PCoIP Client SDK for Linux Developers' Guide

23.04

Table of Contents

PCoIP Client SDK for Linux 23.04

The Client SDK for Linux provides developers tools to create a custom PCoIP client with a unique user interface and workflow, or to embed a PCoIP session into another program or solution. Clients built with the SDK can connect to remote desktops using PCoIP agents or PCoIP Remote Workstation Cards.

The SDK is provided as part of HP Anyware by special agreement with HP.

The Client SDK for Linux includes the following components:

- Code Examples

- Programming Guide

- Session Client Source Code

- 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 Client SDK for Linux can be used on the following operating systems:

- Ubuntu 18.04

- Ubuntu 20.04

- Ubuntu 22.04

- RHEL/CentOS 7.9 *(Technical Preview)*

What Can You Build With the PCoIP Client SDK?

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 or trademarks, and 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 PCoIP protocol, how it is used, and the difference between brokered and non-brokered sessions.

This guide is not intended for users who do not require a custom PCoIP client.

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 HP Anyware 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

- Troubleshooting issues related to setting up, developing, and building the SDK

Developing with the Session API and Core API

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?

In addition to **bug fixes and stability enhancements**, the PCoIP Client SDK for Linux 23.04 introduces the following:

Support for Tangent Panels

The Client SDK for Linux now supports certain Tangent panels. For a list of supported panels, see Supported Tangent Panels.

Support for PCoIP Ultra Collaboration in SDK Clients

The PCoIP Client SDK for Linux 23.04 now includes a method that consumes the Invitation URI for a PCoIP Collaboration Session, enabling SDK clients to join Collaboration sessions. For more information, see The Collaborator Broker Client Example.

> 🔥 **Important: Maximum Number of Collaborators**
>
> Up to 5 collaborators can join a single collaboration session. Once 5 collaborators join a collaboration session, additional participants will not be able to join, unless one or more collaborators leaves the session.

Support for Wacom Cintiq Pro 16 (DTH-167)

The Client SDK for Linux now supports the newer version of the Wacom Cintiq Pro 16 (DTH-167). This device is supported with both local termination and bridged connection modes when connecting to PCoIP agents for Windows and Linux (PCoIP Graphics Agent for macOS is not supported).

System Requirements

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

System	Version Required
PCoIP Software Client Operating System	• Ubuntu 18.04 • Ubuntu 20.04 • Ubuntu 22.04 • RHEL/CentOS 7.9 *(Technical Preview)*
Compatible PCoIP Agents	The Client SDK for Linux can connect to any PCoIP agent. Some features require specific agent versions; see the *Feature Support* section of this guide for details. We recommend always using the same version of PCoIP agent and PCoIP client.
Compatible PCoIP Remote Workstation Cards[1]	TERA22x0 with firmware 20.04+ and PCoIP Remote Workstation Card Software for Windows or Linux 20.04+.
Supported IP versions	IPv4 and IPV6.

Hardware System Requirements

For different display configurations Teradici recommends certain processor and RAM combinations:

• For up to dual 1920 x 1080 display configuration Teradici recommends 1.6 GHz dual core processor or higher with at least 4 GB RAM.

• For up to dual 4K/UHD Teradici recommends a 3.0 Ghz quad core processor or higher with at least 8GB Dual Channel RAM.

---

1. For details on feature limitations between PCoIP Software Clients and PCoIP Remote Workstation Cards, see [Connecting to PCoIP Remote Workstation Cards](#). ↵

About the PCoIP Client SDK for Linux

The Client SDK for Linux does not ship with its own session client binary; it uses the standard [PCoIP Software Client for Linux](#) binary instead. Download and install the PCoIP client application, and then invoke it from the SDK.

> 🔥 **Important: If you have used previous versions of the Client SDK for Linux**
>
> Prior to the 23.01 release, the Client SDK for Linux shipped with an included session client binary. If you are coming from a previous release, you should install the [PCoIP Software Client for Linux](#), and update your code to use the new file location.

Once installed, the binary can be found here:

```
/usr/bin/pcoip-client
```

For an example of programmatically invoking the in-session PCoIP client, search for `launchSessionClient` in the following file:

```
/usr/lib/x86_64-linux-gnu/pcoip-client/examples/broker_client_example/main.cpp
```

After the PCoIP connection is established, several options are available from the in-session client. For details, see the URI configuration options described in the [PCoIP Software Client for Linux Administrators' Guide](#).

Listing the SDK Package Files

1. Fetch the latest .deb package for the SDK by running the following command:

```
apt-get download pcoip-client-sdk
```

2. To list the contents of the SDK package, run the following command, where `pcoip-client-sdk_23.04.1-18.04_amd64.deb' is the name of the SDK package:

```
dpkg -c pcoip-client-sdk_23.04.1-18.04_amd64.deb
```

Feature Support

Wacom Tablet Support

The Software Client for Linux supports Wacom tablets in two configurations: *bridged*, where peripheral data is sent to the desktop for processing, and *locally terminated*, where peripheral data is processed locally at the Software Client.

Locally terminated Wacom tablets are much more responsive and tolerate low latency connections.

Whether the Wacom tablet is locally terminated or bridged, the process of remoting the USB device is the same, you need to click connect in the USB menu or auto forward the device. If local termination is supported for the particular device, it will automatically be chosen over bridging. For more information on this, see Local Termination Blacklist.

Locally Terminated Wacom Tablets

Local termination requires:

- A Standard agent or Graphics agent for Windows version 21.01 or higher.
- A Standard agent or Graphics agent for Linux version 21.01 or higher.

PCoIP client support for *locally terminated* Wacom tablets and the Software Client for Linux

	PCoIP Standard Agent for Linux	PCoIP Graphics Agent for Linux	PCoIP Standard Agent for Windows	PCoIP Graphics Agent for Windows	PCoIP Remote Workstation Card
Intuos Pro Small *PTH-460*	✔	✔	✔	✔	—
Intuos Pro Medium *PTH-660*	✔	✔	✔	✔	—
Intuos Pro Large *PTH-860*	✔	✔	✔	✔	—
Cintiq 22HD *DTK-2200*	✔	✔	✔	✔	—
Cintiq Pro 24 - Pen Only *DTK-2420*	✔	✔	✔	✔	—
Cintiq 22 *DTK-2260*	✔	✔	✔	✔	—
Cintiq 22HDT - Pen & Touch *DTH-2200*	✔	✔	✔	✔	—
Cintiq Pro 24 - Pen & Touch *DTH-2420*	—	—	—	—	—
Cintiq Pro 32 - Pen & Touch *DTH3220*	✔	✔	✔	✔	—

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. A HID Local Termination Blacklist has been added to override the preferred local termination mode. Devices on the blacklist would be bridged to the remote desktop. For more information, see [Local Termination Blacklist](#).

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

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

	PCoIP Standard Agent for Linux	PCoIP Graphics Agent for Linux	PCoIP Standard Agent for Windows	PCoIP Graphics Agent for Windows	PCoIP Remote Workstation Card
Intuos Pro Small *PTH-460*	✔	✔	✔	✔	—
Intuos Pro Medium *PTH-660*	✔	✔	✔	✔	—
Intuos Pro Large *PTH-860*	✔	✔	✔	✔	—
Cintiq 22HD *DTK-2260, DTK-2200*	✔	✔	✔	✔	—
Cintiq Pro 24 *DTK-2420*	✔	✔	✔	✔	—
Cintiq 22HDT - Pen & Touch *DTH-2200*	✔	✔	✔	✔	—
Cintiq Pro 24 - Pen & Touch *DTH-2420*	✔	✔	✔	✔	—
Cintiq Pro 32 - Pen & Touch *DTH3220*	✔	✔	✔	✔	—

Known Issues with Wacom Tablets and Functionality

Whilst testing and developing the compatibility of HP Anyware with different Wacom tablets, certain performance issues arise. The following is a list of the current known issues with certain Wacom tablets:

- The touch feature only works on the Cintiq Pro 32 Pen & Touch (DTH-2420). It does not work on any other bridged and locally terminated devices across all platforms.

- ExpressKey Remote does not work on the Wacom Cintiq Pro 32 (DTH-3220). You should still remote to this device when remoting to the Wacom tablet.

- There are cursor limitations when working with the Wacom Cintiq 22HD (DTK-2200) and Wacom Cintiq Pro 24 (DTK-2420) for both bridged and locally terminated devices across all platforms.

- There is a issue with the control buttons on the Wacom Cintiq Pro 32 (DTH-3220) device. The buttons do not function on locally terminated devices across all platforms.

Wacom Cintiq Pro 32 Tablet Selection

The Wacom Cintiq Pro 32 shows up as multiple devices in the USB menu. You should remote the following USB devices as outlined in the image below to remote to this Wacom tablet:

- ExpressKey Remote

- Cintiq Pro 32 Touch

- Wacom Cintiq Pro 32

Alt Text

A HID Local Termination Blacklist has been added to override the preferred local termination mode. Devices on the blacklist would be bridged to the remote desktop. For more information, see [Local Termination Blacklist](#).

USB Support

PCoIP Clients supports redirecting USB devices to a remote session. Administrators can set rules governing allowed and disallowed devices, device classes, or device protocols.

> 🔥 **Important: USB support is enabled by default**
>
> USB bridging is enabled by default. If you want to restrict or disable USB support, you can globally disable or set rules governing USB behavior via GPO settings on the PCoIP Agent.

