PCoIP® Client Software Development Kit for Windows

Version 3.5

Developers' Guide

teradici®

TER1608003-3.5

Contents

Who Should Read This Guide

This guide provides information for software developers and systems integrators with an understanding of SDK integrations and virtualization systems who are developing customized PCoIP clients. Readers should already understand the Teradici Cloud Access Software and how it is used in virtual environments, in both brokered and non-brokered sessions.

This guide will break down how to use the session client executable and how to use the session client API.

This document is not intended for users who are unfamiliar with SDK integrations, or for Teradici Cloud Access Platform users who do not require a customized PCoIP client.

In this guide, you'll learn about:

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

Note: Understanding terms and conventions in Teradici guides
For information on the industry specific terms, abbreviations, text conventions, and graphic symbols used in this guide, see Using Teradici Product and Component Guides and the Teradici Glossary.

What's New in this Release?

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

Enhanced Security and Stability

This release contains improvements and enhancements around the security and stability of the Session Client SDK.

Session Log-ID

The session log-id has been added to allow for traceability between all the PCoIP log files.

Wacom Tablet Support

PCoIP Software Clients support Intuos Pro PTH-660 and PTH-860 tablets when connected to Standard Agents or Graphic Agents for Windows on low-latency networks (<25ms).

About PCoIP Sessions

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

System Actors

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

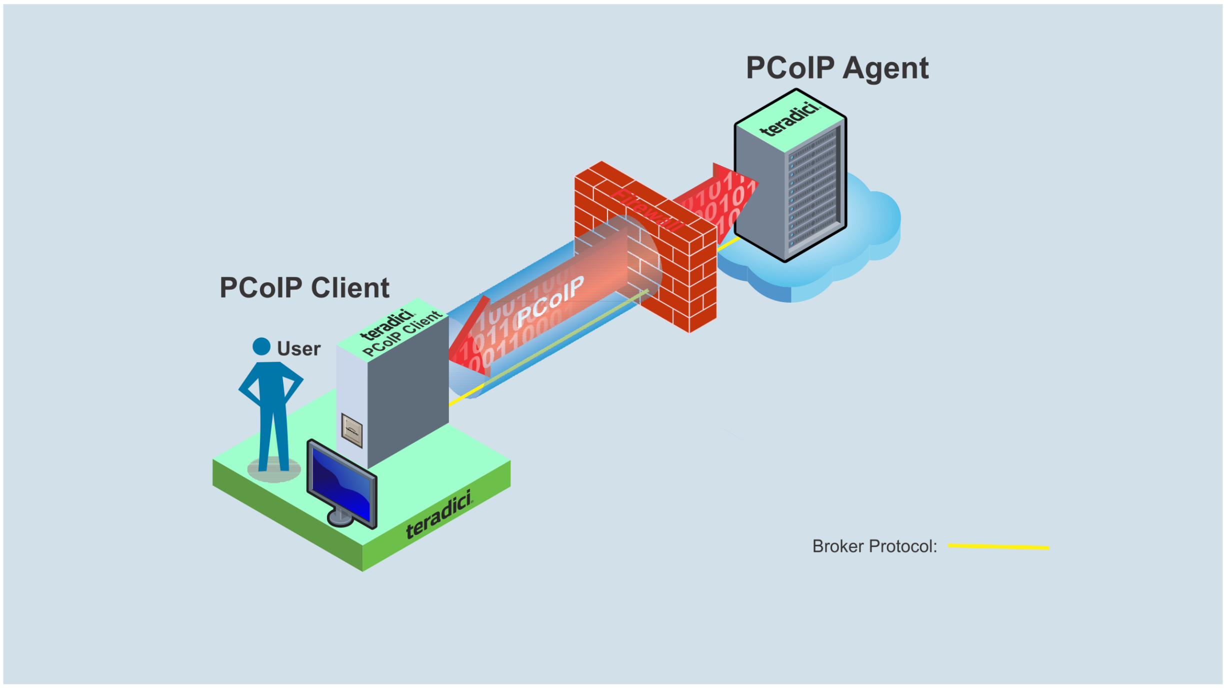
- **PCoIP client** The hardware or software device, local to the user, which requests and drives the PCoIP session by negotiating with PCoIP brokers and PCoIP agents.
- **PCoIP broker** Brokers maintain lists of active users, their authentication information, and the host machines they can connect to.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

