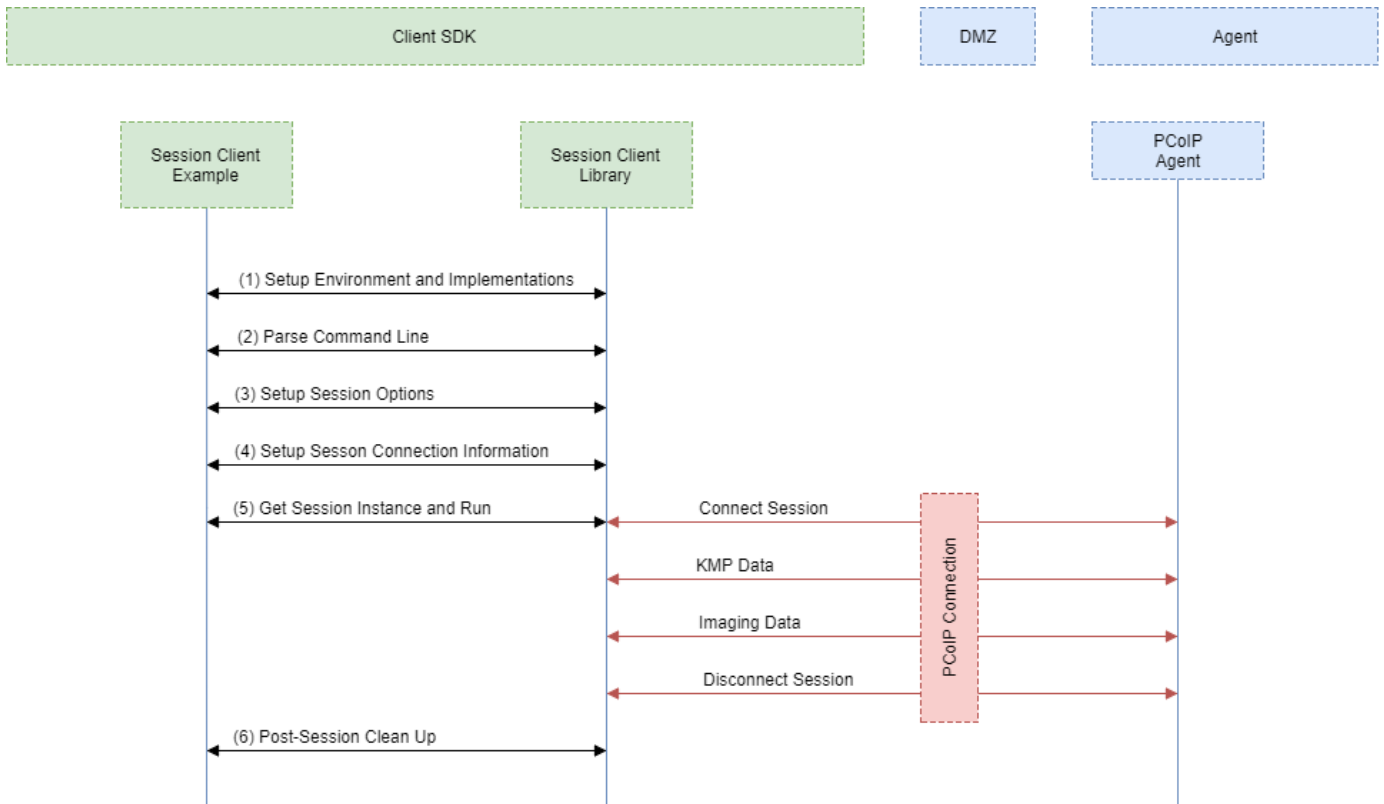
# Passing Customization Parameters to the Session Client

When using the `1` parameter to automatically pass session information to the session client, you can pass additional session client parameters by enclosing them in double quotes. This enables you to demonstrate session client functionality without racing to build a command line string within the session client's 60-second window. For example, to invoke the session client with the menu bar disabled, type:

```
./broker_client_example.app/Contents/MacOS/broker_client_example -i  
login_info.txt -l "disable-menubar"
```

# Using the Client Session API

Use of the Client Session C++ API is demonstrated in `[path to SDK]/sdk/usr/lib/examples/client_session_example_client_session_main.cpp`. It includes the steps outlined in the diagram below:



## PCoIP Session Client API Sequence Diagram

Each of these steps is used by the Session Client API, with a comment identifying the step number:

1. **Setup Environment and Implementations:** Os specific initialization. Instantiate a Configuration Provider object.
2. **Parse Command Line:** Define and parse supported command line parameters.
3. **Setup Session Options:** Validate the options passed via the command line and setup the Configuration Provider object accordingly.
4. **Setup Session Connection Information:** Pack the session parameter, received via the command line, into the structures required by the API.



