

PCoIP Client SDK for Windows Developers' Guide

23.06

Table of Contents

PCoIP Client SDK for Windows 23.06	7
Supported Platforms	7
What Can You Build With the PCoIP Client SDK?	8
Who Should Read This Guide?	9
Developing with the Session API and Core API	9
What's New in This Release?	10
High Performance Client Mode is Now Default	10
Collaboration Manager Opens from In-Session Menu	10
New Anyware Modes Information Dialog	10
Bloomberg Keyboard Support	11
System Requirements	12
Hardware System Requirements	12
About the PCoIP Client SDK for Windows	13
SDK Header Files	14
PCoIP Sessions	18
About PCoIP Sessions	18
System Actors	18
PCoIP Session Phases	20
About Brokered and Non-Brokered Connections	20
PCoIP Session Phases	21
About Brokered and Non-Brokered Connections	22
Feature Support	23
Wacom Tablet Support	23
Locally Terminated Wacom Tablets	23
Bridged Wacom Tablets	24
Known Issues with Wacom Tablets and Functionality	26

Webcam Support	27
Requirements	27
Notes and Limitations	27
Setup	27
Connecting to a USB Device	28
Calling USB Functions	31
Signing the Client Application	31
Enabling the Signed Application to be trusted by the USB Service	31
Disconnecting a Session	33
Session Reconnection	33
Changing the PCoIP Software Client Window Mode	34
Activating Full Screen Modes	34
Miminizing the PCoIP Software Client from a Full-screen Mode	35
Activating Windowed Mode	35
Enhanced Audio and Video Synchronization	36
Sending a Ctrl-Alt-Del Command	37
Changing the Language	38
PCoIP Ultra	39
PCoIP Ultra Enhancements	39
Requirements	40
Enabling PCoIP Ultra	40
Customizable Session Features	42
Disable Session Menu Bar Visibility	43
Disable Hot Keys	43
Windowed or Fullscreen Mode	43
Set Host Resolution	43
Custom Client Branding	43
Image Scaling	44
Maintain Aspect Ratio	44

USB Auto-Forward	45
USB Vendor ID/Product ID Auto-Forward	45
Disable USB	45
Locale	45
Session Log-ID	46
Log Level	47
Log Folder	47
Log Prefix	47
Force Native Resolution	47
Branding Your Session Client	48
Supporting USB Devices	49
Configuring Wacom Tablets	50
System Precedence	54
HID Local Termination Blacklist	55
Session API	56
How to Establish a PCoIP Session	56
PCoIP Session-Creation Steps and Actors	58
Session Client Integration	63
 Session Client Binary Integration	63
 Session Client API Integration	64
Passing Customization Parameters to the Session Client	66
Using the Client Session API	67
 Applications and Files Required	67
Creating a Branding Text Layout File	69
Creating a Branding Package	72
Using the Branding Package	73
Limits on Customization	74
 Windows Limitations	74

Broker API	75
The Broker Client API Example	75
The Broker Client API Example Sequence	75
Building the Broker Client Example	76
Running the Prebuilt PCoIP Client Binary	76
Running the Prebuilt AWS Example Binary	78
Using the Broker Client API Example	80
Initiate Broker Connection Flow	82
Launching the Session Client from Broker Client API Example	83
The Collaboration Broker Client Example	84
Building the Broker Client Example	85
Running the Prebuilt Collaboration Example Binary	86
Using the Collaboration Broker Client Example	87
About the login_info_collaborator.txt File	87
Initiating the Collaboration Broker Connection Workflow	89
Modifying the login_info_collaborator.txt File	89
Launching the Collaborator Session Client	89
Building the SDK for Windows	90
Configuring CMake for PDB Files	90
Windows Build Procedure	91
Setting Up a PCoIP Agent Test Environment	92
Connecting To Your PCoIP Agent	93
Establishing a PCoIP Connection Using a Teradici PCoIP Software Client	93
Extensions	95
PCoIP Support Bundler Tool	95
Manually Bridging USB Devices	96
Updating the Client USB Package	97
Exit Codes for Programmatic USB Installations and Uninstallations	98

Frequently Asked Questions	99
Frequently Asked Questions	99

PCoIP Client SDK for Windows 23.06

The Client SDK for Windows 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 Windows 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 Windows can be used on the following operating systems:

- Windows 10 (64-bit)
- Windows 10 IoT 64 21H2 Enterprise LTSC
- Windows 11

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 Client SDK for Windows introduces the following:

High Performance Client Mode is Now Default

The Client SDK for Windows's *high performance client mode* is now enabled by default, and recommended for most users. Previously, this mode was available as a technology preview. The previous standard mode can still be enabled by switching to *Standard Performance Mode*, if desired. For more information, see [Client Modes](#) in the Software Client for Windows guide.

Collaboration Manager Opens from In-Session Menu

The Collaboration Manager can now be launched from a new *Collaboration* menu option in the Client SDK for Windows's menu, in addition to launching from the system menu bar. For more information, see [Sharing Your Session With Collaborators](#) in the Software Client for Windows guide.

New Anyware Modes Information Dialog

Information related to PCoIP Ultra acceleration modes is now available via the [Connection Health Indicator](#) on the Client SDK for Windows interface. A tooltip has been added next to **Mode** on the **Anyware Health** window, which opens the **Anyware Modes** dialog containing a brief description on the PCoIP Ultra acceleration modes.

PCoIP Ultra modes are configured on the PCoIP agent. For more information, see [PCoIP Ultra Modes](#) in the Software Client for Windows guide.

Bloomberg Keyboard Support

Bloomberg Keyboards 4 and 5 are now supported when connecting a Client SDK for Windows to PCoIP Agents for Windows. For more information, see [Bloomberg Keyboard Support](#) in the Software Client for Windows guide.

System Requirements

The following table outlines the system requirements for the PCoIP Software Client for Windows:

System	Version Required
PCoIP Software Client Operating Systems	<ul style="list-style-type: none">• Windows 10 (64-bit)• Windows 10 IoT 64 21H2 Enterprise LTSC• Windows 11
Compatible PCoIP Agents	The Client SDK for Windows can connect to any PCoIP agent. Some features require specific agent versions; see the <i>Feature Support</i> 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 Windows

The Client SDK for Windows does not ship with its own session client binary; it uses the standard [PCoIP Software Client for Windows](#) 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 Windows

Prior to the 23.01 release, the Client SDK for Windows shipped with an included session client binary. If you are coming from a previous release, you should install the [PCoIP Software Client for Windows](#), and update your code to use the new file location.

Once installed, the binary can be found here:

```
%PROGRAMFILES(X86)%\Teradici\PCoIP Client\bin\pcoip_client.exe
```