Isochronous USB device support

USB devices with time-sensitive information, such as webcams, are not generally supported. However, Teradici's technology partners provide additional solutions to expand peripheral support such as webcams. For more information, look for partners listed under *Peripherals* on the [Teradici Technology Partners](#) page.

Webcam Support

The PCoIP Software Client for Linux now supports USB webcams when connecting to a PCoIP Agent for Windows. USB webcams can now be used while in the remote desktop, including with applications such as Microsoft Teams or Zoom.

For detailed information which models have been tested and the performance metrics associated with these models see here. This knowledge base article also deals steps on how to test and verify other webcam models.

As of 21.07, this feature is enabled by default.

Requirements

Webcam support in Anyware requires the following:

• PCoIP Software Client for Windows or PCoIP Software Client for Linux, 21.07+

• PCoIP Standard Agent for Windows or PCoIP Graphics Agent for Windows, 21.03+

• USB-attached webcam.

Notes and Limitations

• Webcams must be connected via USB. Webcams that are not USB, such as embedded laptop webcams, are not supported.

• Linux agents are not supported.

• PCoIP Software Client for macOS is not supported.

Setup

On the PCoIP Software Client, connect the webcam as described in the following sections:

• PCoIP Client for Windows: USB Bridging of Webcams.

• PCoIP Client for Linux: USB Bridging of Webcams.

PCoIP Sessions

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 virual 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
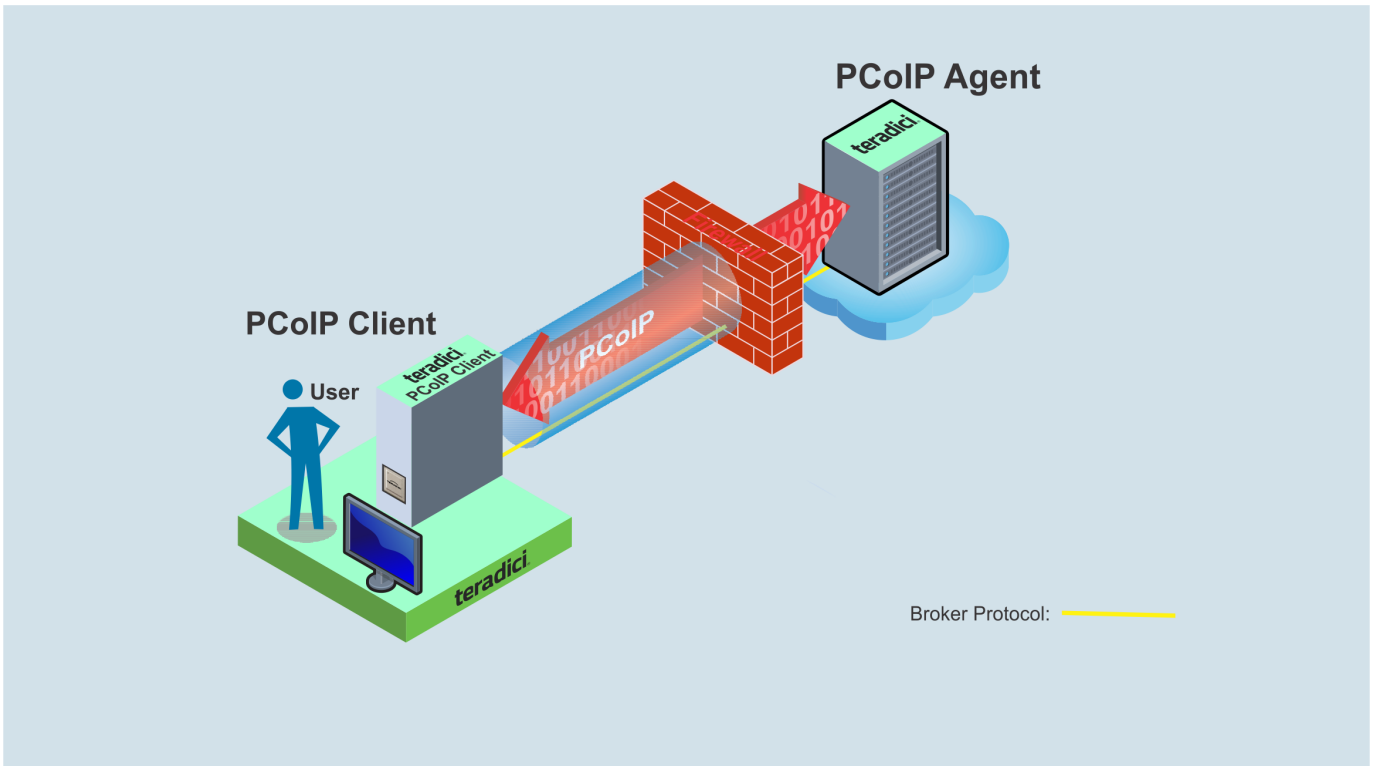
Brokered Connection
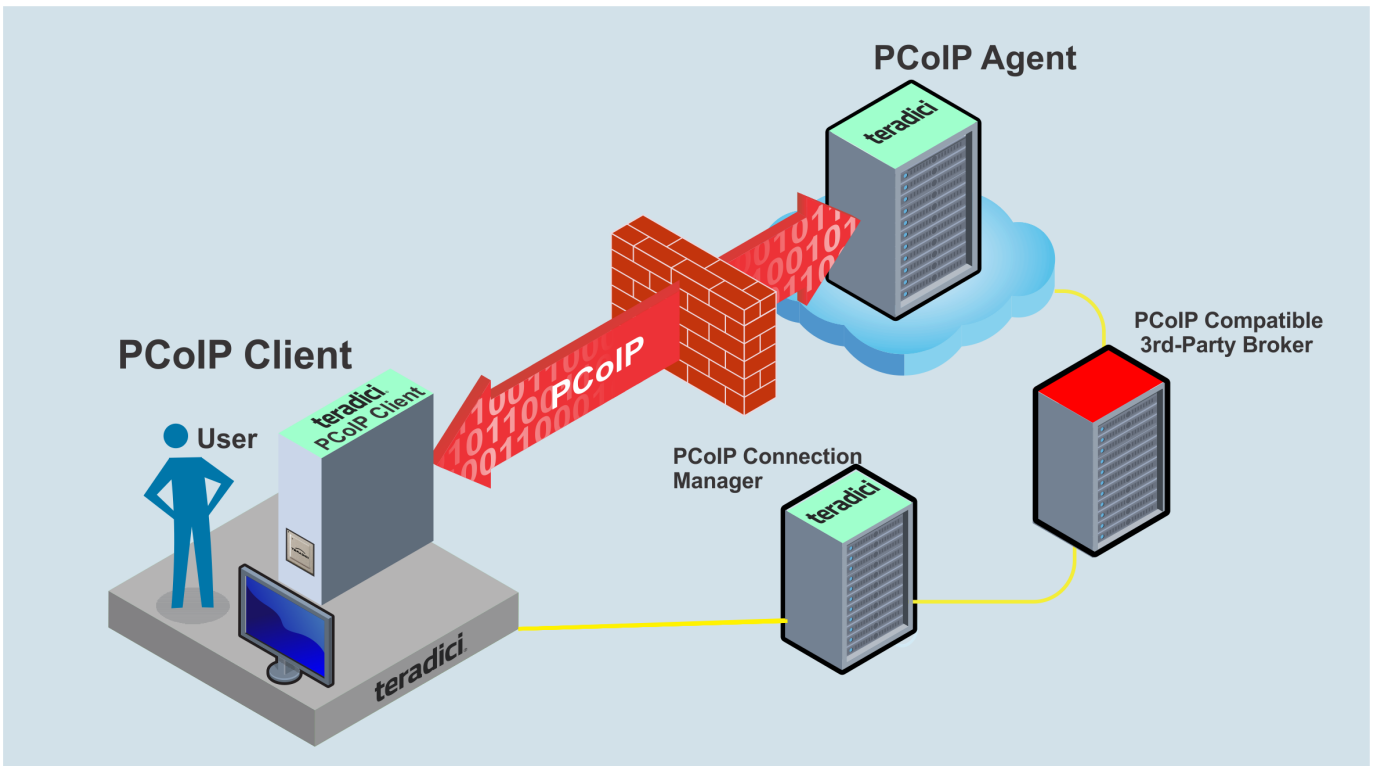
PCoIP Session Phases

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

Installing the PCoIP Client SDK for Linux

Installing the PCoIP Client SDK for Linux

1. Install the PCoIP Client SDK for Linux repository. A shell script to install this is <u>here</u>.

> ✏️ **Teradici Distribution Platform**
>
> Teradici's distribution platform has recently changed. The new system requires access to the dl.teradici.com subdomain. The old system, which used downloads.teradici.com, is deprecated and will be decommissioned in the future.

> ⚠️ **teradici-repo Package**
>
> If you do not install the teradici-repo package then you will not be able to successfully install the PCoIP Software Client. You may be experiencing this issue if you see an error message stating **Unable to locate pcoip-client**. Please ensure you download and install the repo.

2. Install the PCoIP Client SDK for Linux:

```
sudo apt update
sudo apt install pcoip-client-sdk
```

3. Launch the *pcoip-client-sdk* to create default configuration files and then quit the client.

Feature Support and Configuration

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.