Terminology note: Hosts and desktops

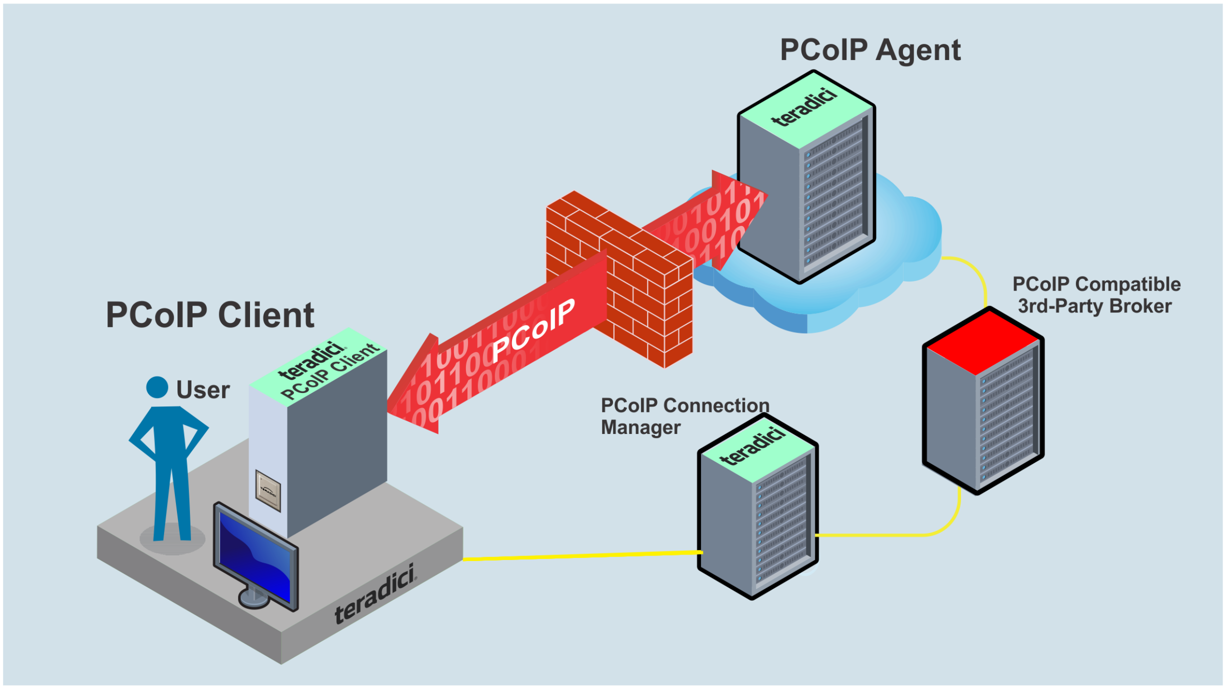
Host refers to a Windows, or Linux machine, either virtual or physical, which has a PCoIP agent installed and which can serve a remote desktop to a PCoIP Client.

Desktop refers to the entity which is delivered to client as a remote workload. This is typically a full Windows or Linux desktop, but it can also be configured to present a single application.