5. **Get Session Instance and Run::** Obtain the main session object, set the Configuration Provider and run the session. The run call blocks until the session is terminated.
6. **Post-Session Clean-up:** Performs any post-session shutdown processing.

# Creating a Branding Text Layout File

The layout file format used to customize the session client is an UTF-8 XML text file. The layout schema is a top-level `<pcoip-client-branding/>` element with a version attribute describing the schema version, and containing the required elements described next:

```
<pcoip-client-branding version="1.0">
...
</pcoip-client-branding>
```

The available elements are outlined in the following table:

Parent Element	Child Element	Description
<code>&lt;app-name&gt;</code>		<b>Required!</b> The name of your custom session client. This will be used as the application window file.
<code>&lt;app-icon&gt;</code>		<b>Required!</b> The file name of your application icon. In Windows, this appears in the Windows toolbar and the window header.
<code>&lt;toolbar-menu&gt;</code>		<b>Required!</b> Describes the text labels used in OS toolbar menu's.
	<code>&lt;about-item&gt;</code>	The text label for the <b>About...</b> menu item. Optional. Without this field, there will not be an <b>About</b> menu item.
	<code>&lt;quit-item&gt;</code>	The text label for the <b>Quit...</b> menu item. Optional. Without this field, you will not have a <b>Quit</b> menu item.
<code>&lt;about&gt;</code>		<b>Required!</b> Describes the contents of the <b>About...</b> dialog. Must have the following required attributes: <b>title</b> ( <i>string</i> ): The dialog text and <b>minWidth</b> ( <i>number</i> ): The minimum pixel width of the dialog. For example: <code>&lt;about title="My Custom Client" minWidth="100"&gt;</code>

Parent Element	Child Element	Description
	<code>&lt;line&gt;</code>	Describes a line of text in the dialog. All lines are optional, but the <b>About</b> window will be empty unless you provide at least one. <code>&lt;line&gt;</code> accepts the following attributes: <b>align</b> ( <i>string keyword</i> ): The text alignment; for example, "center". This alignment applies to all child elements of the line. Lines can be self-closed to create a blank line (). <code>&lt;line&gt;</code> can contain the following elements: <code>&lt;logo&gt;</code> contains a filename for a logo or other graphic element: <code>&lt;logo&gt;about_logo.png&lt;/logo&gt;</code> . <code>&lt;text&gt;</code> contains the display text for each line: <code>&lt;text&gt;This text is displayed on the line. &lt;/text&gt;</code> . <code>&lt;hyperlink&gt;</code> takes a url parameter and creates a working hyperlink: <code>&lt;hyperlink url="www.teradici.com"&gt;Teradici&lt;/hyperlink&gt;</code> .

A full text layout file looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<pcoip-client-branding version="1.0">
<app-name>My Custom Client</app-name>
<app-icon>app_icon.png</app-icon>
<toolbar-menu>
<about-item>About My Custom Client</about-item>
<quit-item>Quit My Custom Client</quit-item>
</toolbar-menu>
<about title="About My Custom PCoIP Client" minWidth="0">
<line align="center"><logo>about_logo.jpg</logo></line>
<line />
<line align="center"><text>My PCoIP Client</text></line>
<line align="center"><text>Version 0.0.0</text></line>
<line align="center"><text>© Copyright 2016 My
Corporation</text></line>
<line align="center">
<hyperlink url="www.my-company.com">My company</hyperlink>
</line>
<line />
<line align="center">
<text>For help, click here: </text>
<hyperlink url="www.google.com">Google</hyperlink>
</line>
</about>
</pcoip-client-branding>
```

# Creating a Branding Package

In order to customize your session client, you must create a client branding package using the Teradici Custom Branding Package Utility. The Teradici Custom Branding Package Utility is located in the following location:

- macOS clients: "path-to-unzipped-sdk-package"/sdk/usr/bin/TeradiciBrandingPackageUtility

To create a custom branding package:

1. Create a product icon as a png file at 128px x 128px.
2. Create a company logo as a png file at any size.
3. Create a text layout file describing the customized UI element strings and dialog content.
4. Create the branding package with `TeradiciBrandingPackageUtility`.