To connect to a USB device:

1. Attach the USB device you wish 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*.

3. Click **Connect** beside the USB device you wish to use.
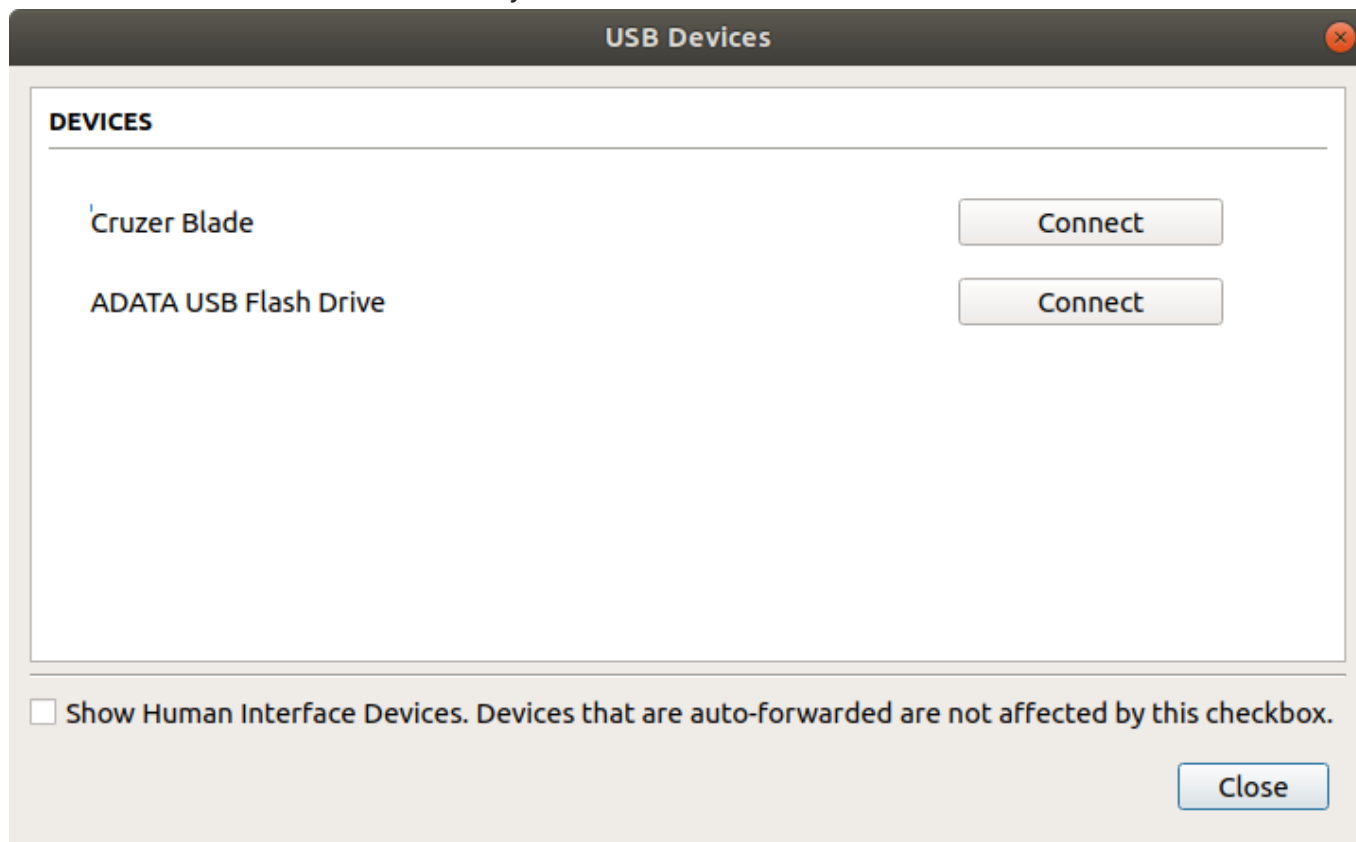
> 🔥 **Connecting to Human Interface Devices**
>
> Most Human Interface Devices (HIDs), such as keyboards and mice, are automatically handled by the PCoIP Software Client and do not appear on in this list even if they use a USB connection.
>
> If you need to connect a Human Interface Device that cannot 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.

2. Click **Disconnect** beside the USB device you wish to disconnect.
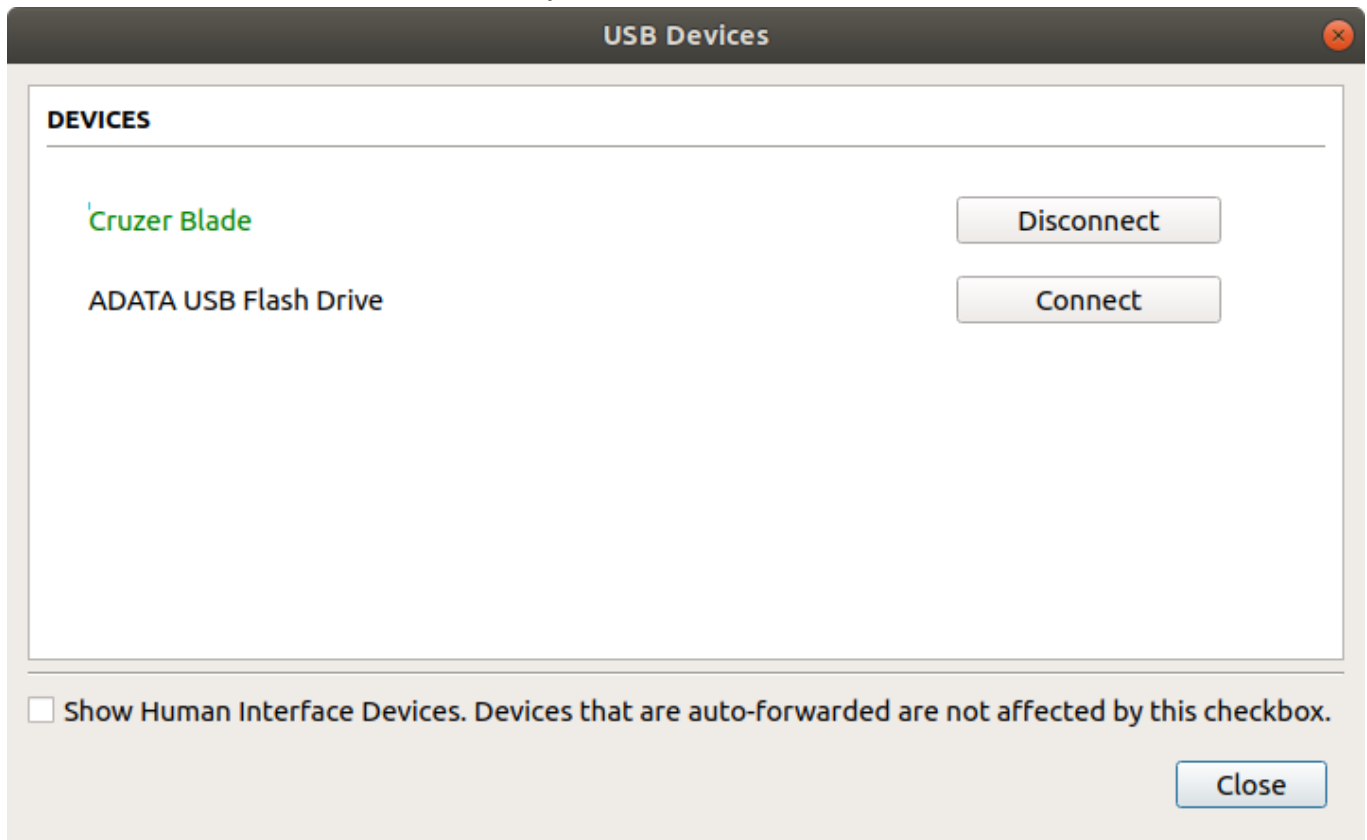
Disconnecting a Session

To disconnect a PCoIP session from either client, select the **Connection > Disconnect** menu option.
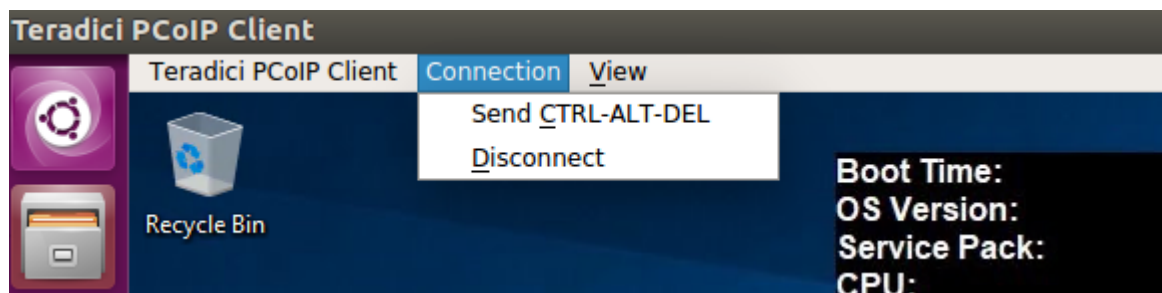
Quitting the application will also disconnect the current session.

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+Alt+Enter** 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**.

> ℹ️ **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.

Miminizing 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+Alt+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+Alt+Enter** while in any full-screen mode.