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



Direct Connection



Brokered Connection

Session Client Integration

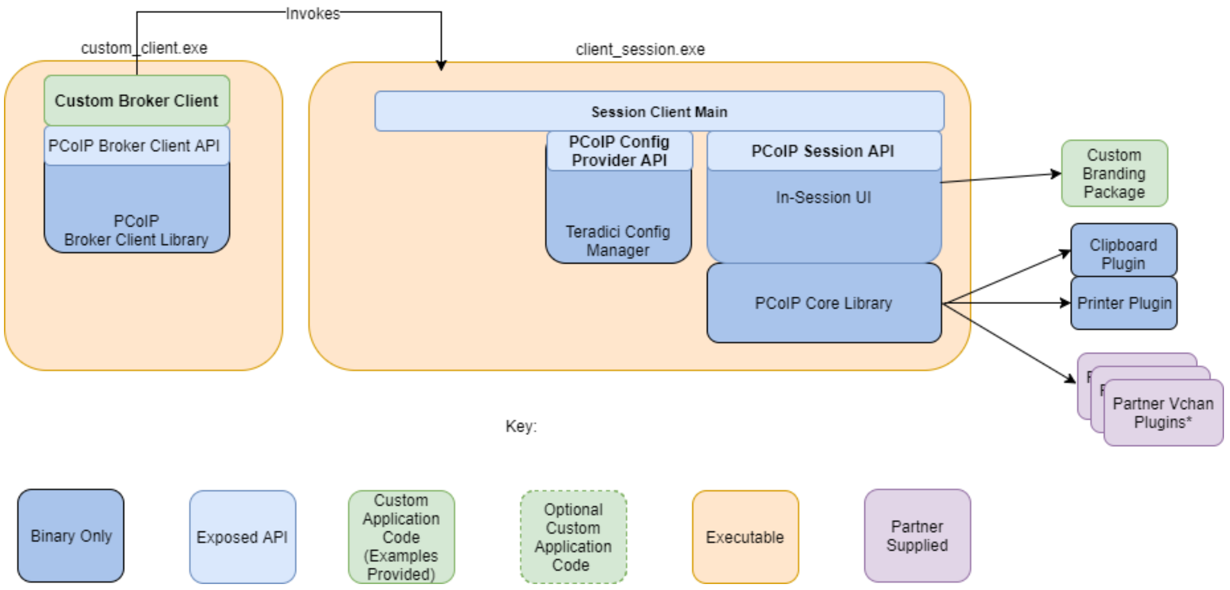
There are two methods of integrating the in-session phase, integrating using the session client binary or using the session client API. The following diagrams show these methods in relation to integrating the SDK into a custom application:

Session Client Binary Integration

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

- Write a custom pre-session executable according to your workflow and needs using the *broker_client_example* source code as a starting point. For more information see Session Client API Integration on page 9.
- Use the stock *ClientSession app* executable (found within the SDK at product/window/x86/release/ClientSession.dmg) to establish the connection. By default, ClientSession.app will be installed to /Applications folder.