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

```
<working directory>\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 Windows Administrators' Guide](#).

For 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, see [Minimal Client Example](#).

SDK Header Files

The following table contains a brief description of the header files included in the PCoIP Client SDK for Windows:

!!! note Note: Location of the Header Files The header files are available at the following locations in the PCoIP Client SDK for Windows package: - Windows and Linux: The **include** folder - macOS: The **Frameworks** folder

Header File	Description
broker_client.h	This file defines the APIs supported by the Broker client SDK. It is the only file that needs to be included for using the Broker Client SDK.
broker_client_export.h	This file is automatically generated to facilitate using the SDK DLLs in a Windows environment.
broker_client_types.h	This file contains type definitions for <code>broker_client</code> .
pcoip_avsyncmode.h	This file defines the <code>AVSyncmode</code> APIs supported by the PCoIP protocol shared library or the PCoIP protocol static library for interfacing with external implementations.
pcoip_client.h	This file defines the APIs supported by the PCoIP Client SDK. It is the only file that needs to be included for using the PCoIP Client SDK.
pcoip_client_audio.h	This file defines the audio APIs supported by the PCoIP protocol shared library or the PCoIP protocol static library for interfacing with external implementations.
pcoip_client_collaboration.h	This file defines the collaboration APIs supported by the PCoIP protocol shared library or the PCoIP protocol static library for interfacing with external implementations.
pcoip_client_config.h	This file defines the configuration APIs supported by the PCoIP protocol shared library or the PCoIP protocol static library for interfacing with external implementations.
pcoip_client_cursor.h	This file defines the cursor APIs supported by the PCoIP protocol shared library or the PCoIP protocol static library for interfacing with external implementations.
pcoip_client_display.h	This file defines the display APIs supported by the PCoIP protocol shared library or the PCoIP protocol static library for interfacing with external implementations.
pcoip_client_error.h	This file defines the error enumeration used in the SDK.
pcoip_client_keyboard.h	This file defines the keyboard APIs supported by the PCoIP protocol shared library or the PCoIP protocol static library for interfacing with external implementations.
pcoip_client_licensing.h	This file defines the licensing APIs supported by the PCoIP protocol shared library or the PCoIP protocol static library for interfacing with external implementations.
pcoip_client_logging.h	This file defines the logging APIs supported by the PCoIP protocol shared library or the PCoIP protocol static library for interfacing with external implementations.
pcoip_client_mouse.h	This file defines the mouse APIs supported by the PCoIP protocol shared library or the PCoIP protocol static library for interfacing with external implementations.
pcoip_client_performance.h	This file defines the performance APIs supported by the PCoIP protocol shared library or the PCoIP protocol static library for interfacing with external implementations.

Header File	Description
pcoip_client_session.h	This file defines the session APIs supported by the PCoIP protocol shared library or the PCoIP protocol static library for interfacing with external implementations.
pcoip_client_touch.h	This file defines the touch APIs supported by the PCoIP protocol shared library or the PCoIP protocol static library for interfacing with external implementations.
pcoip_client_types.h	This purpose of this header file is to only support backward compatibility. It should not be used.
pcoip_client_usb.h	This file defines the USB APIs supported by the PCoIP protocol shared library or the PCoIP protocol static library for interfacing with external implementations.
pcoip_core_api.h	This purpose of this header file is to only support backward compatibility. It should not be used.
pcoip_core_export_header.h	This file is automatically generated to facilitate using the SDK DLLs in a Windows environment.
pcoip_core_types.h	This purpose of this header file is to only support backward compatibility. It should not be used.

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:

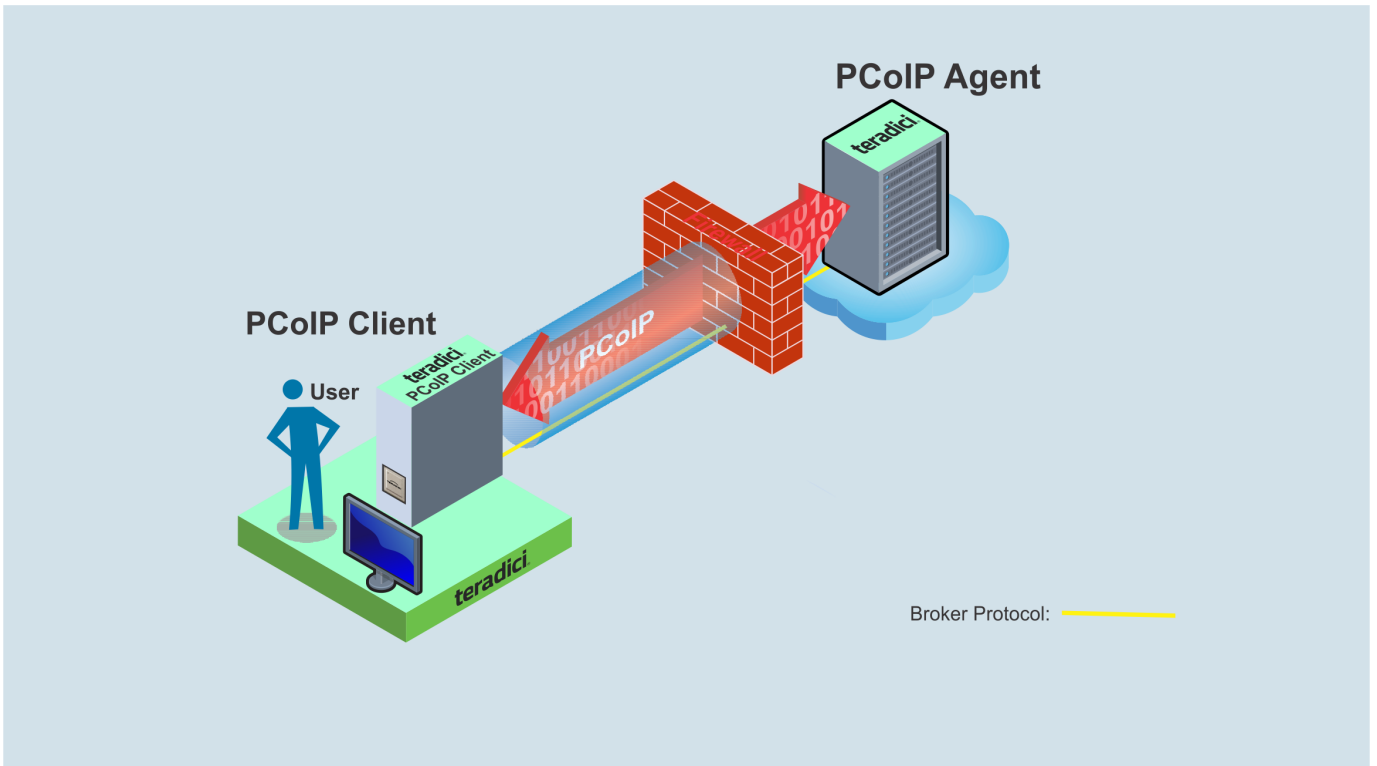
- **PCoIP Client:** The hardware or software device, local to the user, which requests and drives the PCoIP session by negotiating with PCoIP brokers and PCoIP agents.
- **PCoIP Broker:** Brokers maintain lists of active users, their authentication information, and the host machines they can connect to. Except for systems using direct connections between hosts and clients, all PCoIP sessions are negotiated via 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

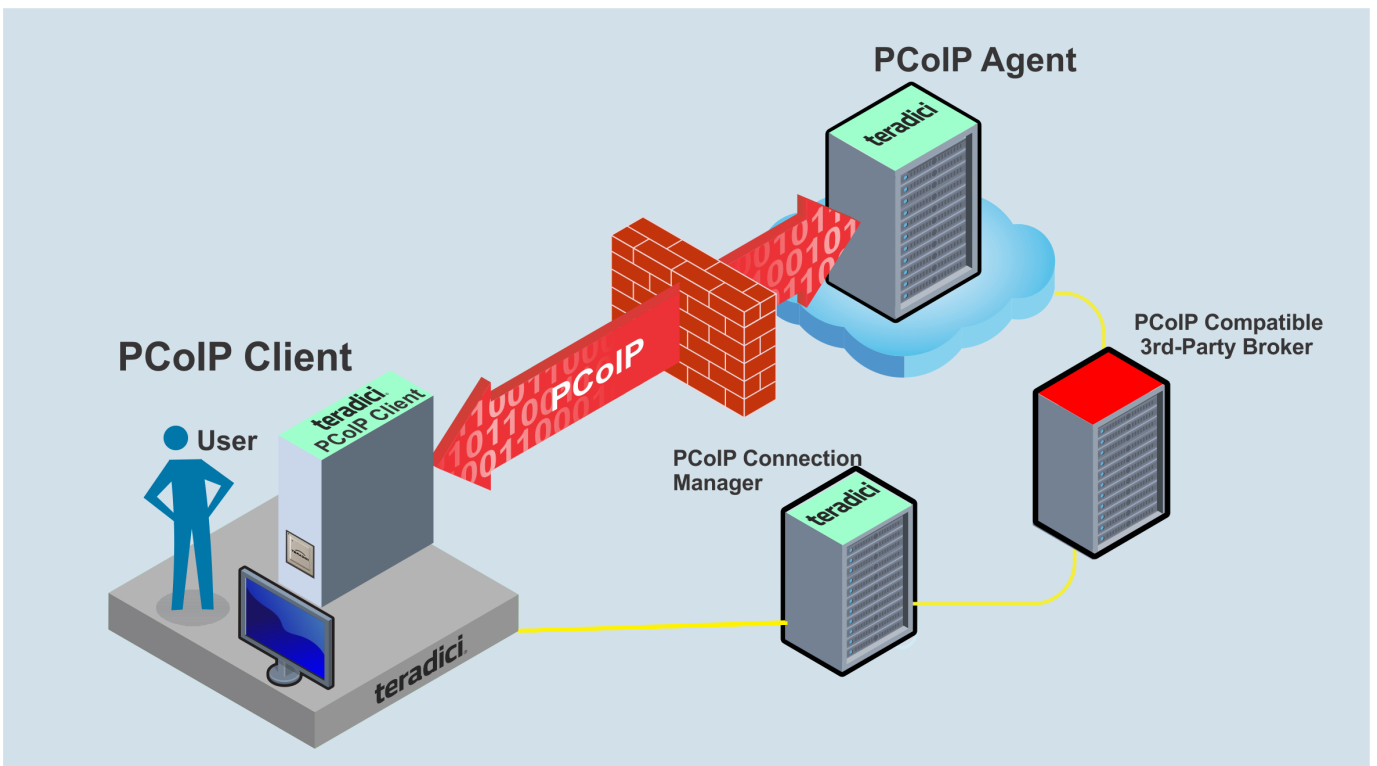
- **Host** refers to a virtual or physical machine with an installed PCoIP agent, which can establish remote sessions with a PCoIP client.
- **Desktop** refers to an entity which is delivered to the client as a remote workload. This is typically a full Windows, Linux, or macOS desktop, but it can also be configured to deliver 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.exe`, 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