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.
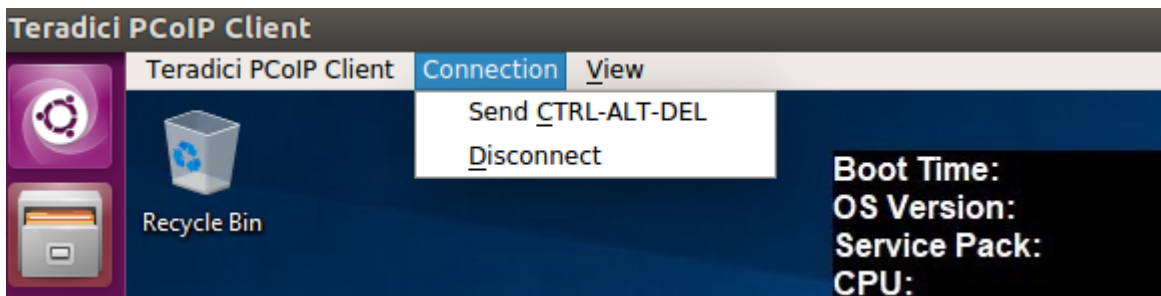
Changing the Language

In addition to English, the PCoIP Software Client also supports a number of different languages. During installation, you can select one of the supported languages.

To change the language in GNOME

 • In the **Settings > Language and Region** panel, select another language from the language setting.

To change the language on the command line

Alternatively, the language can be set with the `locale` parameter on the command line:

```
pcoip-client locale <locale-code>
```

The following table lists the available language translations and codes:

Language	Code
German	de
Spanish	es
French	fr
Italian	it
Japanese	ja
Korean	ko
Portuguese (EU)	pt
Portuguese (Brazil)	pt_BR
Russian	ru
Turkish	tr
Chinese (Simplified)	zh_CN
Chinese (Traditional)	zh_TW

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 20.10, 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 Linux 20.10 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 requirments:

· 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 Linux users**
>
> Support for PCoIP Ultra on the PCoIP Software Client for LInux 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.

Enabling PCoIP Ultra

In order to enable PCoIP Ultra features you need to turn on GPO variables in the PCoIP Agent. For information on enabling this with Linux, see Enabling PCoIP Ultra for Linux. For information on enabling this with Windows, see Enabling PCoIP Ultra for Windows.

Auto-Offload with PCoIP Ultra

The auto-offload feature with PCoIP Ultra enables users to allow PCoIP Ultra to select the best protocol, whether that is CPU or GPU, based on display rate change. CPU Offload is used by default to provide the best image fidelity, GPU Offload is used during periods of high display activity to provide improved frame rates and bandwidth optimization. This setting is only effective if the remote host and client endpoints are capable of both CPU and GPU Offload. You can select this option when you are enabling PCoIP Ultra.

For information on how to do this, see PCoIP Ultra - Windows, and PCoIP Ultra - Linux.

H.264 Hardware Decode

The PCoIP Software Client for Linux supports H.264 hardware decode for selected hardware configurations by using the PCoIP Client to enable the decode. For information on this, see here.

Copyright 2023 HP Development Company, L.P

PCoIP Codec Indicator

When enabling PCoIP Ultra there will be an onscreen indicator at the bottom left corner of the screen.
PCoIP Ultra CPU optimization is indicated with a dark blue dot. PCoIP Ultra GPU optimization is
indicated by a magenta dot. To disable this codec update the `pcoip.code_indicator` parameter:

```
~/.pcoip.rc pcoip.codec_indicator = 0
```

Ensure that you maintain the space before and after the `=` sign.

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

It is possible to redirect USB flash drives with the PCoIP Client SDK for Linux. The following steps outline how to enable support for remoting USB devices:

1. Install **usb_helper** to /usr/libexec/pcoip-client.

2. Grant the nescessary capability to the **usb_helper** by running:

```
setcap "cap_setgid+i" /usr/libexec/pcoip-client/usb-helper
```

3. Grant the nescessary capability to the client binary by running:

```
setcap "cap_setgid+p" /path/to/your/custom/client
```

4. Adjust the nescessary capability within the client binary. The client binary must have **CAP_SETGID** in the Inheritable set. See *capabilities(7)* and *cap_set_proc(3)* for more details. Teradici strongly advises dropping **CAP_SETGID** from the Permitted set and perform the entire conversion at the earliest opportunity within the client application.

> ✏️ **Running Setcap**
>
> setcap must be called after the binary is in its final location. If the binary is changed, rebuilt or moved, then setcap must be re-run before USB functionality will be restored.

Kernel Network Configuration

The `pcoip-configure-kernel0networking.sh` script is installed with the PCoIP Client SDK for Linux. This script tunes the kernel networking configuration to facilitate the network performance required by the Client. You need to run this script after installing the PCoIP Software Client for Linux.

Run the following command:

```
sudo pcoip-configure-kernel-networking --persistent
```

The script contains the following parameters:

- `rmem_default`: A kernel parameter that controls the default size of receive buffers used by sockets.

- `rmem_max`: A kernel parameter that controls the maximum size of receive buffers used by sockets.

> ⚠️ **`rmem_max` size**
>
> This parameter must be at least as large as `rmem_default`/

- `ipv4.udp_mem`: A kernel parameter that controls the maximum total buffer-space to allocate.

- `netdev_max_backlog`: A kernel parameter that controls the maximum size of the receive queue.

HID Local Termination Blacklist

Local Termination of Wacom tablets provides the best user experience in networks with high latency. However, some features of the tablet may not be fully supported with local termination. A HID Local Termination Blacklist has been added to override the preferred local termination mode.

Devices on the blacklist would be bridged to the remote desktop. The local termination settings can be edited in the ini file:

```
localtermination_black_list "vid,pid vid2,pid2"
```

Adding a Custom Logo

The Software Client for Linux can be configured in the Pre-Session mode to place a custom logo beside the PCoIP logo. For best results, the logo should be a .png file with a transparent background, and ideally be of size 245x100 or smaller. The image is named "customlogo.png".

Linux location: `/etc/xdg/Teradici/PCoIPClient/`

Broker Client API

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. The Broker Client API is a two-step procedure, where as a first step, the Broker Client Example is built. The second step consists of running either a prebuilt PCoIP binary, or a prebuilt AWS binary.

> ⚠️ **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 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.

Building the Broker Client Example

> ✏️ **Detailed Instructions**
>
> For more information on building the Broker client example, see Building the Linux SDK.

1. Install the latest PCoIP Software Client. For more information see the topic "Installing the PCoIP Software Client for Linux" in the Administrators Guide.

2. Download the latest Linux SDK.

3. To install the SDK, extract the contents of the `PCoIPSoftClientSDK` folder to a location of your preference.

4. From the PCoIPSoftClientSDK folder, use the `cd` command to change the directory to `broker_client_example`.

5. Create a build directory by running the following command: `mkdir build`

6. Change the directory to `build` by running the following command: `cd build`

7. Configure the SDK example by running the following command: `cmake -G Xcode ..`

8. Build the example by running the following command: `cmake --build . --config RelWithDebInfo`

> ✏️ **Custom Broker Client Library Implementations**
>
> This procedure creates `broker_client_example.app` and `broker_client_example_aws.app` in the `RelWithDebInfo` folder, making the following executables available:
>
> - ./RelWithDebInfo/broker_client_example.app/Contents/MacOS/`broker_client_example`
> - ./RelWithDebInfo/broker_client_example_aws.app/Contents/MacOS/`broker_client_example_aws`
>
> # Running the Prebuilt PCoIP Client Binary

This example uses the supplied credentials to first establish a PCoIP session and then launch the PCoIP Software Client. The PCoIP Software Client thus launched, then connects to the remote agent.

> ✏️ **Location of the login_info File**
>
> The `login_info.txt` file is available in the following location: ~/Documents/ PCoIPSoftClientSDK/Examples/broker_client_example/bin

1. Install the latest PCoIP Software Client. For more information see the topic "Installing the PCoIP Software Client for macOS" in the Administrators Guide.

2. Download the latest MacOS SDK.

3. To install the SDK, open the `.dmg` file, and drag the `PCoIPSoftClientSDK` folder to your **Documents** folder (or another location of your preference).

4. Open the `login_info.txt` file.

5. Add the remote PCoIP agent credentials in the format "FQDN domain username password hostname", where,

   - `FQDN`: The address of an HP Anyware broker.

   - `domain`: The domain that the host machine is on.

   - `username`: The username of the account you are connecting to.

   - `password`: The plain text password of the account you are connecting to. The password cannot contain spaces.

   Example:

   ```
   desktop.example.com adomain.local foouser barpassword bazmacpro01
   ```

1. Launch the example from a Terminal session:

   ```
   cd ~/Documents/PCoIPSoftClientSDK/Examples/broker_client_example/bin/
   broker_client_example.app/Contents/MacOS

   killall PCoIPClient

   ./broker_client_example -l -i ~/Documents/PCoIPSoftClientSDK/Examples/
   broker_client_example/bin/login_info.txt
   ```

Running the Prebuilt AWS Example Binary

This topics contains instructions for running the AWS example binary that is packaged along with the Client SDK. The example uses credentials that you provide to first establish a PCoIP session and then launch the PCoIP Software Client. The PCoIP Software Client thus launched, then connects to the remote agent.

> ✏️ **Location of the login_info_aws File**
>
> The `login_info_aws.txt` file can be found in the following location: ~/Documents/ PCoIPSoftClientSDK/Examples/broker_client_example/bin

1. Install the latest PCoIP Software Client.. For more information see topic "Installing the PCoIP Software Client for macOS" in the Administrators Guide.