```
TeradiciBrandingPackageUtility.exe -x my_custom_branding.txt -i  
my_custom_icon.png my_custom_logo.png -o my_custom_branding.bp
```

The system will respond with the output file and hash:

```
Output file: my_custom_branding.bp  
Hash:cbc3fd3c6d001a1e1f06342bccccf2a62bd748c3cf1dd2e4c9c29561ea07bd089
```

5. Note the output file name and the hash value. These will be passed to `client_session`.

# Using the Branding Package

Once you have created the branding package, it can be used by the session client. The pre-session client is responsible for verifying the package and passing it to the session client executable.

## To use the branding package:

1. Verify the branding package signature.
2. Call the session client executable and pass the branding package name and hash using the parameters `-branding-package` and `-branding-hash`.

For example (one command):

```
open /Applications/ClientSession.app -branding-package my_custom_
branding.bp
-branding-hash
cbc3fd3c6d001a1e1f06342bccc2a62bd748c3cf1dd2e4c9c29561ea07bd089
<other-params>
```

# Limits on Customization

The macOS has elements which are part of the operating system user interface and cannot be modified programmatically, as described in the following sections.

## macOS Limitations

The following run-time limitations are enforced by macOS on the application menu (beside the menu apple icon menu):

- The menu title cannot be altered. The menu title will be **PCoIP Client**.
- The *Hide...* menu item label cannot be altered. The menu item will be **Hide PCoIP Client**.
- The *Quit...* menu item label cannot be altered. The menu item will be **Quit PCoIP Client**.

The session client will also appear as **PCoIP Client** in the *Force Quit* dialog.

### Bypassing run-time Configuration Limitations

The limitations described here are enforced at run time. It is possible to bypass these restrictions by editing the `plist` file. Modifying this file will invalidate the Teradici signature.

### macOS 10.12 Application Support

In order to enable branding packages in macOS 10.12, you must remove the quarantine bit from all files in the session SDK package. If you To remove the quarantine bit, open a terminal and type:

```
xattr -dr com.apple.quarantine <PATH_TO_SDK_FOLDER>
```

# Core API Change Log

This section outlines API call updates and changes for the core API from the different versions of the PColP Client SDK for macOS.

**20.01** - No updates or changes.

**19.11** - Relative Mouse Feature: See [pcoip\\_core\\_api](#) for details.

**19.08** - No updates or changes.

# PCoIP Core Library Integration

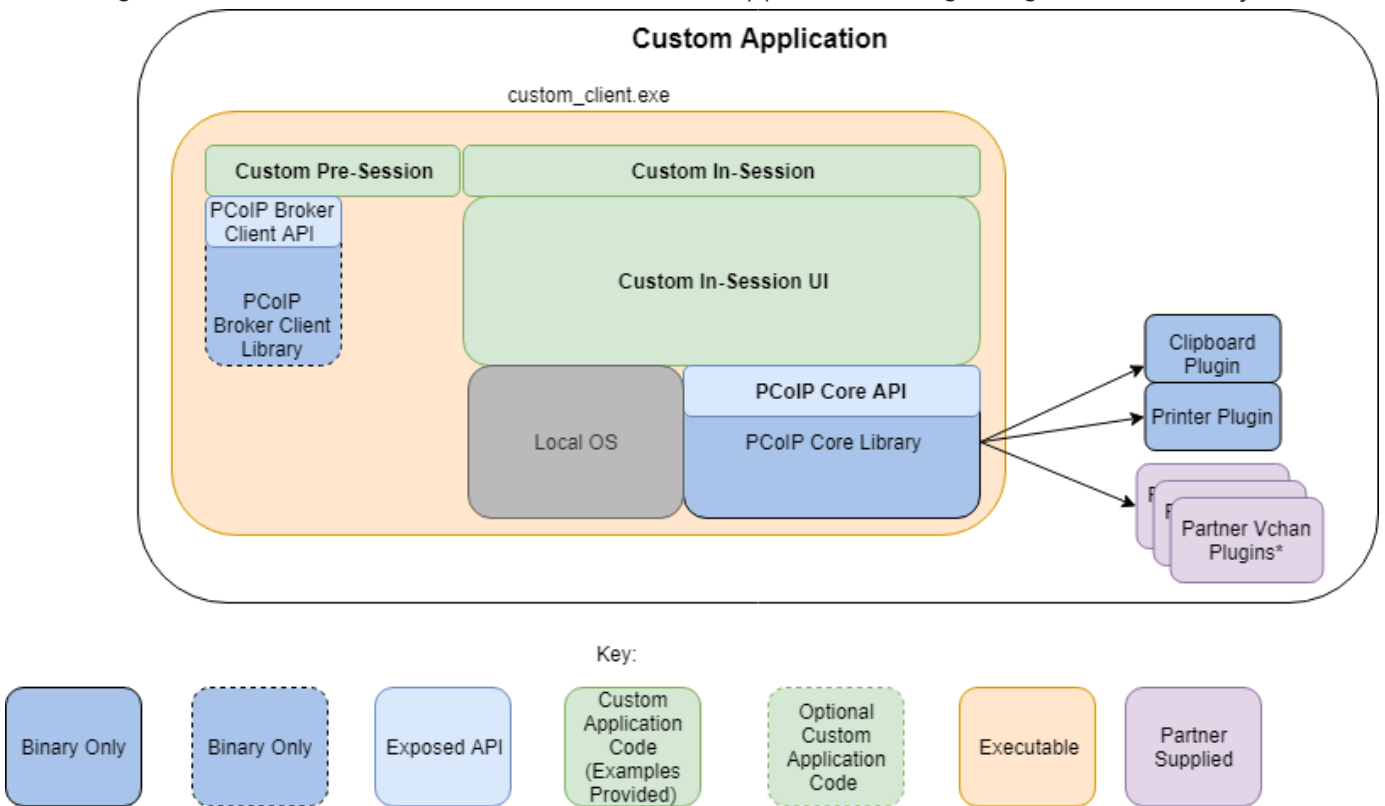
The core library allows an application developer to integrate a PCoIP session directly into an application. The core library facilitates connection to a remote host, decoding PCoIP image data directly into an application supplied frame buffer as well as remoting audio, keyboard and mouse events and supported USB devices. The application is responsible for all of the interactions with the window manager and the local operating system. The following diagram shows how an application would integrate the core library and broker library into a custom application.

## PCoIP Core API

This method uses the PCoIP Core API to integrate the PCoIP protocol into a custom application. This method is necessary if you want custom features in the client beyond that which is possible using the Session Client API.

### PCoIP Core API

This diagram shows the architecture with a custom application integrating the core library.





\*Partner Virtual Channel Plugins can be developed using the Virtual Channel SDK, to enable this you can combine the PCoIP Client SDK with the PCoIP Virtual Channel SDK.

## PCoIP Core API and Application Functionality

Using the Core API the application developer has more control over the client application than the PCoIP Session API but the developer must implement the integration between the local OS and the core library.

### Broker Interactions

The core library does not implement any of the broker protocol interactions. The application is responsible for obtaining the PCoIP agent address, port number and session tag, using the Broker Client API or some other brokering method, and passing these to the core library.

### Session Establishment

The application must provide the PCoIP agent address, port number and session tag to the core library in order to establish a PCoIP session.

### Display Topology

The application is responsible for deciding what displays to present to the user. The application may specify up to four displays with resolutions up to 4096x4096. The application must determine the correct display position and the rotations (if any).

### Keyboard Events

The application is responsible for handling local keyboard events, and providing the core library with scan codes to send to the remote host.

### Mouse Events

The application must provide the mouse events, with either absolute or relative coordinates, for the core library to send to the remote host.

## Touch Events

If the application supports touch events, the core library can forward these to the remote host.

## Cursor Handling

Local cursor handling is critical for providing the most responsive user experience possible. The application is responsible for updating the local cursor shape and position on call-back from the core library.

## USB Redirection

The application is responsible for determining if any USB devices should be connected to the remote session. The core library will disconnect the USB devices from the local system and redirect the device to the remote system.

## Session Reconnection

The core library supports reconnecting a session (for up to 20 minutes) in the event of a network disruption. The application is notified when a network disruption occurs and provides a method for terminating reconnection attempts.

## Session Termination

The core library provides a method for the application to terminate the session.

# Setting Up the Development Environment

Once you have successfully established a session between a PCoIP Software Client and a PCoIP host, you can start developing your own PCoIP client. To begin, set up your client development environment, as discussed next.

## To set up your client development environment:

Currently OpenSSL and Boost are required as third party libraries. Download and install the third party libraries to your system. The CMake modules provided automatically detects the system libraries at CMake configure time.

### CMake support for third-parties

When the third-party libraries are installed on the system, the CMake modules will automatically find them. Please see [Building the SDK for mac](#) section for instructions on building your applications using CMake.