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

Feature Support

Wacom Tablet Support

The Session Client SDK for Windows 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

Locally-terminated tablets have greatly improved responsiveness, and tolerate higher-latency (including 25ms and higher) networks.

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.

The following Wacom tablet models have been tested and are supported with local termination.

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

	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 <i>PTH-460</i>	✓	✓	✓	✓	—
Intuos Pro Medium <i>PTH-660</i>	✓	✓	✓	✓	—
Intuos Pro Large <i>PTH-860</i>	✓	✓	✓	✓	—
Cintiq 22HD <i>DTK-2200</i>	✓	✓	✓	✓	—
Cintiq Pro 24 - Pen Only <i>DTK-2420</i>	✓	✓	✓	✓	—
Cintiq 22 <i>DTK-2260</i>	✓	✓	✓	✓	—
Cintiq 22HDT - Pen & Touch <i>DTH-2200</i>	—	—	—	—	—
Cintiq Pro 24 - Pen & Touch <i>DTH-2420</i>	✓	✓	✓	✓	—
Cintiq Pro 32 - Pen & Touch <i>DTH3220</i>	✓	✓	✓	✓	—

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 Windows.

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

	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 <i>PTH-460</i>	✓	✓	✓	✓	—
Intuos Pro Medium <i>PTH-660</i>	✓	✓	✓	✓	—
Intuos Pro Large <i>PTH-860</i>	✓	✓	✓	✓	—
Cintiq 22HD <i>DTK-2260, DTK-2200</i>	✓	✓	✓	✓	—
Cintiq Pro 24 <i>DTK-2420</i>	✓	✓	✓	✓	—
Cintiq 22HDT - Pen & Touch <i>DTH-2200</i>	✓	✓	✓	✓	—
Cintiq Pro 24 - Pen & Touch <i>DTH-2420</i>	✓	✓	✓	✓	—
Cintiq Pro 32 - Pen & Touch <i>DTH3220</i>	✓	✓	✓	✓	—

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 remoteing 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

Webcam Support

PCoIP Software Client for Windows 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](#).

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

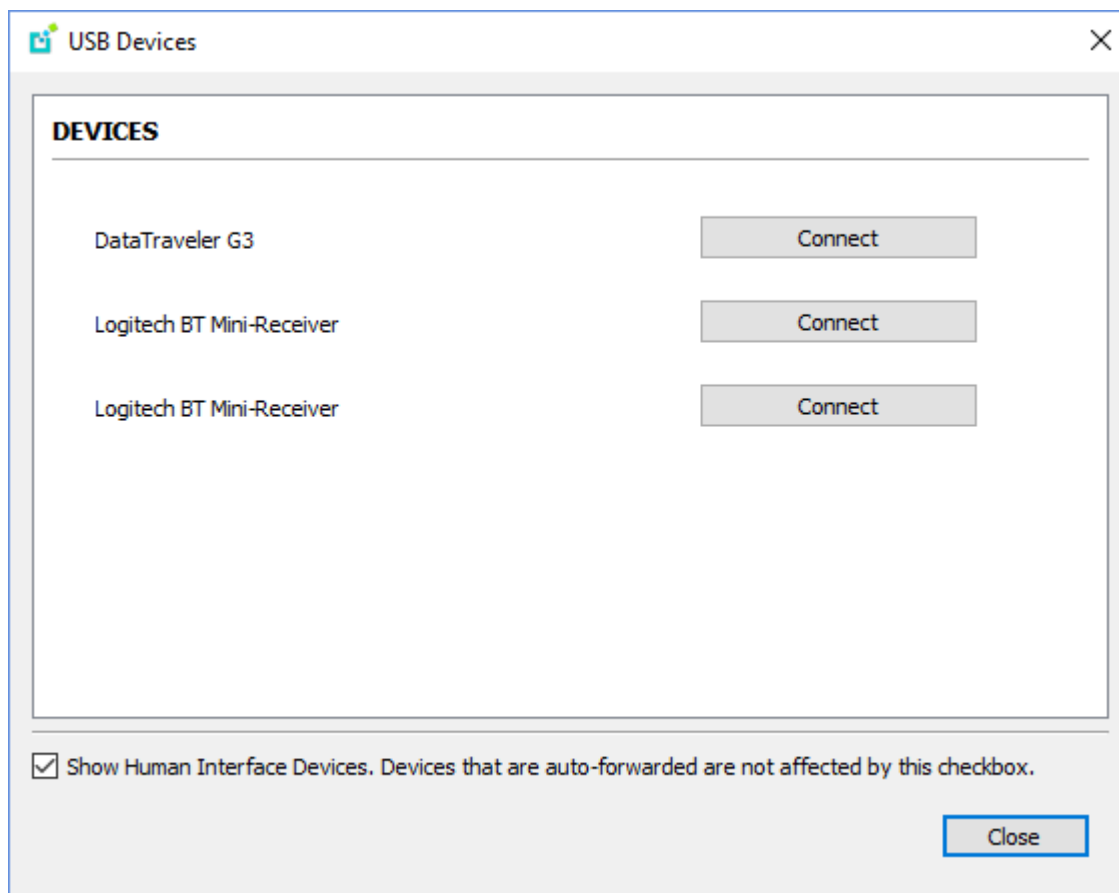
The USB menu will only show up if the PCoIP Agent has been configured to enable USB redirection and a USB device has been detected by the PCoIP Client.

ePadLink Signature Pad and some Wacom tablets are connected via local termination

If you are using an ePadLink Signature Pad and certain Wacom tablets they can be connected via local termination. You do not need to carry out any other steps to establish this connection. For information on which Wacom tablets are locally terminated, see [Wacom Tablets](#).

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

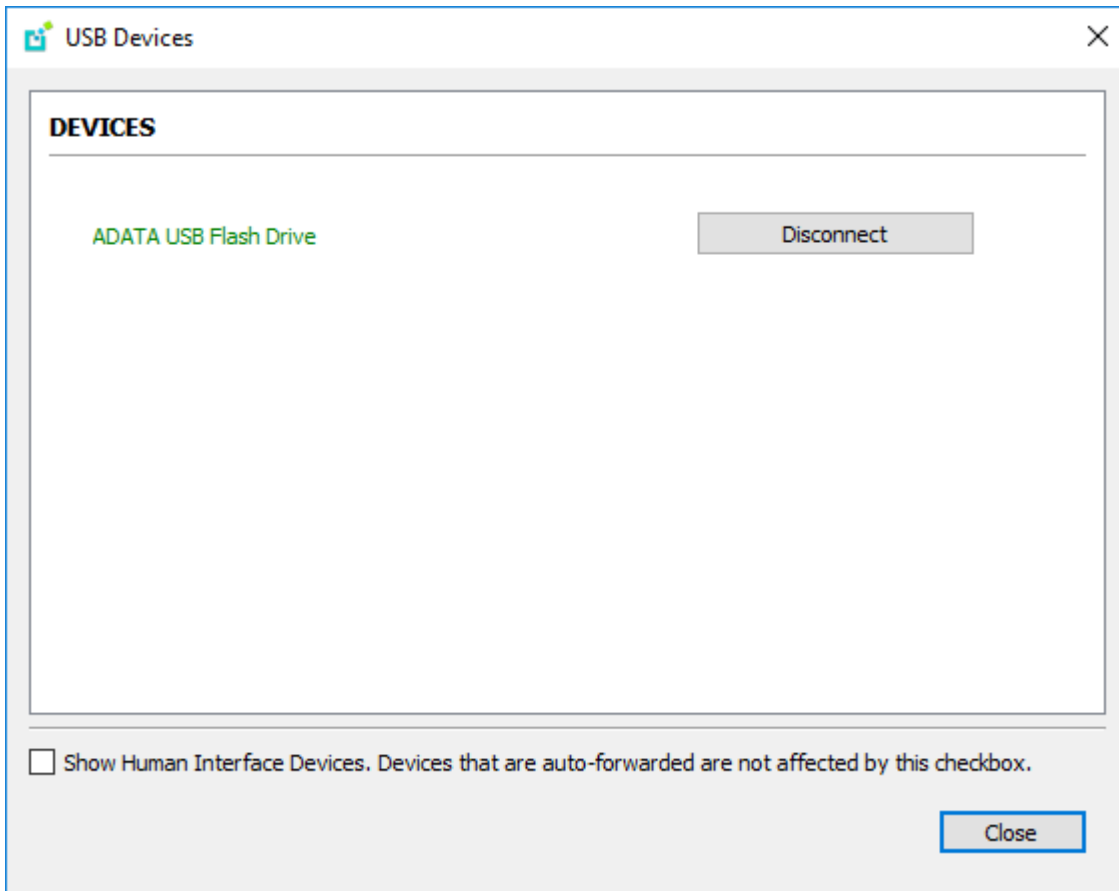
Most Human Interface Devices (HIDs), such as keyboards and mice, are automatically handled by the PCoIP Software Client and do not appear on in this list even if they use a USB connection.

If you need to connect a Human Interface Device that 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.
2. Click **Disconnect** beside the USB device you wish to disconnect.



Calling USB Functions

Calling USB functions will fail unless the application a user uses for calling the functions is signed and the Teradici USB service trusts the application.

Signing the Client Application

The application must be signed in a way that is trusted by the OS. To check this:

1. Right click on the executable.
2. Select the Signature tab.
3. Verify the signature is valid.

While developing the application the signing process may be incorporated into the build process, either in the build script or as a post-build task in the IDE. If a self-signed certification is being used for testing purposes, it must be added to the trusted root CAs to enable the OS to trust it.

Enabling the Signed Application to be trusted by the USB Service

The registry key for the Teradici USB service is located at *HKLM\System\CurrentControlSet\Services\usbhub*. Within this folder there is a subkey called **Issuers**. That value contains the SHA512 hashes of ASN.1 encoded public keys of all trusted signers.

In order to make the USB service trust the application, the SHA512 hash of the ASN.1 encoded public key of the certificate used to sign the application must be added. The hash of the certificate public key may be obtained using the `CryptHashPublicKeyInfo` Windows API.

Final Version Installer

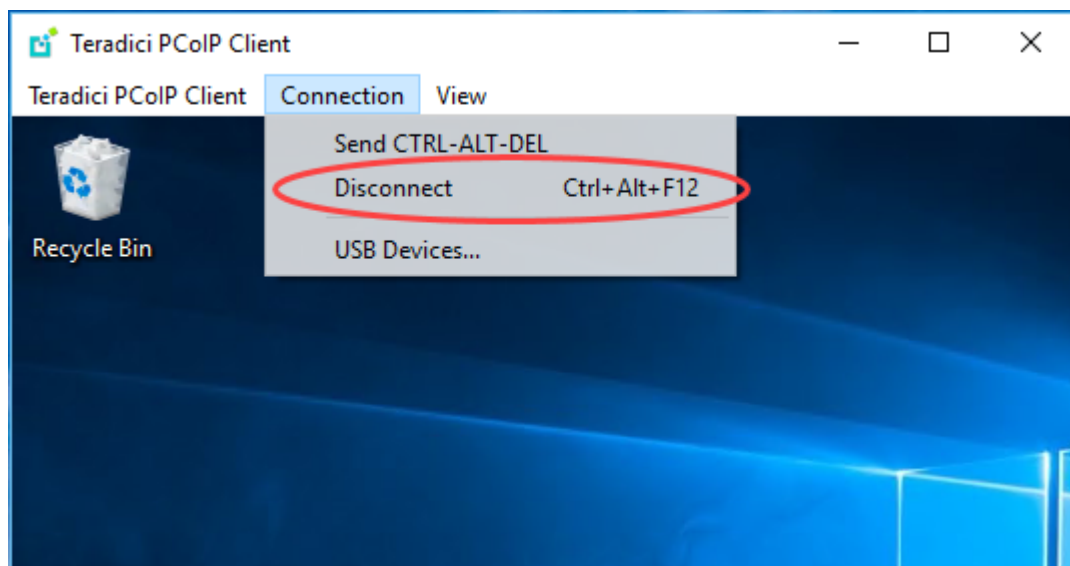
The installer of the final version must add the hash of the signor for the final application to the **Issuers** registry value.

There is an additional value that may be created under **parameters** for debugging purposes. It is a string value named **LogDir**. If this value exists the USB service will create a log file that may help

identify access issues. If the value is empty, the log will be created in the same folder as the USB service executable.

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 + Alt + 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 it is not successful, 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**.

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.

Enhanced Audio and Video Synchronization

Enhanced Audio and Video Synchronization (A/V Sync) 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 video 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.

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:

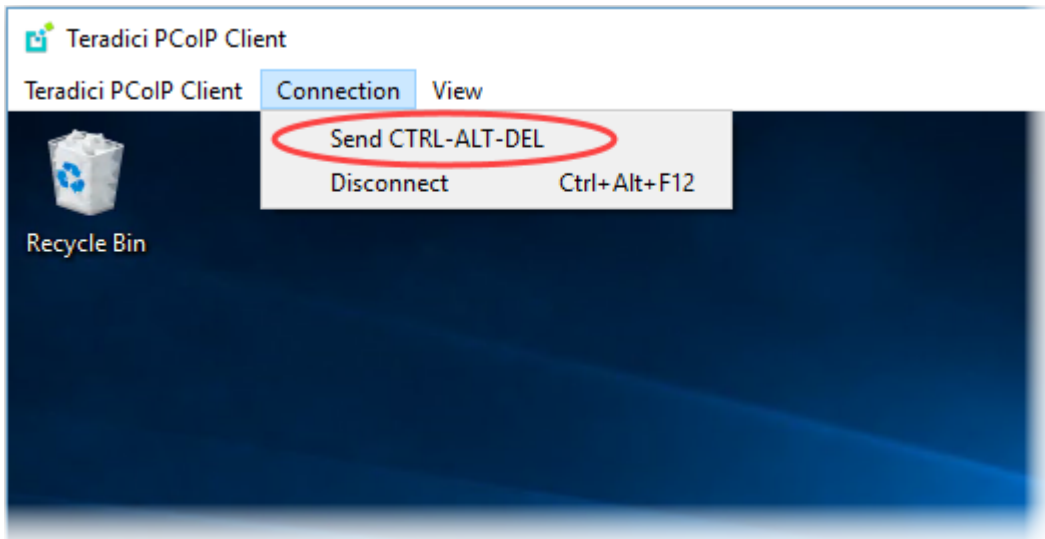