2. Download the latest MacOS SDK.

3. To install the SDK, open the `.dmg` file, and drag the `PCoIPSoftClientSDK` folder to your Documents folder (or another location of your preference).

4. Open the `login_info_aws.txt` file.

5. Add the remote Amazon WorkSpace credentials in the format "aws_registration_code domain username password token_or_password resource_name", where,

   • `aws_registration_code`: The registration code for Amazon WorkSpace.

   • `domain`: The domain that the host machine is on.

   • `username`: The username of the account you are connecting to.

   • `password`: The plain text password of the account you are connecting to. The password cannot contain spaces.

   • `token_or_password`: The authentication token or password.

   • `resource_name`: The name of the resource to connect to.

Example:

```
 eNgrTX$xa adomain.local foouser barpassword
755392850753714691480827201091130414341633969745587360082822514614949064198027266
aws57864x1
```

1. Launch the example from a Terminal session:

```
    cd ~/Documents/PCoIPSoftClientSDK/Examples/broker_client_example/bin/
broker_client_example_aws.app/Contents/MacOS

    killall PCoIPClient

    ./broker_client_example_aws -l -i ~/Documents/PCoIPSoftClientSDK/
Examples/broker_client_example/bin/login_info_aws.txt
```

Using the Broker Client API 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 API example demonstrates the success path for establishing new PCoIP sessions.

> ⚠ **Do not Use the Broker Client API 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:

usr/bin/

There are several files in this directory, but only two are relevant for the broker client example:

- `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.

> ℹ **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):
>
> ```
> my-machine.mydomain.com mydomain myuser "mypassword" desktop-resource
> ```

In the example above:

- The **FQDN** of the host server is `my-machine.mydomain.com`

- The **domain** is `mydomain`

- The **user** is `myuser`

- The **password** is `mypassword`

- The **host name** is `desktop-resource`

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 command prompt and type:

   ```
   broker_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"
   ```

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
```

The Collaboration Broker Client API Example

The included sample Collaboration Broker Client demonstrates how the APIs can be used to customize and control the pre-session and session phases of the Collaboration connection. The Collaboration Broker Client API is a two-step procedure, where as a first step, the Collaboration Broker Client Example is built. The second step consists of running a prebuilt Collaboration binary.

✏️ **More About the Collaboration Broker Client**

The Collaboration Broker Client Example extends on the [Broker Client Example](#) demonstrate how the Broker Client can join a collaboration session.

⚠️ **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.

Building the Broker Client Example

> ✏️ **Detailed Instructions**
>
> For more information on building the Broker client example, see Building the Linux SDK.

1. Install the latest PCoIP Software Client. For more information see the topic "Installing the PCoIP Software Client for Linux" in the Administrators Guide.

2. Download the latest Linux SDK.

3. To install the SDK, extract the contents of the `PCoIPSoftClientSDK` folder to a location of your preference.

4. From the PCoIPSoftClientSDK folder, use the `cd` command to change the directory to `broker_client_example`.

5. Create a build directory by running the following command: `mkdir build`

6. Change the directory to `build` by running the following command: `cd build`

7. Configure the SDK example by running the following command: `cmake -G Xcode ..`

8. Build the example by running the following command: `cmake --build . --config RelWithDebInfo`

> ✏️ **Custom Broker Client Library Implementations**
>
> This procedure creates `broker_client_example.app` and `broker_client_example_aws.app` in the `RelWithDebInfo` folder, making the following executables available:
>
> - ./RelWithDebInfo/broker_client_example.app/Contents/MacOS/`broker_client_example`
> - ./RelWithDebInfo/broker_client_example_aws.app/Contents/MacOS/
>   `broker_client_example_aws`

Running the Prebuilt Collaboration Example Binary

This topics contains instructions for running the Collaboration example binary that is packaged along with the Client SDK. The example uses credentials provided by the primary session owner to first establish a PCoIP session and then launch the PCoIP Software Client as a collaborator. The PCoIP Software Client thus launched, then connects to the remote agent.

> ✏️ **Location of the login_info_collaborator File**
>
> The `login_info_collaborator.txt` file is available in the following location: /PCoIPSoftClientSDK/Examples/broker_client_example/bin

1. From the broker_client_example folder, open the `login_info_collaborator.txt` file.

2. Add the collaborator session details in the format "invite_URI collaborator_name invite_code", where,

   - `invite_uri`: The collaboration invitation URI provided by the primary session owner.
   - `collaborator_name`: The name of the collaborator.
   - `invite_code`: The invitation code provided by the primary session owner.

   Example:

   ```
    pcoip://10.11.12.13/connect?
   data=eyJhbGciOiJIUzI1NiJ9.eyJSb2xlIjoiQWRtaW4iLCJJc3N1ZXIiOiJJc3N1ZXIiLCJVc2Vybm
   collaborator 123435
   ```

1. Launch the example from a Terminal session:

   ```
       cd ~/Documents/PCoIPSoftClientSDK/Examples/broker_client_example/bin/
   broker_client_example_collaborator.app/Contents/MacOS

       killall PCoIPClient

       ./broker_client_example_collaborator -l -i ~/Documents/
   PCoIPSoftClientSDK/Examples/broker_client_example/bin/
   login_info_collaborator.txt
   ```

Using the Collaboration Broker Client Example

The SDK provides a sample command line pre-session client called the Collaboration Broker Client Example, which calls the included broker client libraries and enables you to join a PCoIP collaboration session. The Collaboration Broker Client API Example demonstrates the success path for joining a PCoIP collaboration session. Collaborators can join PCoIP sessions after they receive invitation links and codes from session owners. Session owners generate invitation links and codes on remote PCoIP agent machines.

> ⚠ **Do not Use the Broker Client Example in Production**
>
> The Collaboration 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 Collaboration Broker Client API is located here: /PCoIPSoftClientSDK/Examples/ broker_client_example/bin. The following Collaboration Broker Client files are available at this location:

- broker_client_example_collaborator.exe, which is the sample command-line client

- login_info_collaborator.txt, which ontains authentication information used by the client

About the login_info_collaborator.txt File

The Collaborator Broker Client API example uses a local text file to provide session input values. The following is a sample login_info_collaborator.txt file:

```
pcoip://10.11.12.13/connect?
data=eyJhbGciOiJIUzI1NiJ9.eyJSb2xlIjoiQWRtaW4iLCJJc3N1ZXIiOiJJc3N1ZXIiLCJVc2Vybm
collaborator 123435
```

In this example:

- The invitation URI is pcoip://10.11.12.13/connect?
data=eyJhbGciOiJIUzI1NiJ9.eyJSb2xlIjoiQWRtaW4iLCJJc3N1ZXIiOiJJc3N1ZXIiLCJVc2VybmFtZSI6Ikphdm

- The Collaborator Name is `collaborator`.

- The Invite Code is 123435.

> ✏️ **Collaboration Broker Client Remote Sessions**
>
> The remote sessions established by `broker_client_example_collaborator.exe` are the same as collaboration sessions established using the Anyware Software Client, except that the input values are provided by the login_info_collaborator.txt file instead of the PCoIP Client.

Initiating the Collaboration Broker Connection Workflow

Initiating the Collaboration Broker Connection workflow is a two-step process. As a first step, modify the login_info_collaborator.txt file to add the URI of the host, the collaborator's name, and the invitation code. Then, run `broker_client_example_collaborator.exe` using `login_info_collaborator.txt` as an argument. To do this, use the -l switch to have the Collaboration Broker Client API example invoke the collaborator session client.

> ✏ **More About the -l Switch**
>
> The -l switch enables you to invoke the Session Client API without the 60-second connect tag window.

Modifying the login_info_collaborator.txt File

1. Open the login_info_collaborator.txt file in a text editor.

2. Add the following information, in this sequence, separated by spaces:

   • The URI of the host server

   • The Collaborator Name

   • The Invite Code

3. Save the text file.

Launching the Collaborator Session Client

1. Open Command Prompt.

2. Use the `cd` command to change the directory to `bin`.

3. Run the following command, providing the login_info_collaborator.txt file as an argument:

```
./broker_client_example_collaborator -l -i login_info_collaborator.txt
```

Session Client API

Session Client API Change Log

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

20.10 - No updates or changes.

20.07 - No updates or changes.

20.04 - No updates or changes.

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 `pcoip-client`:

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

・**Connection tag:**

```
SCS1WsopFJ3iz1l48PTJMXFkcD4b6M9aiakHXH3ellLhUROBceWAifSSn%2b4AV1FC
8IihWVmsISmYFKeA25AtzFrdMpdaCtqlic0zfxAA
```

・**IP address:** `10.64.60.115`

・**Log ID:** `a1ff3a80-8801-1038-19bd-0005680aded`

・**Session ID:** `2305843009213693961`

```
pcoip-client -i connect-tag=
SCS1WsopFJ3iz1l48PTJMXFkcD4b6M9aiakHXH3ellLhUROBceWAifSSn%2b4AV1FC
8IihWVmsISmYFKeA25AtzFrdMpdaCtqlic0zfxAA address=10.64.60.115
session-id=2305843009213693961
log-id=a1ff3a80-8801-1038-a9bd-00505680aded
```

Encode all information into a string container (URI) and then pass to the executable. The following command invokes pcoip-client 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:**

```
SCS1WsopFJ3iz1l48PTJMXFkcD4b6M9aiakHXH3ellLhUROBceWAifSSn%2b4AV1FC
8IihWVmsISmYFKeA25AtzFrdMpdaCtqlic0zfxAA
```

・**URI:** "teradici-pcoip://10.64.60.115:4172?session-id=230584300921369396"

> ✏️ **URI Format Documentation**
>
> There is a document describing the URI format in the root of the SDK.