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



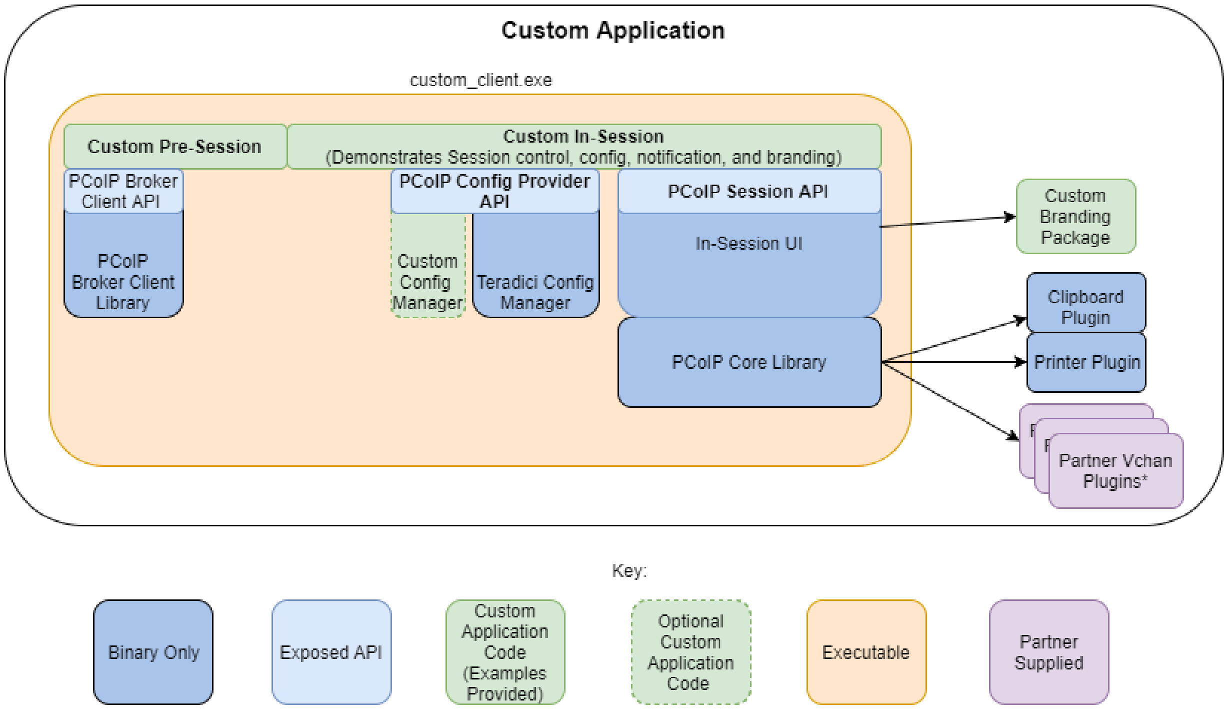
Session Client Binary Integration

Session Client API Integration

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

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

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



Session Client API Integration

*Partner Virtual Channel Plugins can be developed using the Virtual Channel SDK, to enable this you can combine the PCoIP Client SDK with the PCoIP Virtual Channel SDK. For more information, see the *PCoIP® Virtual Channel Software Development Kit 1.1 Developers' Guide*.

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.

  Developers have great latitude to customize this phase for almost any particular requirement. The broker client libraries are provided as one brokering implementation. A completely different brokering solution that does not use the provided broker client libraries is possible. For a step-by-step walkthrough of the pre-session phase, with examples, see *The Broker Client Example Sequence* on page 36.

- **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* on page 14.

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.

Note: About the included example pre-session client
The included pre-session client, `broker_client_example.exebroker_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.

About the PCoIP Client Software Development Kit for Mac for Windows

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

The SDK includes the following components:

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

Supported Platforms

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

Windows	MacOS
• Windows 10 (64-bit) • Windows 7 (64-bit)	• macOS 10.13 Sierra • macOS 10.12 El Capitan

What Can You Build With the PCoIP Client SDK?

The PCoIP Client SDK provides developers the ability to embed a PCoIP session into any program or solution, or create a standalone client with a completely custom user interface and workflow.

With complete control over how a PCoIP client is built, you can create clients that incorporate customizations in both *pre-session* and *in-session* phases of a PCoIP connection. For example, the following customizations are all typical use cases:

Pre-Session Customizations	In-Session Customizations
• Customizing the client user interface to create a branded, end-to-end solution using company assets such as: ◦ Corporate logos ◦ Slogans, trademarks, or other text ◦ Corporate colors and iconography • Developing customized authentication workflows, either directly or using a broker • Automatically connecting users to specific desktops or applications, based on an identified user type or task • Embedding a remote workload into an application	• 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 full-screen mode • Configuring resolutions • Securely using local and bridged USB devices

Customizing the PCoIP Session

The following PCoIP session features can be customized:

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

Tip: 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) `Ctrl`+`Option`+`F12`(which disconnects a PCoIP session)are available to users by default. The parameter for this feature is `disable-hotkeys`.

Windowed or Fullscreen Mode

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

Set Host Resolution

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

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

Custom Client Branding

You can customize the branding of your custom session client in several ways by creating a client branding package. These customizations affect the user’s experience once they have connected to their PCoIP session. The parameters for this feature are `branding-package` and `branding-hash`.

The following elements can be customized in the session client:

- The OS application title and logo
- The session client toolbar title and logo
- The logo displayed in the OS taskbar
- The following default menu item labels:
    - *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 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 and 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 the 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 accomodate 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:

Available Localization Codes and Languages

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)

Note: Default language
By default the language is English.

Session Log-ID

`log-id` is an optional UUID that uniquely indentifies 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

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

Log File Name

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

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

Force Native Resolution

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

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.

Note: A license is required
Before using your test environment, you must install a PCoIP agent development license on the host machine. You received a license when you subscribed to a Teradici All Access solution, specifically Cloud Access or Cloud Access Plus.

If you do not have a license, obtain one from Teradici before proceeding.