1. If you are in full-screen mode, reveal the menu bar on the display you want to enhance by moving the mouse cursor to the top of the screen.
2. On the display you want to enhance, select **View>Enhanced A/V Sync** to toggle the enhanced sync mode.

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 **Control Panel > Region and Language > Formats** tab, select another language in the **Format** 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 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 Windows 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 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 Windows users

Support for PCoIP Ultra on the PCoIP Software Client for Windows 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](#).

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 indicator create the following registry key and set

to 0:

**Computer\HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Teradici\PCoIP\pcoip_admin_defaults\
pcoip.codec_indicator (DWORD)**

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 Folder
- Log Prefix
- 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+Delete+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. The following rules apply to VID and PID values:

- VID/PID values are comma-separated: `xxx,yyy`
- VID/PID pairs are space-separated: `aaa, bbb ccc,ddd`
- VID/PID pairs with invalid values will be discarded. Discarded rules appear in the event log.
- Up to 20 devices will be passed; if more than 20 are attempted, the first 20 will be accepted and rest ignored. Ignored rules appear in the event log.

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
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 Folder

`log-folder` is the Log Folder parameter. This is the user defined folder for log files.

Log Prefix

`log-prefix` is the Log Prefix parameter. This is the log filename that the SDK will add time information to for each log file. The resulting log file will be `[[log-folder]/[log-prefix]][timeinformation]`.

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-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 labels:
 - *About PCoIP Client*
 - Quit PCoIP Client_ (Windows only)
- 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.

For more detailed, technical information on branding your session client, creating and using the branding text layout file, see [Creating a Branding Text Layout File](#).

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. By default, Windows PCoIP agents allow bridging of all USB devices. 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.

Controlling USB support on Windows PCoIP Agents

For information about USB bridging configuration on Windows PCoIP agents, see one of the following guides: - [Teradici PCoIP Graphics Agent for Windows Administrators' Guide](#) - [Teradici PCoIP Standard Agent for Windows Administrators' Guide](#)

USB support on Linux PCoIP Agents

Linux PCoIP agents do not support non-HID USB devices. Only HID devices such as keyboards and mice can be used with Linux PCoIP agents.

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.

Windows Clients Require an Additional Package

To enable USB support on Windows clients, you must install the Client USB package

For more technical information around USB device configuration, installation and Wacom tablet configuration, see [Manually Bridging USB Devices](#).

Configuring Wacom Tablets

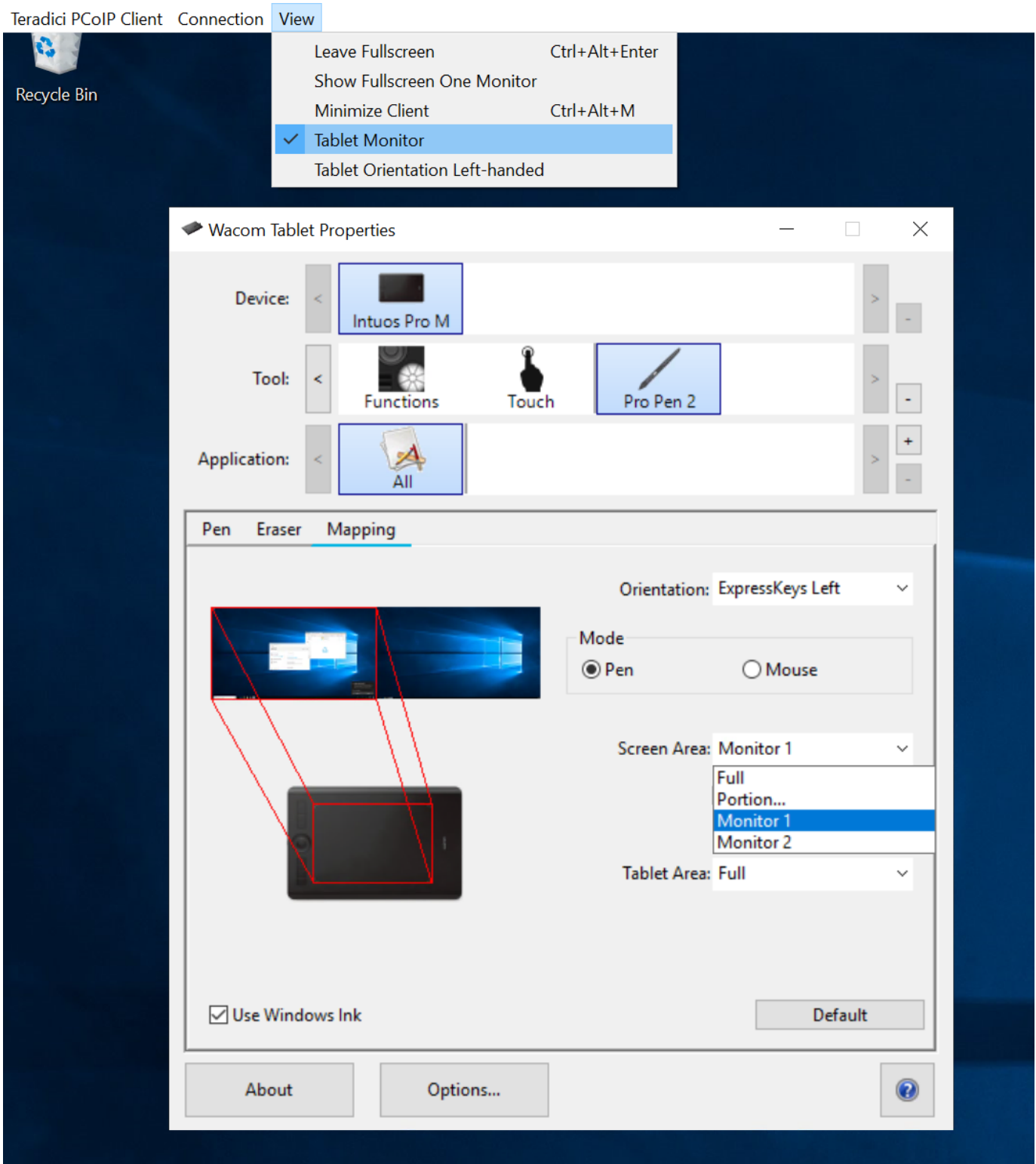
This section outlines how to configure your Wacom tablet through the PCoIP Client session. There are two available features within the PCoIP Client that can be used to configure the monitor display and orientation. This only applies to locally terminated Wacom Tablets.

Tablet Monitor

The Tablet Monitor feature enables you to select the monitor you want to use with your Wacom tablet. You can change between using a pen or mouse and select the orientation position.

To configure Tablet Monitor settings:

1. Select **View** from the in-session options bar.
2. Check the **Tablet Monitor** option.
3. Open **Wacom Tablet Properties** from the Wacom Desktop Center.
4. Select your device, tool and application.
5. Select your screen area from the dropdown menu.



Tablet Orientation Left-handed

The left-handed orientation configures the tablet for a left-handed orientation. Select **ExpressKeys Right** for a left-handed orientation, and **ExpressKeys Left** for a right-handed orientation. Rotate the tablet to the desired orientation.

To configure Tablet Orientation:

1. Select **View** from the in-session options bar.
2. Check the **Tablet Orientation Left-handed** option.
3. Open **Wacom Tablet Properties** from the Wacom Desktop Center.
4. Select your device, tool and application.
5. Select your orientation from the dropdown menu.

Teradici PCoIP Client Connection View

- Leave Fullscreen Ctrl+Alt+Enter
- Show Fullscreen One Monitor
- Minimize Client Ctrl+Alt+M
- Tablet Monitor
- ✓ Tablet Orientation Left-handed

Wacom Tablet Properties

Device: Intuos Pro M

Tool: Functions Touch Pro Pen 2

Application: All

Pen Eraser Mapping

Orientation: ExpressKeys Right

Mode: Pen

Screen Area: Monitor 1

Force Proportions

Tablet Area: Full

Use Windows Ink

Default

About Options...

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