```
'pcoip-client "connect-
tag=SCS1WsopFJ3iz1l48PTJMXFkcD4b6M9aiakHXH3ellLhUROBceWAifSSn%2b4AV1FC8IihWVmsIS
"teradici-pcoip://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 at **/usr/lib/x86_64-linux-gnu/pcoip-client/examples/ minimal_client/main.cpp**:

```
pcoip_client_sdk_session/modules/broker_client_example/src/broker_
client_example_main.c
```

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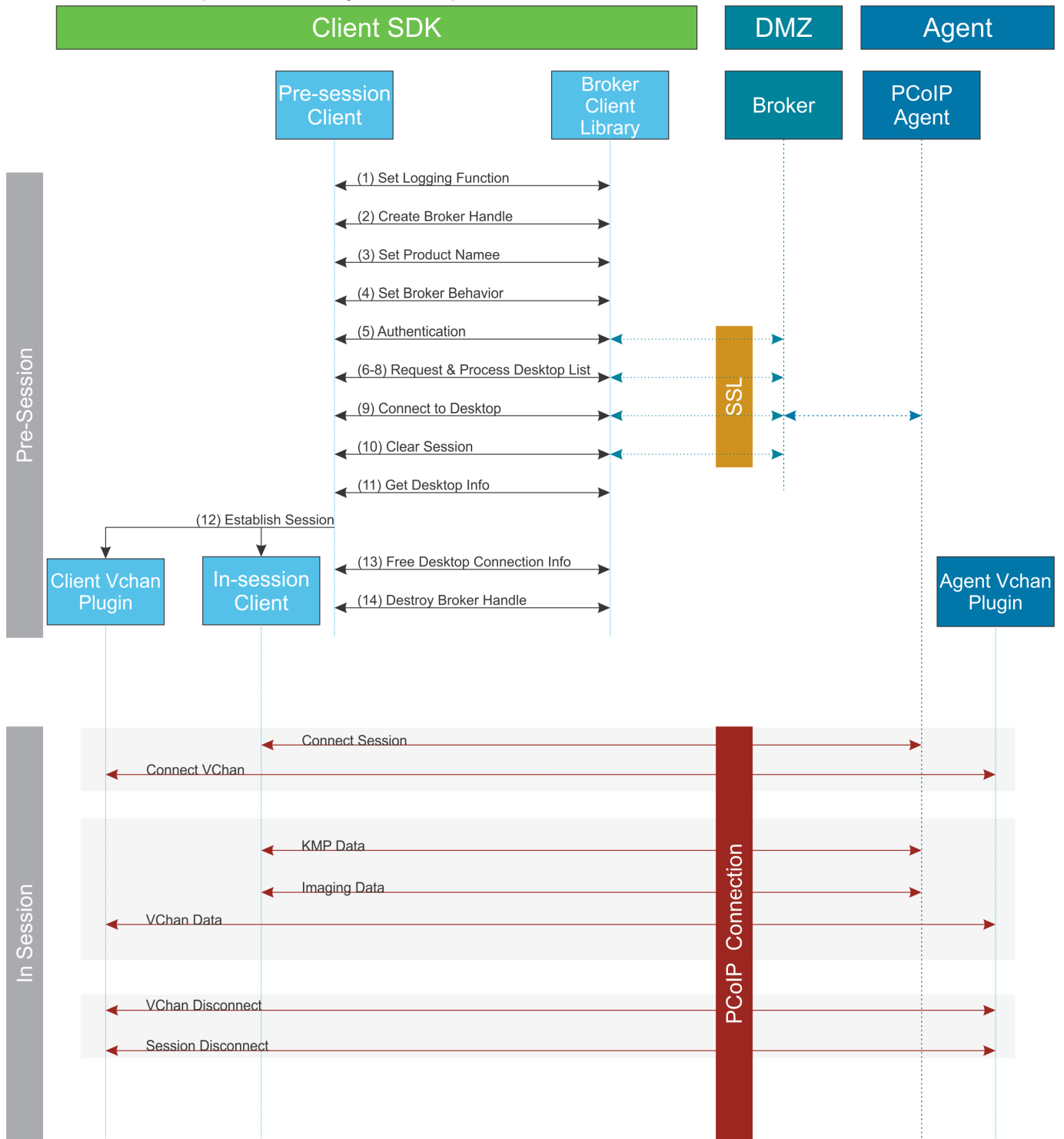
Each of these steps is used in the sample code, with a comment identifying the step number.

✏️ **Custom Log Implementations**

You can design and implement your own logging functionality, so along 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 `pcoip-client`, 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:
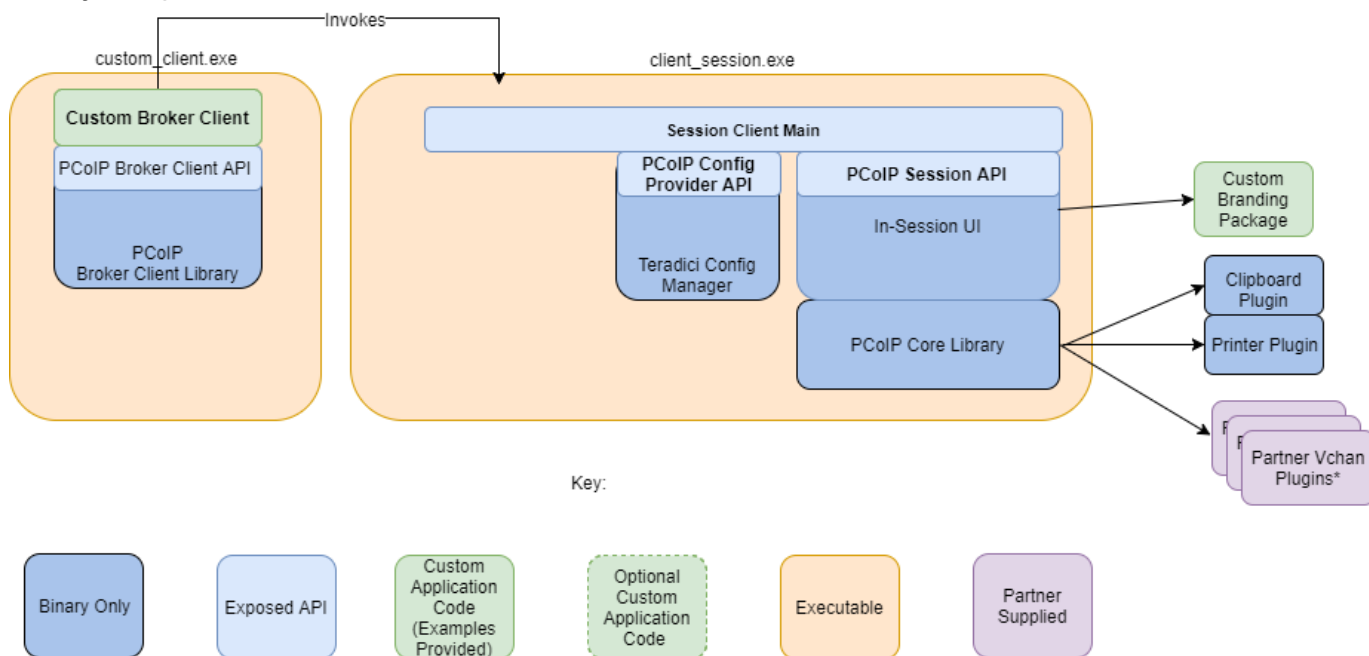
Session Client Binary Integration

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 yout workflow and needs using the `broker_client_example` source code as a starting point. For more information see [Session Client API Integration](#).
- Use the `pcoip-client` executable, which must be [installed separately](#), to establish the connection.