1. Unzip the SDK tar file.
2. Copy the files from **usr/bin**, **usr/include/** and **usr/lib** to the corresponding system directories, **/usr/bin/**, **/usr/include** and **/usr/lib**.
3. Copy the `ClientSession.app` into **/Applications/ClientSession.app** if you like to use the `ClientSession.app` from the system Applications directory. You can also copy the `.dylibs` from the `ClientSession.app` to the directory where you are building your custom application.
4. At this point the SDK is installed on the system. For build instructions, please see [Building the SDK for macOS](#).

# Updating SDK Components

Updating the SDK to a new version can be done by replacing the old binaries with new versions in place. There is no special upgrade path. Upgrading components will not break compatibility with existing APIs.

# macOS Build Prerequisites

The following must be installed to build the PCoIP Client SDK on macOS:

- Xcode 10 <https://developer.apple.com/xcode/downloads/>
- Boost 1.63
- OpenSSL 1.1.1
- CMake 3.13 or above: [Download](#)

## CMake Version Requirement

The PCoIP Client SDK does not support versions of CMake higher than 3.13.

# Broker API Change Log

This section outlines API call updates and changes for the broker API from the different versions of the PCoIP Client SDK for macOS.

**20.01** - No updates or changes.

**19.11** - No updates or changes.

**19.08** - A Session Provision Error was added for provisioning failures.

# The Broker Client Example

The included sample broker client demonstrates how the APIs can be used to customize and control the pre-session and session phases of the connection.

## Code is an API Demonstration Only

The sample session client, described in the following sections, demonstrates a simple connection scenario using the supplied `broker_client_library`. The example unrealistically assumes that all requests and calls succeed as expected, and performs only basic error handling. An actual client implementation is likely to be far more complex; for example, you will need to handle failed broker certificate verification, account for other authentication steps beyond a simple user ID and password combination, and any other circumstances dictated by your system requirements.

The following steps outline how to build this example:

1. Install the SDK.
2. Create a build/directory inside `broker_client_example`.
3. From `build/`, run `cmake configure` and `build` commands.
4. Update the `login_info.txt` with the address and credentials of an authentic agent so that the built client can talk to the agent.
5. Run the `broker_client_example` application. It should be able to launch a brokered PCoIP Session to the agent by invoking the `session_client`.

## The Broker Client Example Sequence

This section describes how the broker client example implements the PCoIP session sequence. It also provides an overview of invoking and using the executable session client.

### Custom Broker Client Library Implementations

PCoIP clients interact with PCoIP-compatible brokers and PCoIP agents using an abstraction layer called a **broker client library**. The following example uses the supplied broker client library. You may, however, choose to write your own broker client library to meet specific requirements, or use a thirdparty broker library which does not use the PCoIP Broker Protocol. Refer to the [PCoIP® Connection Broker Protocol Specification](#) for details on how to design and implement your own connection broker.



# Using the Broker Client Example

The SDK provides a sample command line pre-session client called broker client example. This would enable you to call the included broker client libraries and establish a PCoIP connection. The broker client example demonstrates the success path for establishing new PCoIP sessions.

## Do not Use the Broker Client Example in Production

The broker client is provided as an example only, and should not be used in production. The client does not have thorough error handling and does not validate or sanitize user input.

The sample broker client is located here:

`"path-to-unzipped-sdk-package"/sdk/usr/bin/broker_client_example`

The two relevant files for the broker client example are:

- `broker_client_example`
- `login_info.txt`

The `broker_client_example` executable is the sample command-line client; the `login_info` text file contains authentication information used by the client.

## About `login_info.txt`

The broker client example uses a small local text file to supply session input values. The following is a sample `login_info.txt` file (one line):