```
%PROGRAMDATA%\Teradici\Teradici PCoIP Client.ini
```

User Scope

```
%APPDATA%\Teradici\Teradici PCoIP Client.ini
```

To add a new key append key=value to the .ini file:

If you are creating a new file, the format of the file should be:

```
[General]  
<key>=<value>
```

For example:

```
[General]  
security_mode=2
```

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"
```

Session API

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.exe`:

Pass the pieces of Information Individually to the Executable The following command invokes `pcoip_client.exe` to establish a PCoIP session and passes the connection and session information as parameters, where:

- **Connection tag:**

```
SCS1WsopFJ3iz1l48PTJMXFkcD4b6M9aiakHXH3e1lLhUR0BceWAifSSn%2b4AV1FC8IihWmsISmYFKeA25AtzF
```

- **IP address:** 10.64.60.115

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

- **Session ID:** 2305843009213693961

```
pcoip_client.exe -i connect-tag=
SCS1WsopFJ3iz1l48PTJMXFkcD4b6M9aiakHXH3e1lLhUR0BceWAifSSn%2b4AV1FC
8IihWmsISmYFKeA25AtzFrdMpdaCtqliC0zfxAA 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.exe` 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%2b4AV1FC8IihWmsISmYFKeA25AtzFrdMpdaCtqliC0zfxAA

- **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.exe connect-tag=
SCS1WsopFJ3iz1l48PTJMXFkcD4b6M9aiakHXH3e1lLhUR0BceWAifSSn%2b4A
V1FC8IihWmsISmYFKeA25AtzFrdMpdaCtqliC0zfxAA log-id=a1ff3a80-8801-1038-
a9bd-00505680aded "teradici-pcoip://
10.64.60.115:4172?session-id=230584300921369396"
```