> ⚠️ **Security Considerations**
>
> The values passed to the `pcoip-client` 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 `pcoip-client` 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 `pcoip-client` 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 `pcoip-client` 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 `pcoip-client` executable beyond that which is possible via its command line interface, or if you need to integrate the pre-session and `pcoip-client` 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 `pcoip-client` 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
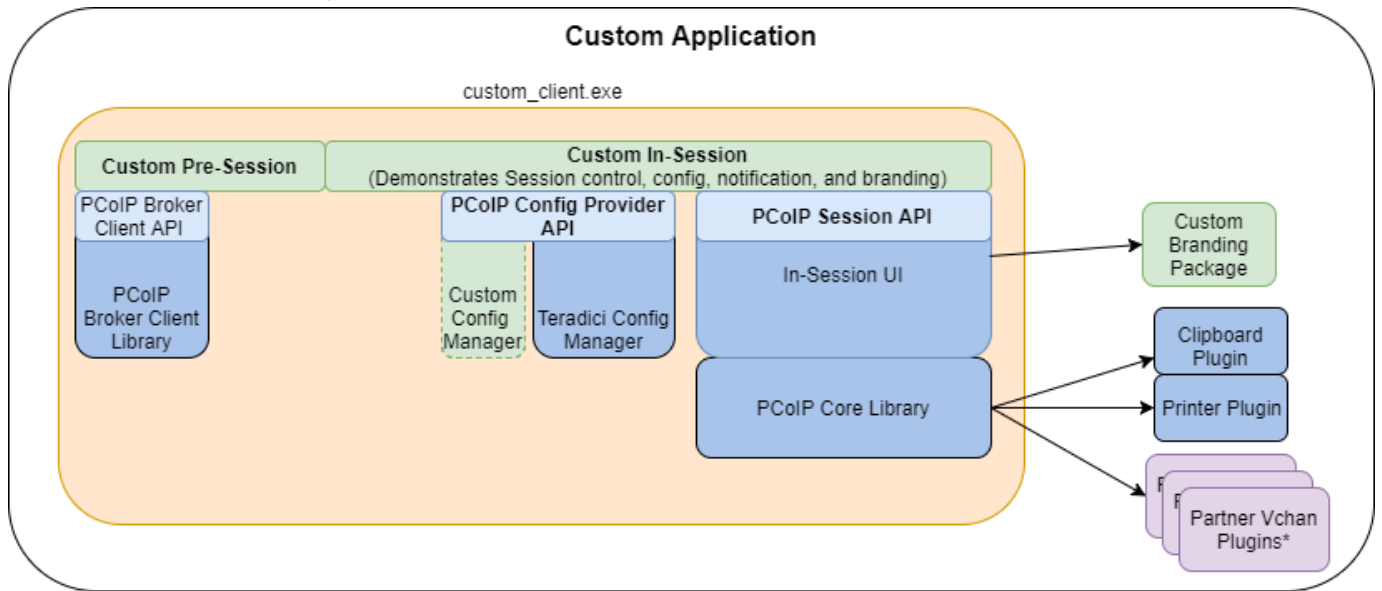
*Partner Virtual Channel Plugins can be developed using the Virtual Channel SDK to enable this you can combine the PCoIP Client SDK with the PCoIP Virtual Channel SDK. For more information, see the [Virtual Channel SDK](#).

Passing Customization Parameters to the Session Client

When using the `-l` 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 -i login_info.txt -l "disable-menubar"
```

Using the Session Client API

Use of the Session Client C++ API is demonstrated in **/usr/lib/x86_64-linux-gnu/pcoip-client/ examples/pcoip-client_example/pcoip-client_main.cpp**. It includes the steps outlined in the diagram below:
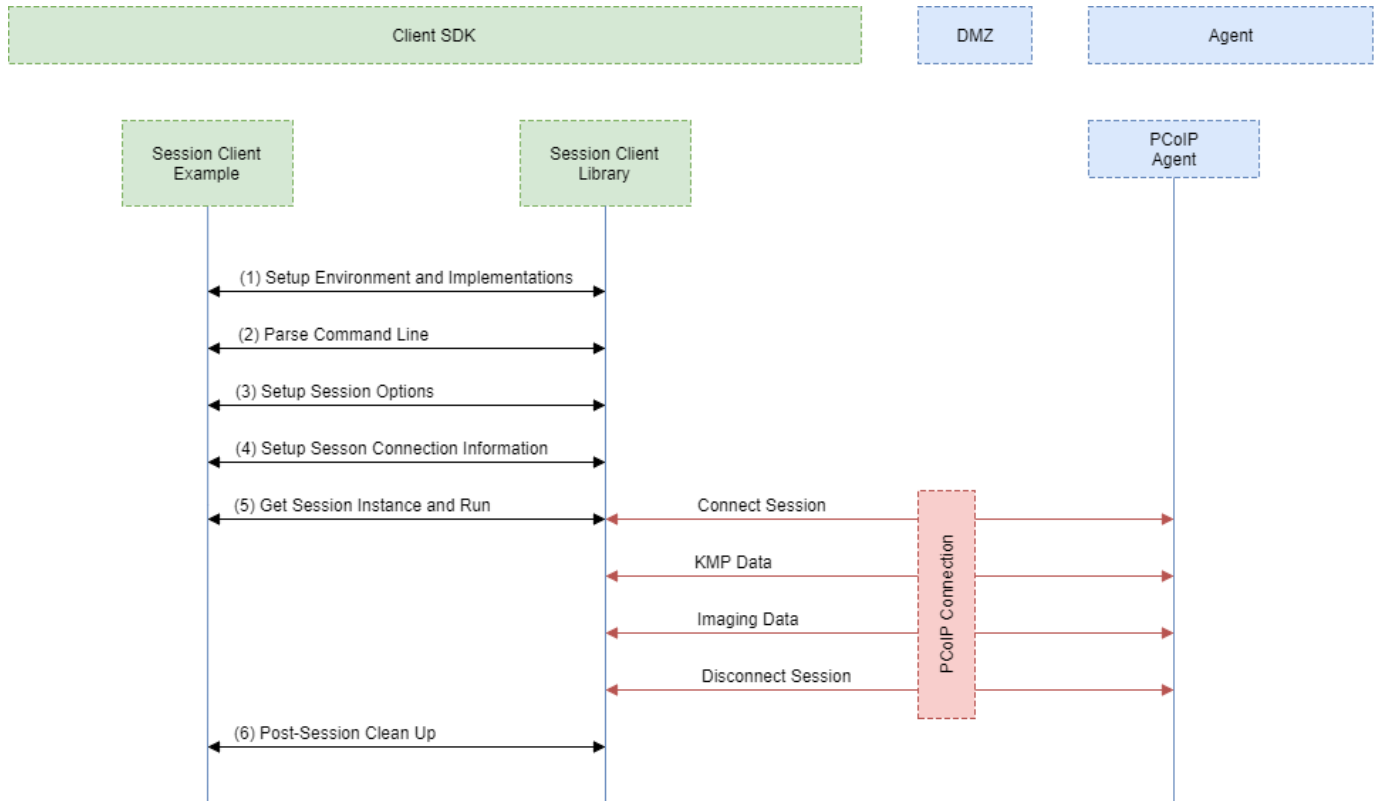
PCoIP Session Client API Sequence Diagram

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

1. **Setup Environment and Implementations:** Os specifc 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
`<app-name>`		**Required!** The name of your custom session client. This will be used as the application window file.
`<app-icon>`		**Required!** The file name of your application icon. In Windows, this appears in the Windows toolbar and the window header.
`<toolbar-menu>`		**Required!** Describes the text labels used in OS toolbar menu's.
	`<about-item>`	The text label for the *About...* menu item. Optional. Without this field, there will not be an *About* menu item.
	`<quit-item>`	The text label for the *Quit...* menu item. Optional. Without this field, you will not have a *Quit* menu item.
`<about>`		**Required!** Describes the contents of the *About...* dialog. Must have the following required attributes: **title** (*string*): The dialog text and **minWidth** (*number*): The mimumum pixel width of the dialog. For example: `<about title="My Custom Client" minWidth="100">`
	`<line>`	Describes a line of text in the dialog. All lines are optional, but the *About* window will be empty unless you provide at least one. `<line>` accepts the following attributes: **align** (*string keyword*): 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 (). `<line>` can contain the following elements: `<logo>` contains a filename for a logo or other graphic element: `<logo>about_logo.png</logo>`. `<text>` contains the display text for each line: `<text>This text is displayed on the line.</text>`. `<hyperlink>` takes a url parameter and creates a working hyperlink: `<hyperlink url="www.teradici.com">Teradici</hyperlink>`.

A full text layout file looks like this:

```xml
<?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:

- **Linux clients: 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 -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:cbc3fd3c6d001a1e1f06342bcccf2a62bd748c3cf1dd2e4c9c29561ea07bd089
```

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

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):

   ```
   pcoip-client -branding-package my_custom_
   branding.bp
   -branding-hash
   cbc3fd3c6d001a1e1f06342bcccf2a62bd748c3cf1dd2e4c9c29561ea07bd089
   <other-params>
   ```

Core API

Core API Change Log

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

20.10

PcoipClientInitData

- PcoipClientInitData structure has been changed for licensing related data. The following fields have been added:
  - proxyServer
  - proxyPort
  - numberOfActivationCodes
  - activationCodes

pcoip_core_api.h

- pcoip_core_api.h structure change: *struct LicenseInfo* has changed to the following:

```
struct LicenseInfo
{
    enum PCoIPLicensingResult resultCode;   ///< the result code as reported
by the licensing module
    bool hasExpiry;  ///< license has a valid expiry time
    time_t expiryTime;   ///< time that license expires expressed as UNIX time
};
```

The change to this struct affects the pcoip_client_get_licensing_info() API function. Replaced the int32_t daysRemaining with time_t expiryTime and added field *bool hasExpiry*.

- New api: pcoip_client_log_level_get(). Returns the current event level defines as PCOIP_EVENT_LEVEL_* constants.

20.07

pcoip_core_api.h

- PcoipClientinitData structure change: *trustCenterUrl* renamed to *licServerURL*.

- New api: pcoip_core_aws_allowed(). Returns whether connections to Amazon WorkSpaces are allowed.

- New api: pcoip_core_api_vmware_allowed(). Returns whether connections to VMware Horizon are allowed.

- New api: pcoip_core_api_pcoip_allowed(). Returns whether PCoIP connections are allowed.

20.04

pcoip_core_api.h

- pcoip_client_init() signature change: Uses new *PcoipClientInitData* structure. It includes data needed for licensing in future releases. The new data members should be left empty. Unused parameters were removed.

- Deprecated api: pcoip_client_connection_callback_register()

- New api: pcoip_client_connection_callback_register(). Callback signature changed to include PCoIPAuth state for licensing in future releases.

- New api: pcoip_client_usb_set_local_termination_blacklist(). Sets devices that should be locally terminated.

pcoip_disconnect_types.h

- Added value to enum ePCOIP_DISCONNECT_CAUSE. Will be used for licensing in future releases.

certificate_store.h

- certificate_store set_store_location() removed. Location can now only be set during construction of certificate_store.

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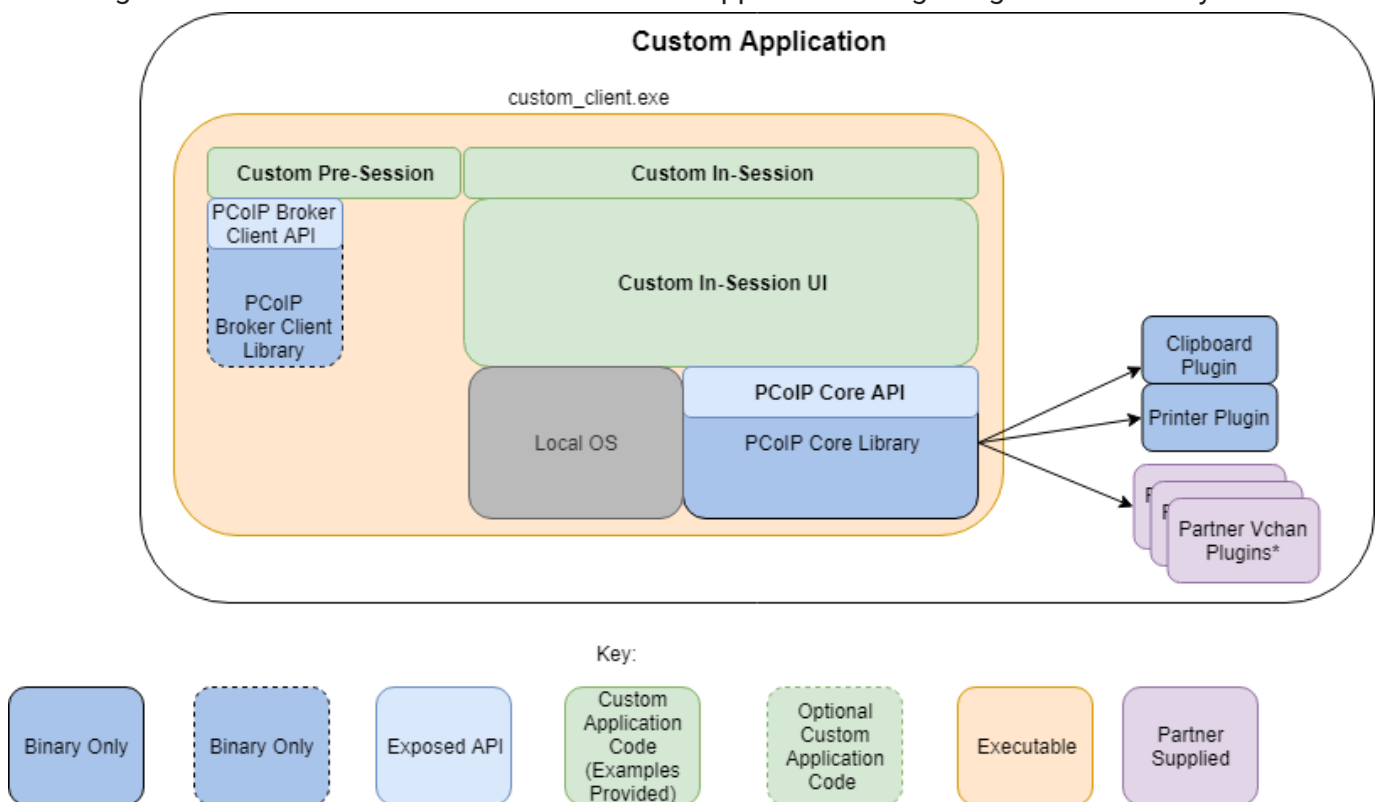
*Partner Virtual Channel Plugins can be developed using the Virtual Channel SDK, to enable this you can combine the PCoIP Client SDK with the PCoIP Virtual Channel SDK.

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:

Refer to the SDK package for the dependencies and libraries required to set-up the development environment. The CMake modules provided automatically detects the system libraries at CMake configuration 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 Linux section for instructions on building your applications using CMake.

1. Install the .deb SDK installer. All nescessary files will be saved to the corresponding system directories.
2. At this point, the SDK is installed on the system. For build instructions, see Building the SDK for Linux.

Updating SDK Components

Updating the SDK to a new version can be done by installing a new version of the SDK. There is no special upgrade path.

Building the SDK for Linux

Building the SDK for Linux

This section includes instructions for building a custom client application by utilizing the SDK.

1. As a reference, follow the **CMakeLists.txt** of the `minimal_client` to understand the build steps using the cmake tool. It is located at `/usr/lib/x86_64-linux-gnu/pcoip-client/examples/minimal_client/CMakeLists.txt`.

2. Run the CMake configure and verify it configures without any errors.

3. Run `make` to build the application.

4. The `find_package()` command of CMake automatically finds the libraries installed by the SDK.

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 HP Anyware 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 HP Anyware 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:
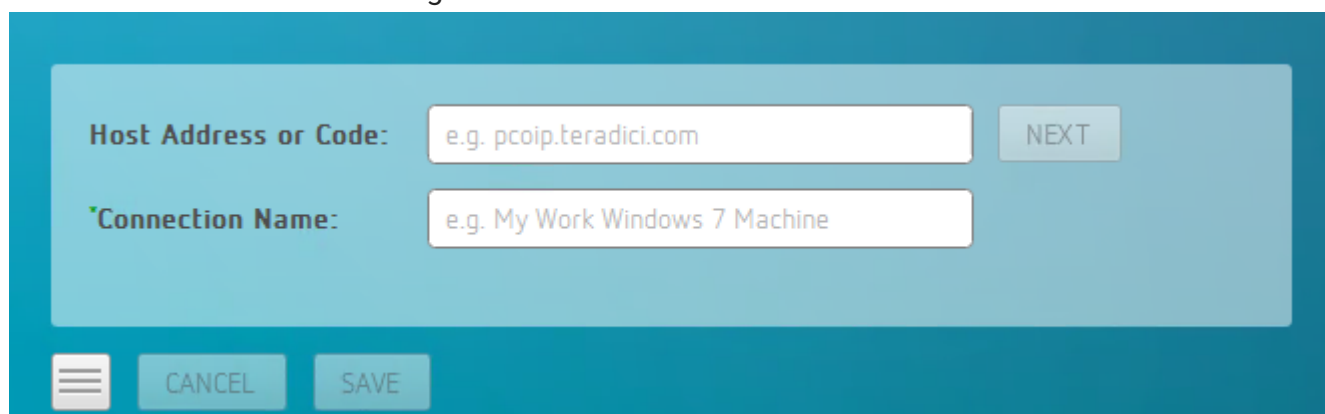
- Teradici PCoIP Software Client for Linux

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:
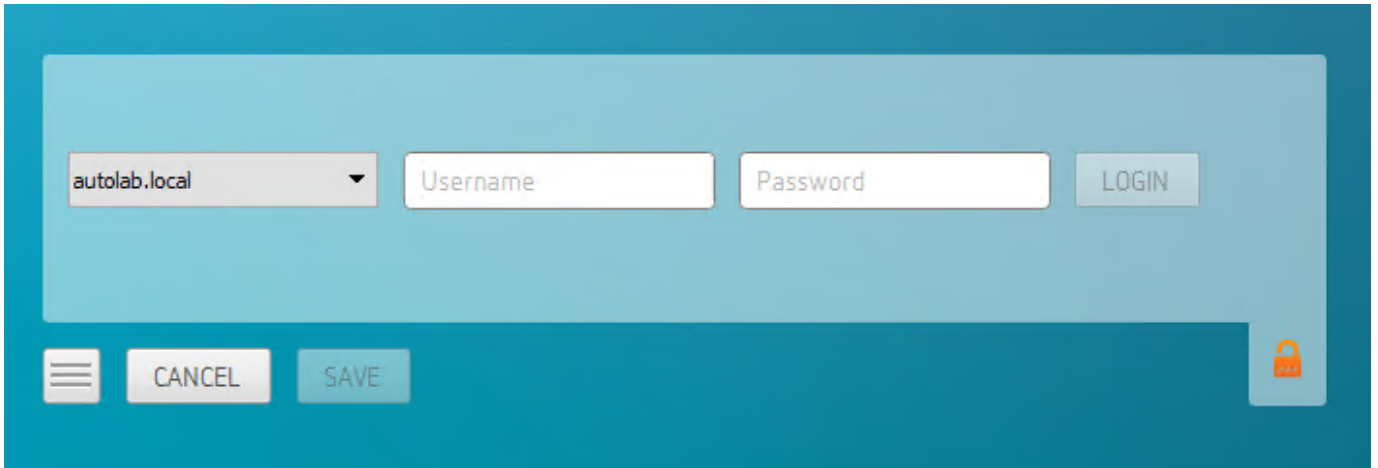
Pre-Session Connection

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

Extensions

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:

```
/usr/lib/x86_64-linux-gnu/pcoip-client/examples/support-bundler/pcoip-client-
support-bundler
```

The file will be created and placed in the `/tmp` directory.

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 `pcoip-client`.

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.