```
sal-w2k8-ch605.autolab.local autolab autorunner "mypassword" sal-w2k8-ch605
```

In the example above:

- The **FQDN** of the host server is `sal-w2k8-ch605.autolab.local`
- The **domain** is `autolab`
- The **user** is `autorunner`

- The **password** is `mypassword`
- The **host name** is `sa1-w2k8-ch605`

Remote sessions established by `broker_client_example` are exactly the same as sessions established using the PCoIP software client, except that the input values are provided by `login_info.txt` instead of the client's user interface.

# Initiate Broker Connection Flow

To initiate the broker connection with the broker client example, set up your `login_info.txt` file and then call `broker_client_example` using `login_info.txt` as an argument.

To initiate the broker connection using the broker client example:

1. Open `login_info.txt` in a text editor.
2. Add the following information, in this order, separated by spaces:
  - The **FQDN** of the host server
  - The server **domain**
  - The **user name**
  - The **user password**
  - The **host name**
3. Save the text file.
4. Open a terminal window and type: `xmlbroker_client_example -i login_info.txt` The broker client example will display a status message similar to the one shown next:

```
Connected Successfully.
Desktop ID : sal-w7p64-sa15.autolab.local
ip_addr : 10.64.60.147
port : 4172
connect_tag:
SCS1fw0Zbk+Eu7q2iz0/M7mxfEE52au/3Jedtgp16L/rA8iB00+Er+YJd0yIL0xd9M
v5V0CDLSDmUNkOCwyyV1+u3w1aA7hXxEWmzhAA
session_id : 2305843009213693954
sni : SAL-W7P64-SA15
URI: "teradici-pcoip://10.64.60.147:4172?sessionid=
2305843009213693954&sni=SAL-W7P64-SA15", PARAMETERS: "connect-
tag=SCS1fw0Zbk%2bEu7q2iz0%2fM7mxfEE52au%2f3Jedtgp16L%2frA8iB00%2bE
r%2bYJd0yIL0xd9Mv5V0CDLSDmUNkOCwyyV1%2bu3w1aA7hXxEWmzhAA"
```

<<<<<< HEAD =====

||||| e2b618efae9bec0f92f0a4fa95e810d724588b2e

## Launching the Session Client from Broker Client Example

Use the `-l` switch (lowercase L, for launch) to have the broker client example invoke the session client. This enables you to send invoke the session client without worrying about the 60-second connect tag window.

To establish a new PCoIP connection using the `l` switch:

1. Open a command prompt and change directory to `bin`
2. Run the command line client, providing the `login_info.txt` file as an argument:

```
broker_client_example -i login_info.txt -l
<<<<<<< HEAD
```

...

```
||||||| e2b618efae9bec0f92f0a4fa95e810d724588b2e
```

# Linking the SDK for macOS

This section includes instructions for linking the SDK libraries with your custom client application:

1. Open a Terminal window and change directory to the root of the mounted SDK disk image.
2. Copy the SDK files to desired location for SDK development, referred to as `<sdk_location>` below.

## CMake modules location

The CMake modules for finding the libraries within the package are located inside the SDK framework / `<sdk_location> / PCoIPSoftClientSDK.framework/Resources/CMake`. The required libraries are namespaced by PCoIPSoftClientSDK.

3. To use the core APIs use the following commands on your applications CMake file for linking your application to the libraries that are needed from the SDK.

```
list(APPEND_CMAKE_PREFIX_PATH " /<sdk_location>/PCoIPSoftClientSDK.framework/
Resources/CMake" )

find_package(PCoIPSoftClientSDK REQUIRED)
```

4. Use the following command to link your application to the core library using the CMake command, replacing `<your_application>` with your own application name:

```
target_link_libraries(<your_application> PRIVATE
PCoIPSoftClientSDK::pcoip_core )
```

You can link to any other required libraries similarly by appending it to the above command.

# Setting Up a PCoIP Agent Test Environment

Before developing your custom client, you should set up a working PCoIP system. Teradici recommends establishing a small proof-of-concept system for custom client testing, consisting of a host machine with an installed PCoIP agent.

## License Requirement

Before using your test environment, you must install a PCoIP agent development license on the host machine. You received a license when you subscribed to a Teradici All Access solution, specifically Cloud Access or Cloud Access Plus. If you do not have a license, obtain one from Teradici before proceeding.

To establish a working proof-of-concept test system:

## PCoIP System Architecture Reference

For details about proof-of-concept deployments, including supported PCoIP agents, environments, and operating systems, see [Teradici All Access Architecture Guide](#).

1. Establish your host virtual machine and determine the PCoIP agent that best fits your actual PCoIP environment.
2. Install the PCoIP agent on the host machine. For PCoIP agent installation instructions, refer to the appropriate administrators' guide:
  - [PCoIP Standard Agent for Windows Guide](#)
  - [PCoIP Graphics Agent for Windows Guide](#)
  - [PCoIP Standard Agent for Linux Guide](#)
  - [PCoIP Graphics Agent for Linux Guide](#)
3. Install your agent license on the host machine. For license installation instructions, see the Administrators' Guide for your host machines PCoIP agent.

## Connecting To Your PCoIP Agent

Once your test system is set up, you can establish PCoIP connections to it using Teradici PCoIP Software Clients. For environment testing and troubleshooting purposes, the Teradici PCoIP Software Client is available here:

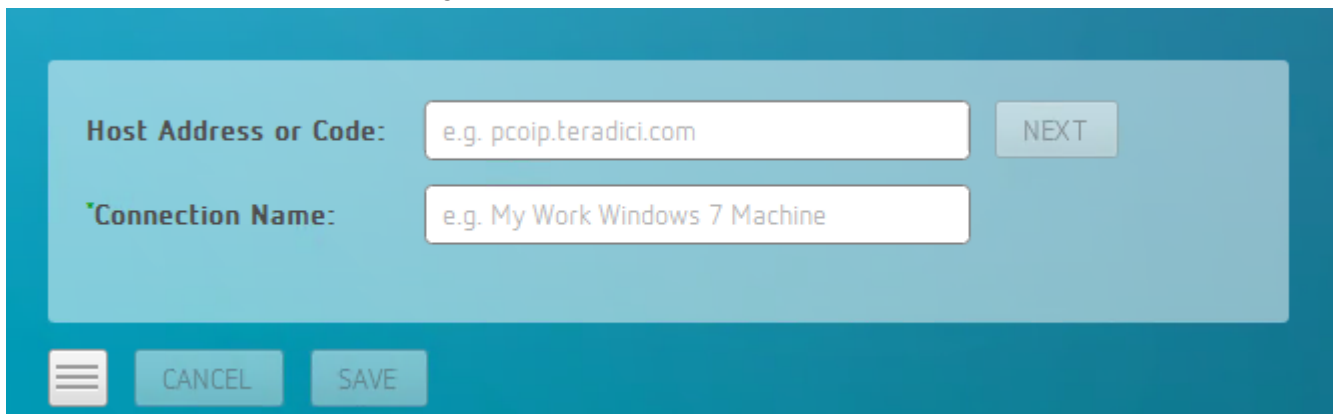
- [Teradici PCoIP Software Client for Mac](#)

## Establishing a PCoIP Connection Using a Teradici PCoIP Software Client

To test your development environment, make a direct (unbrokered) connection to your development host using a Teradici Software Client. If you are able to connect using a Teradici software client, your host is correctly configured. The following illustrations show examples of the pre-session phases using a Teradici software client:

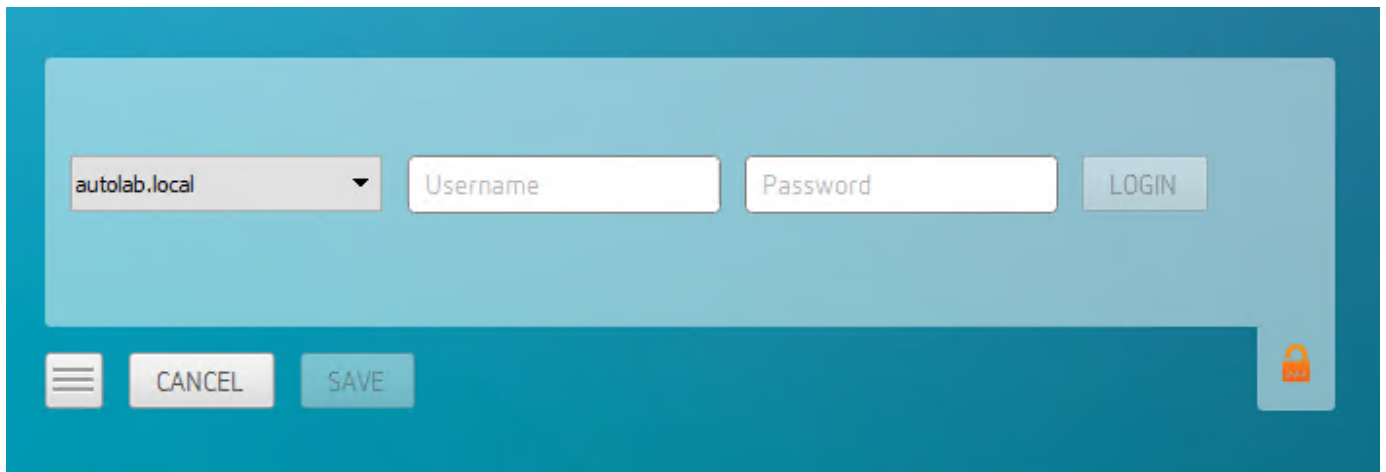
### Pre-Session Connection

The user of the PCoIP Client request the PCoIP Agent by providing the FQDN of the remote workstation where the PCoIP Agent has been installed.



The screenshot shows a dialog box for establishing a PCoIP connection. It features two input fields: 'Host Address or Code' with the example 'e.g. pcoip.teradici.com' and 'Connection Name' with the example 'e.g. My Work Windows 7 Machine'. A 'NEXT' button is positioned to the right of the first field. At the bottom, there is a hamburger menu icon, a 'CANCEL' button, and a 'SAVE' button.

The user of the PCoIP client is then authenticated. If authentication is successful, a PCoIP session will be launched.





# PCoIP Support Bundler Tool

Teradici may request a support file from your system to help troubleshoot and diagnose PCoIP issues.

## Customizable Script

This script can be customized, in the case where the log files are located in a different location the script should be updated appropriately.

To create a support file:

1. Open a terminal window.
2. Launch the support bundler:

```
path-to-unzipped-sdk-package"/sdk/Library/Frameworks/  
PCoIPSoftClientSDK.framework/Examples/support-bundler/pcoip-client-support-  
bundler
```

The file will be created and placed in the user's home directory.

# Manually Bridging USB Devices

If you need to support more than 20 USB devices, or if you expect your users to control which devices can be bridged, they can be manually added by opening the client's **USB Devices** menu and enabling them.

# Frequently Asked Questions

The following are answers to commonly asked questions when contemplating how to develop custom PCoIP Clients using the Teradici PCoIP Client SDK.

**Q: Can I brand the pre-session client with my company logo and colors?**

**A: Yes.** Your Teradici Cloud Access Platform agreement will contain detailed information about corporate logos. Follow the Teradici branding guide for including the PCoIP trademark in your final design.

**Q: Are there guidelines for using the Teradici and PCoIP Brand?**

**A: Yes.** Refer to <http://www.teradici.com/docs/brand-guide> for details.

**Q: Does the SDK support localization?**

**A: Yes.** In pre-session you have complete flexibility to create clients that incorporate customizations. In-session, you can use the `locale` parameter to pass a locale to `ClientSession`.

**Q: Does my client need Teradici licenses to operate?**

**A:** The client itself does not need a license to operate, but the PCoIP agents that it connects to do require licenses. License handling is performed by the PCoIP Broker or PCoIP agent, depending on the connection type. It is not handled by the client.

**Q: How can I add additional functionality to my PCoIP Client?**

**A:** If you have requirements that go beyond the default capabilities of the Client SDK, you can integrate the PCoIP Virtual Channel SDK. The Virtual Channel SDK gives you the ability to create custom PCoIP Virtual Channel plugins which stream data between clients and hosts.

**Q: Will my client work with all PCoIP agents?**

**A: Yes.** The PCoIP protocol works with Cloud Access and Cloud Access Plus.

**Q: What support options are available for the Client SDK?**

**A:** Teradici offers Developer Support and Professional Services options for SDKs and APIs. Please contact your Teradici account manager for details.