PCoIP Session-Creation Steps and Actors

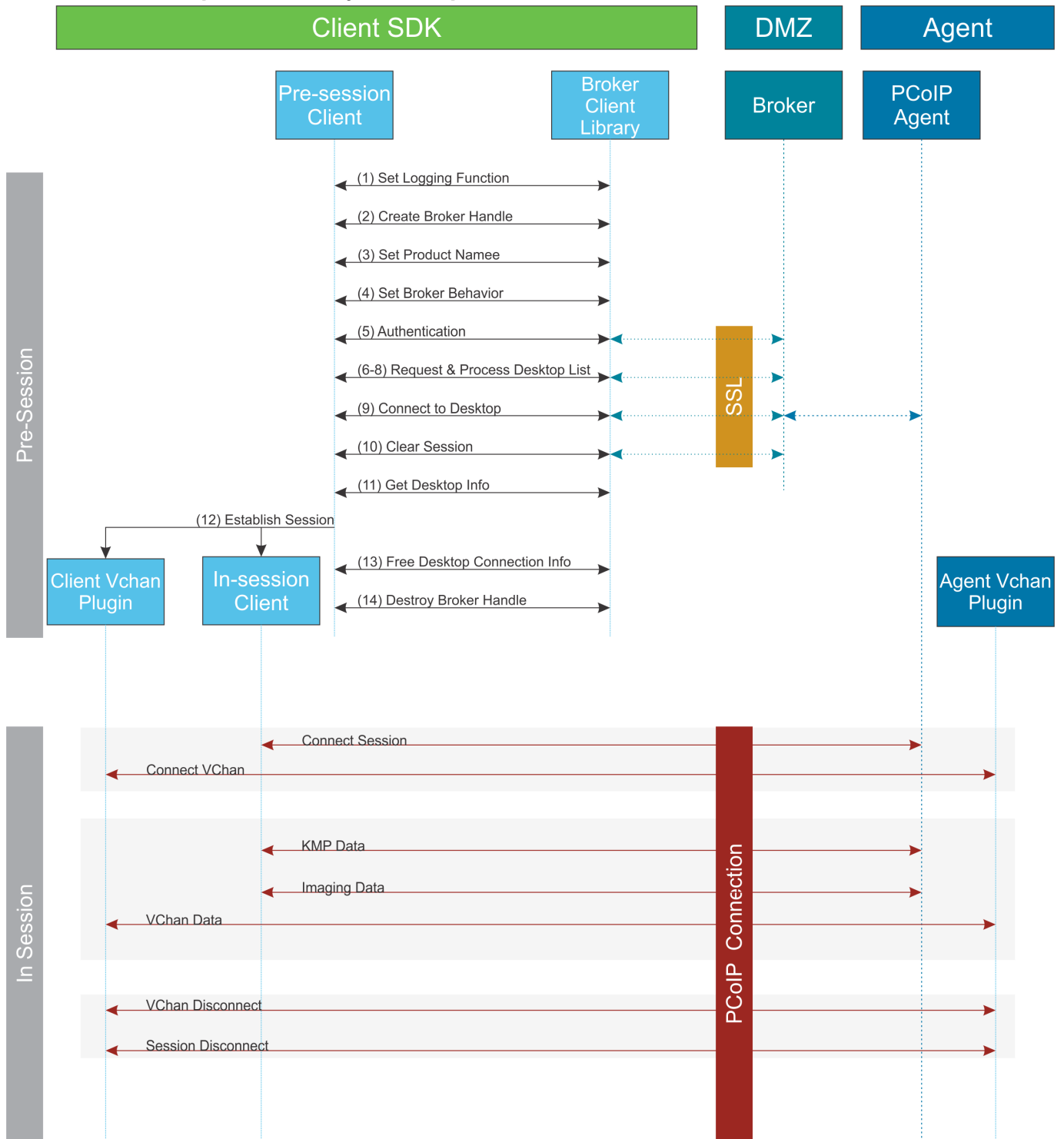
The steps indicated below are used and documented in the bundled sample code. Refer to the code for specific function calls, expected return values and error-handling requirements. The example C++ code can be found in the SDK package located here:

```
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, as 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. Alternatively, this could also be an Amazon Workspaces registration code.
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

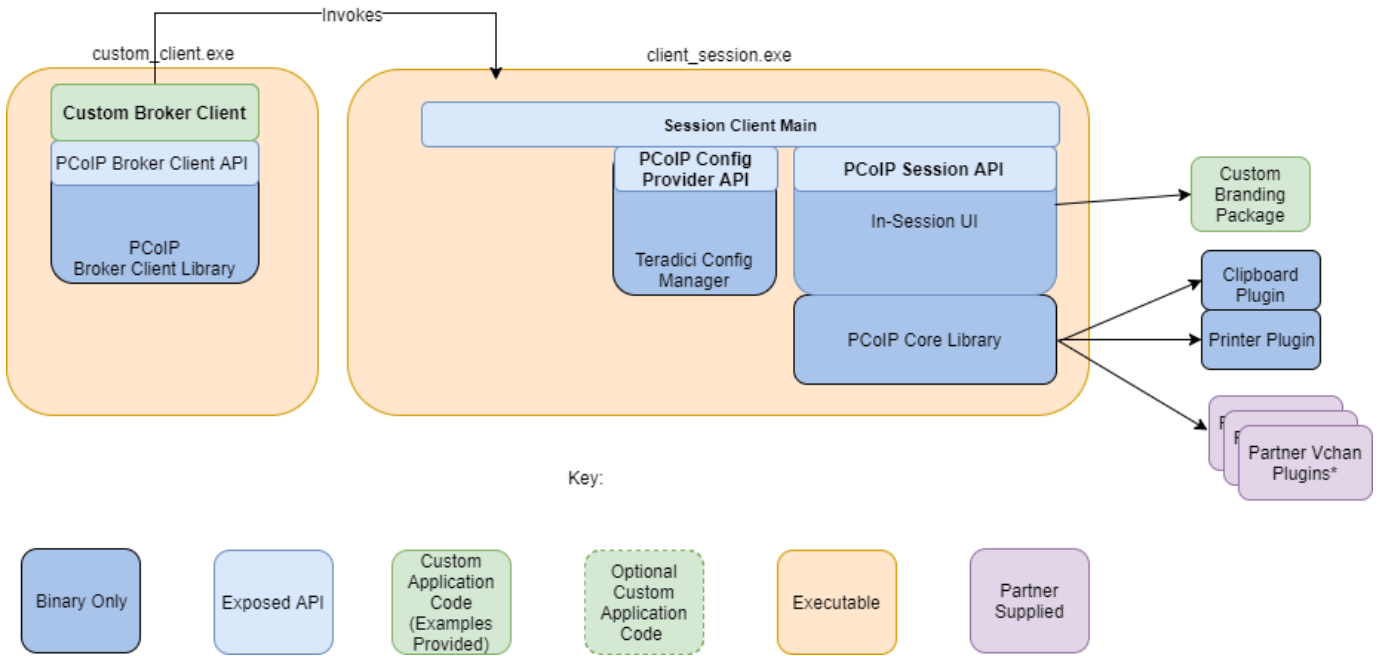
This method uses the session client binary as a separate in-session application. This is the simplest way to use the PCoIP Client SDK and it is recommended as the initial mode to use for developing your own client. In this mode the pre-session and session phases are handled by separate executables:

- 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 `pcoip_client.exe` executable, which must be [installed separately](#), to establish the connection.

Security Considerations

The values passed to the `pcoip_client.exe` 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.exe` 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 could use them before `pcoip_client.exe` 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.exe` into a single executable as described in the following section.

Binary Integration



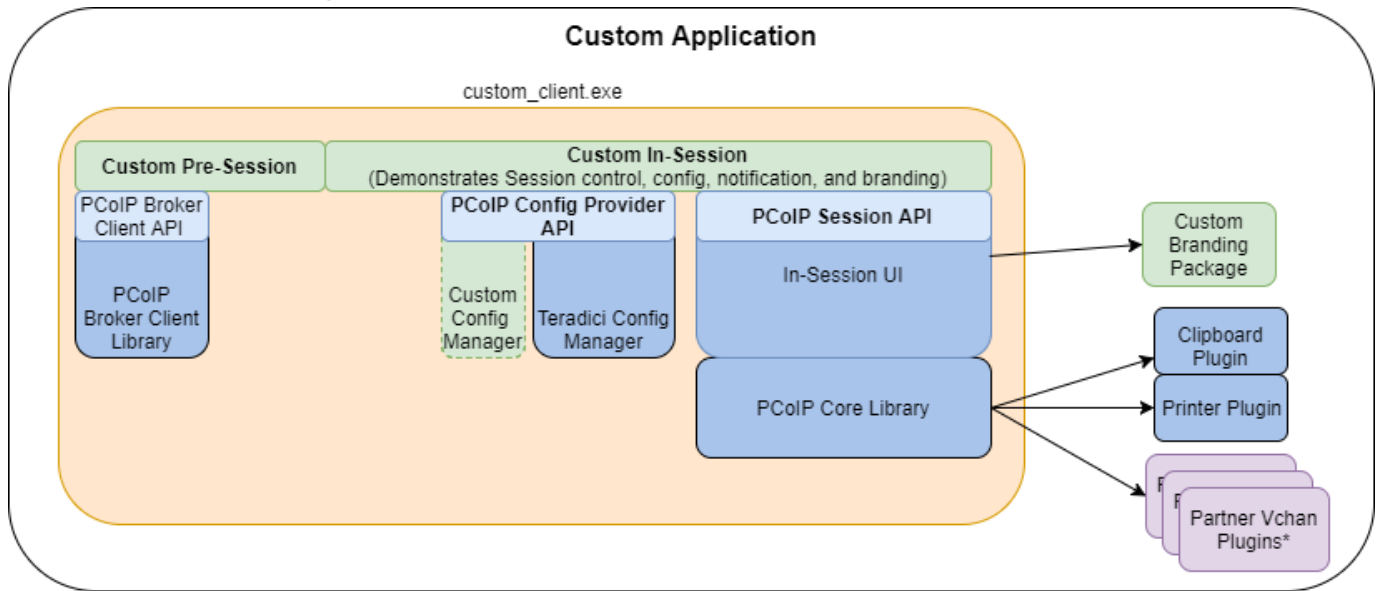
Session Client API Integration

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.exe` executable beyond that which is possible via its command line interface, or if you need to integrate the pre-session and `pcoip_client.exe` 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.exe` 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



Key:



*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](#).

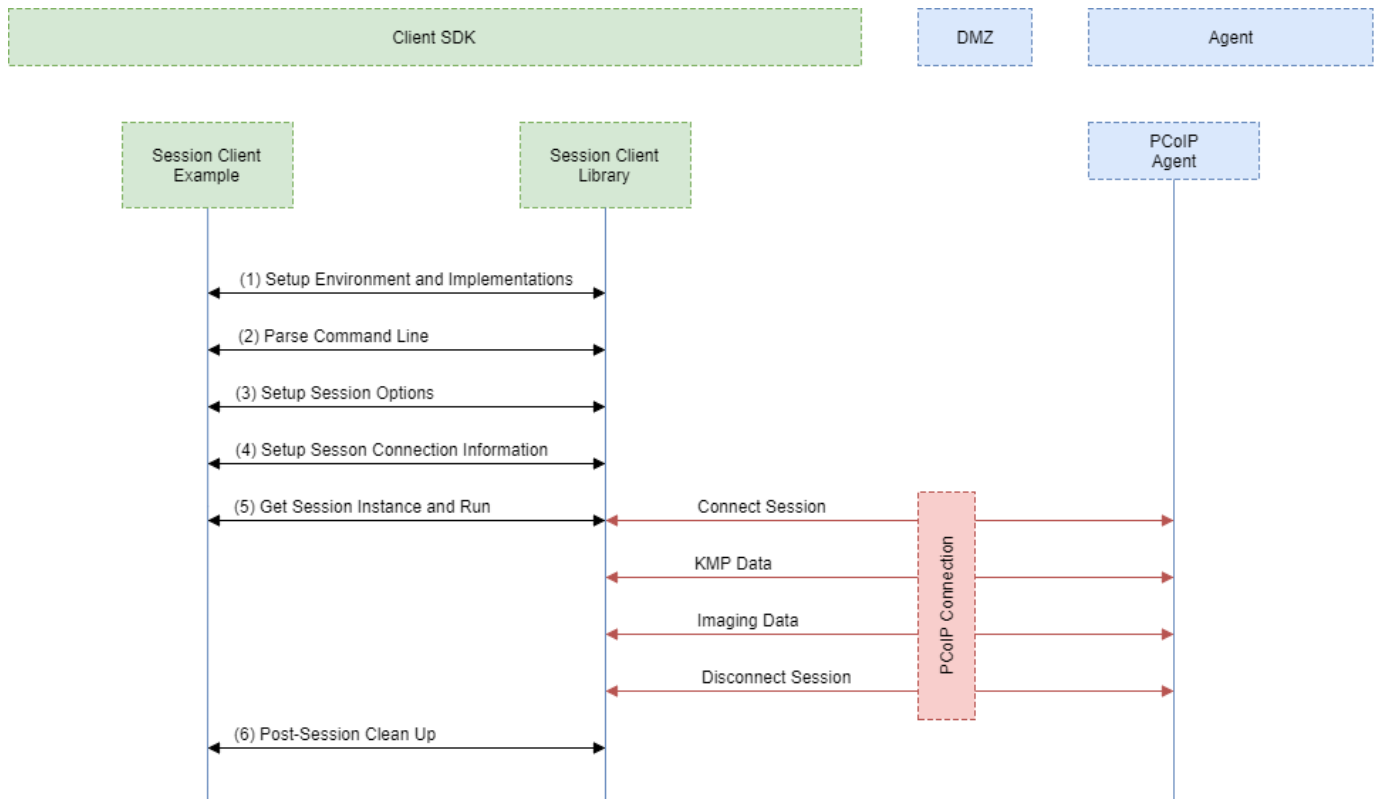
Passing Customization Parameters to the Session Client

When using the `-l` option with `broker_client_example` 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.exe -i login_info.txt -l "disable-menubar"
```

Using the Client Session API

Use of the Client Session C++ API is demonstrated in `<pcoip-client-sdk>\examples\client_session_example\pcoip-client_main.cpp`. It includes the steps outlined in the diagram below:



PCoIP Session Client API Sequence Diagram

Applications and Files Required

The following files and application versions are required to build the minimal client and client session examples:

- Cmake 3.10 or above.
- Windows 10 SDK.
- Microsoft Visual Studio 2019.
- Boost C++ Libraries [1.71.0](#).

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

- **Windows Clients:** `bin\TeradiciBrandingPackageUtility.exe`

To create a custom branding package:

1. Create a product icon (png, 128px x 128px).
2. Create a company logo (png, any size).
3. Create a text layout file describing the customized UI element strings and dialog content.
4. Create the branding package using `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 both 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, noted when creating the branding package, using the parameters `-branding-package` and `-branding-hash`.

For example (one command):

```
pcoip-client -branding-package my_custom_branding.bp  
-branding-hash  
cbc3fd3c6d001a1e1f06342bccccf2a62bd748c3cf1dd2e4c9c29561ea07bd089  
<other-params>
```

Limits on Customization

You may not be able to programmatically modify certain session features due to limitations imposed by the operating systems's user interface.

Windows Limitations

The application process name in Windows Task Manager cannot be altered at run time. The process name will be **PCoIP Client**.

Bypassing run-time Configuration Limitations

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

Broker API

The Broker Client API 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 API Example Sequence

This section describes how the broker client API 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

!!! Note "Custom Broker Client Library Implementations" After you complete step 6, the build/ directory will contain the built examples as .exe files, which you can launch from the command line or double click, depending on what you have built.

1. Install the [latest PCoIP Software Client](#). For more information see the topic "Installing the PCoIP Software Client for Windows" in the [Administrators Guide](#).
2. Download the [latest Windows 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 directory called `build/` inside the `broker_client_example` folder by running the following command: `mkdir build`
6. Change the directory to `build/` by using the `cd` command.
7. Configure the SDK example by running the following command: `cmake -G Xcode ..`
8. Build the example by running the following command: `cmake --build`
9. Ensure that the `CMakeLists.txt` file present inside the directory has the path to the SDK directory appended to `CMAKE_PREFIX_PATH` so that the installed version of `cmake` on the system is able to locate the `cmake` modules for finding the dependencies:

```
list(APPEND CMAKE_PREFIX_PATH "localpath/to/sdk/")  
  
find_package(PCoIPSoftClientSDK REQUIRED)
```

10. Link your application to the core library using the `CMake` command:

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

11. To link other libraries, append them to the above command in step 7.

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: `/PCoIPSoftClientSDK/Examples/broker_client_example`.

1. Install the [latest PCoIP Software Client](#). For more information see the topic "Installing the PCoIP Software Client for Windows" in the [Administrators Guide](#).
2. Download the [latest Windows SDK](#).
3. To install the SDK, extract the contents of the `PCoIPSoftClientSDK` folder to a 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 topic 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: `/PCoIPSoftClientSDK/Examples/broker_client_example`.

1. Install the [latest PCoIP Software Client](#). For more information see the topic "Installing the PCoIP Software Client for Windows" in the [Administrators Guide](#).
2. Download the [latest Windows SDK](#).
3. To install the SDK, extract the contents of the `PCoIPSoftClientSDK` folder to a 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
75539285075371469148082720109113041434163396974558736008282251461494906419802726
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 API 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 API is located here: `bin/` There are several files in this directory, but only two are relevant for the broker client example:

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

```
sa1-w2k8-ch605.autolab.local autolab autorunner mypassword sa1-w2k8-ch605
```

In this example:

- The **FQDN** of the host server is `sa1-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.exe` 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 PCoIP Software Client's user interface.

Initiate Broker Connection Flow

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

To initiate the broker connection using the Broker Client API 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 Windows command line tool and type:

```
broker_client_example -i login_info.txt
```

The Broker Client API example will display a status message similar to the one shown below:

```
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 API Example

Use the `-l` switch (lowercase L, for launch) to have the Broker Client API example invoke the session client. This enables you to send invoke the Session Client API 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 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](#) to 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

!!! Note "Custom Broker Client Library Implementations" After you complete step 6, the build/ directory will contain the built examples as .exe files, which you can launch from the command line or double click, depending on what you have built.

1. Install the [latest PCoIP Software Client](#). For more information see the topic "Installing the PCoIP Software Client for Windows" in the [Administrators Guide](#).
2. Download the [latest Windows 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 directory called `build/` inside the `broker_client_example` folder by running the following command: `mkdir build`
6. Change the directory to `build/` by using the `cd` command.
7. Configure the SDK example by running the following command: `cmake -G Xcode ..`
8. Build the example by running the following command: `cmake --build`
9. Ensure that the `CMakeLists.txt` file present inside the directory has the path to the SDK directory appended to `CMAKE_PREFIX_PATH` so that the installed version of `cmake` on the system is able to locate the `cmake` modules for finding the dependencies:

```
list(APPEND CMAKE_PREFIX_PATH "localpath/to/sdk/")  
  
find_package(PCoIPSoftClientSDK REQUIRED)
```

10. Link your application to the core library using the `CMake` command:

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

Running the Prebuilt Collaboration Example Binary

This topic 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.

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.eyJSc2x1IjoiQWRtaW4iLCJjc3N1ZXIiOiJJc3N1ZXIiLCJvc2Vybni  
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 contains 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.eyJSc2x1IjoiQWRtaW4iLCJpc3N1ZXIiOiJJc3N1ZXIiLCJvc2Vybn  
collaborator 123435
```

In this example:

- The invitation URI is `pcoip://10.11.12.13/connect?data=eyJhbGciOiJIUzI1NiJ9.eyJSc2xlIjoiQWRtaW4iLCJpc3N1ZXliOiJJc3N1ZXliLCJVc2VybmFtZSI6Ikphdm`
- 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
1. 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.exe -l -i login_info_collaborator.txt
```

Building the SDK for Windows

Configuring CMake for PDB Files

In addition to the full .pdb files, a project can be configured to generate stripped .pdb files by adding the `/PDBSTRIPPED:filename` linker switch. For example, in CMake:

```
* set_target_properties( pcoip_client PROPERTIES LINK_FLAGS "/DEBUG  
/PDBSTRIPPED:pcoip_client.pdb" )
```

All full .pdb files are copied over automatically to `CMAKE_PDB_OUTPUT_DIRECTORY` if `CMAKE_PDB_OUTPUT_DIRECTORY` is defined in the cmake file. Stripped .pdb files are generated at the location where the `.VCXPROJ` file is located.

Microsoft Documentation

For more information on debugging with symbols, see [https://msdn.microsoft.com/en-us/library/windows/desktop/ee416588\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ee416588(v=vs.85).aspx)

Windows Build Procedure

External Dependencies

Boost and Qt are external dependencies that the SDK expects to be installed on the system on which the example code is being built.

1. The SDK package has 5 directories. The headers are inside *include/*, libraries are inside *lib/* and *bin/* and the cmake modules required for finding and importing dependencies are inside *cmake/*.
2. In order to build the examples provided inside *examples/*, `cd` into any of the examples directories.
3. Create a folder called *build/* and `cd` into *build/*.
4. To configure the build system run:

```
cmake -A x64 path/to/source/code
```

5. To build the example run:

```
cmake --build .
```

6. Now the *build/* directory should have the built examples as a .exe which you can launch from the command line or double click depending on what you have built.
7. Ensure the CMakeLists.txt file present inside the directory has the path to the SDK directory appended to CMAKE_PREFIX_PATH so that the installed version of cmake on the system is able to locate the cmake modules for finding the dependencies:

```
list(APPEND CMAKE_PREFIX_PATH "localpath/to/sdk/")  
  
find_package(PCoIPSoftClientSDK REQUIRED)
```

8. At this point, you can link your application to the core library using the CMake command:

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

9. You can link 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 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:

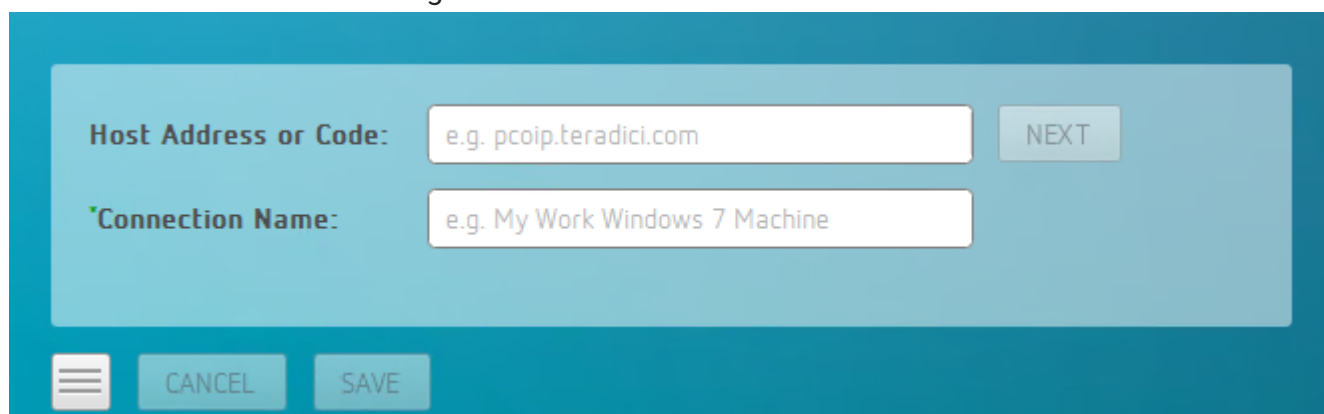
- [Teradici PCoIP Software Client for Windows](#)

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:

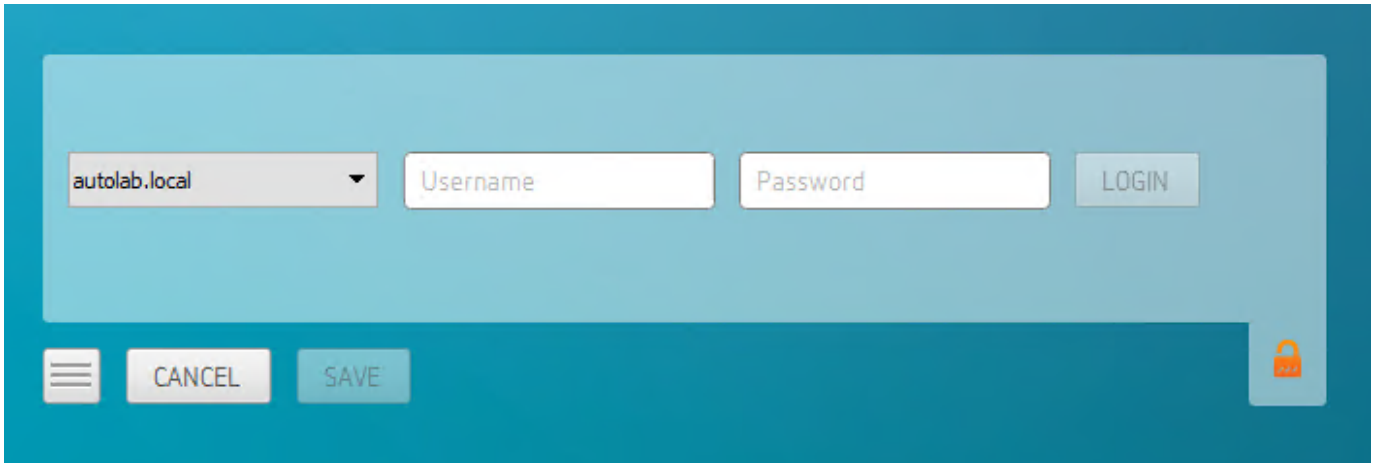
Pre-Session Connection

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 with a teal background. It contains 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 input field. At the bottom left, there is a hamburger menu icon, and at the bottom center, there are "CANCEL" and "SAVE" buttons.

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



The image shows a screenshot of a PCoIP client authentication dialog box. The dialog has a light blue background and a darker blue border. At the top, there is a dropdown menu showing 'autolab.local' with a downward arrow. To its right are two text input fields labeled 'Username' and 'Password'. Further right is a 'LOGIN' button. At the bottom of the dialog, there is a 'CANCEL' button, a 'SAVE' button, and a small orange padlock icon on the right side. A hamburger menu icon is located at the bottom left corner of the dialog.

Extensions

PCoIP Support Bundler Tool

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

To create a support file:

1. Open a PowerShell command prompt.
2. Set the PowerShell execution policy to allow the script to run:

```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

3. Launch the support bundler:

```
path-to-unzipped-sdk-package"/sdk/examples/support-bundler/pcoip-client-support-bundler.ps1
```

The file will be created and placed in the `$env:ProgramData\Teradici\Support\` 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.

Updating the Client USB Package

The Client USB package must be uninstalled before installing a new version.

To update the Client USB Package:

1. Open a command line window.
2. Navigate to the directory where the Client USB package was originally installed.
3. Run the uninstaller located in the `_USB_` directory.
4. Follow the directions in [Installing the Client USB Package for Windows](#).

Rebooting is not Necessary

The installer will recommend rebooting after uninstalling. This is only necessary if you will not be reinstalling, and wish to completely remove the driver from your system. If you will be reinstalling the Client USB package, rebooting is not necessary.

Exit Codes for Programmatic USB Installations and Uninstallations

You can make the installer and uninstaller calls programmatically from your client application. The return and exit codes you should expect are summarized in the following tables:

Installer Return/Exit Codes
0: Success, no reboot required. Expected on a fresh installation.
1: Success, reboot required. Unexpected, but harmless.
2: Success, installation continues after reboot. Expected when running the installer during an upgrade without rebooting (installing a new version after uninstalling an old version, with no reboot in between). This indicates that while the driver is usable immediately, additional action has been scheduled following the next reboot. This prevents scheduled actions by the uninstaller from disabling the driver.
-1: Error, general. An unexpected error occurred. The most likely cause is insufficient privileges to install drivers, or another permissions issue.

Uninstaller Return/Exit Codes
0: -
1: Success. Expected
2: -
-1: Error, general. An unexpected error occurred. The most likely cause is insufficient privileges to ininstall drivers, or another permissions issue

Frequently Asked Questions

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.

Q: Does the SDK provide API for using managed installation systems like MSI?

A: The SDK does not provide an API for managed installation. You are free to choose your own installation method, including MSI.