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

1. Establish your host virtual machine environment and operating system.

   Resource: PCoIP system architecture reference
   For details about proof-of-concept deployments, including supported PCoIP agents, environments, and operating systems, see *Teradici All Access Architecture Guide*.

2. Determine the PCoIP agent that best fits your actual PCoIP environment.
3. Install the PCoIP agent on the host machine.

   For PCoIP agent installation instructions, refer to the appropriate administrators' guide:

   - *Teradici PCoIP® Standard Agent 2.11 for Windows Administrators' Guide*
   - *Teradici PCoIP® Graphics Agent 2.11 for Windows Administrators' Guide*
   - *Teradici PCoIP® Standard Agent 2.11 for Linux Administrators' Guide*
   - *Teradici PCoIP® Graphics Agent 2.11 for Linux Administrators' Guide*
4. Install your agent license on the host machine.

   For license installation instructions, see the Administrators' Guide for your host machine's PCoIP agent.

Connecting To Your PCoIP Test Agent

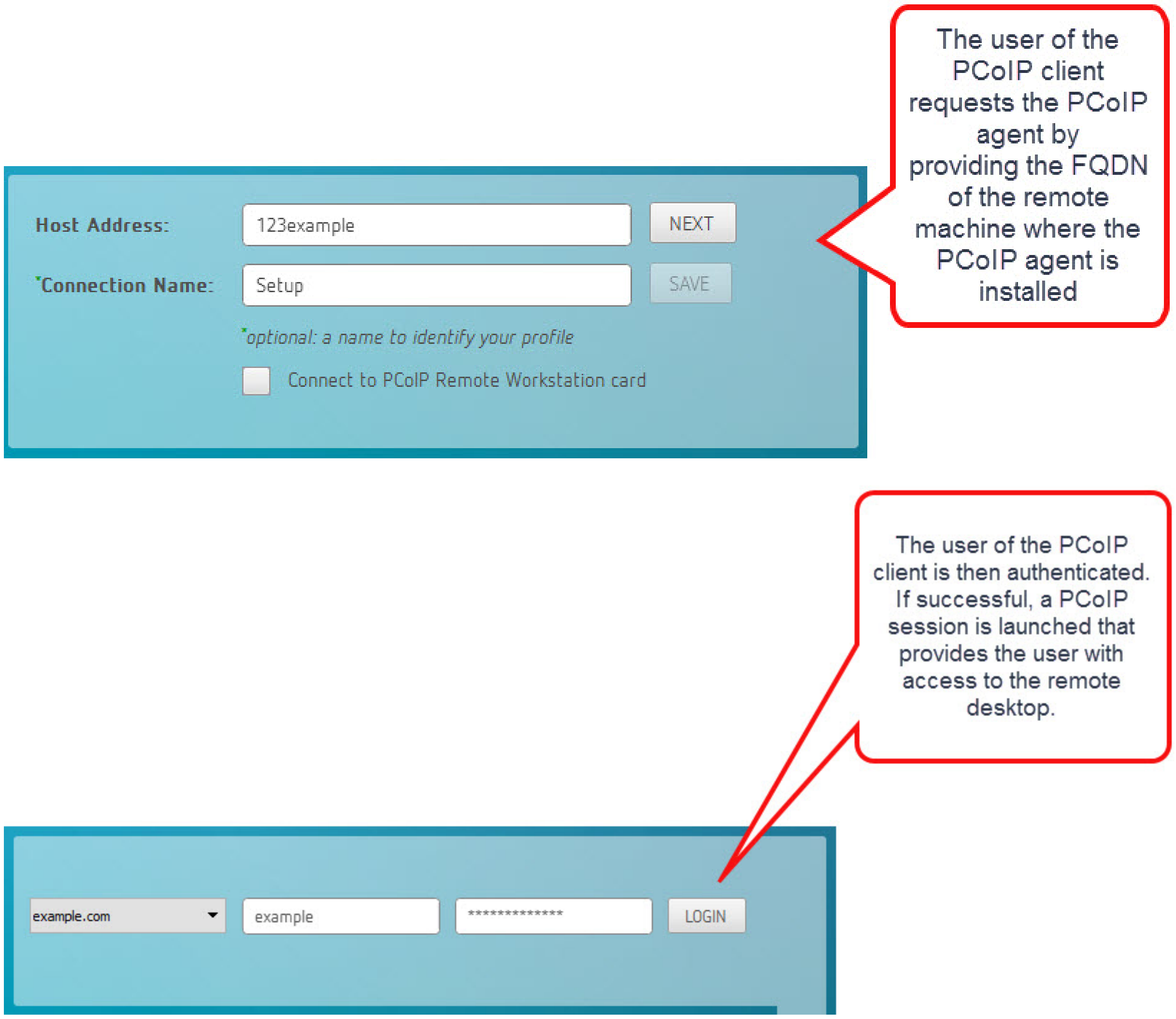
Once your test system is set up, you can establish PCoIP connections to it using Teradici PCoIP Software Clients. For environment testing and troubleshooting purposes, the Teradici PCoIP Software Client is available here:

- Teradici PCoIP Software Client for Windows
- Teradici PCoIP Software Client for macOS

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:


The user of the PCoIP client requests the PCoIP agent by providing the FQDN of the remote machine where the PCoIP agent is installed

Host Address: 123example

*Connection Name: Setup

*optional: a name to identify your profile

Connect to PCoIP Remote Workstation card

The user of the PCoIP client is then authenticated. If successful, a PCoIP session is launched that provides the user with access to the remote desktop.

example.com | example | *************

Pre-Session Connection

Session Client Binary

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

- `<working directory>\pcoip_client_sdk_session\product\windows\x86\Release\ClientSession.app`
- `<working directory>/pcoip_client_sdk_session/product/osx/x86_64/Release/ClientSession.dmg`

For an example of programmatically invoking the in-session client, search for `launch_session` in `pcoip_client_sdk/modules/broker_client_example/src/broker_client_example_main.c`.

After the PCoIP connection is established, several command line options are available from the in-session client, as documented in Customizing the PCoIP Session above.

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.

Limits on Customization

macOS and Windows each have elements which are part of the operating system user interface and cannot be modified programmatically, as described next.

macOS Limitations

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

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

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

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

Important: macOS 10.12 application support
In order to enable branding packages in macOS 10.12, you must remove the quarantine bit from all files in the session SDK package. If you

To remove the quarantine bit, open a terminal and type:

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

Windows Limitations

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

Caution: 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.

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:

Parent element	Child element	Description
`<app-name>`		**Required**. The name of your custom session client. This will be used as the application window title.

Parent element	Child element	Description
`<app-icon>`		**Required**. The file name of your application icon. In macOS, this will be used in the dock. In Windows, this appears in the Windows toolbar and the window header.
`<toolbar-menu>`		**Required**. Describes the text labels used in OS toolbar menus.
	`<about-item>`	The text label for the *About...* menu item. Optional. Without this field, there will not be an *About* menu item.
	`<quit-item>`	The text label for the *Quit...* menu item. Optional. Without this field, you will not have a *Quit* menu item.
`<about>`		**Required**. Describes the contents of the *About...* dialog. Must have the following required attributes: • **title** (*string*): The dialog title text • **minWidth** (*number*): the minimum pixel width of the dialog For example: `<about title="My Custom Client" minWidth="100">`

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

A full text layout file looks like this:

```
<?xml 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**:

  `pcoip_client_sdk_session\product\windows\x86\Release\TeradiciBrandingPackageUtility.exe`

- **macOS clients**:

  `pcoip_client_sdk_session/product/osx/x86_64/Release/TeradiciBrandingPackageUtility`

To create a custom branding package:

1. Create a product icon (png, 128px × 128px).
2. Create a company logo (png, any size).
3. Create a text layout file describing the customized UI element strings and dialog content. For instructions, see *Creating a Branding Text Layout File* on page 24.
4. Create the branding package with `TeradiciBrandingPackageUtility`.

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

   The system will respond with the output file and hash:

   ```
   Output file: my_custom_branding.bp
   Hash:
   cbc3fd3c6d001a1e1f06342bcccf2a62bd748c3cf1dd2e4c9c29561ea07bd089
   ```

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

Using the Branding Package

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

To use the branding package:

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

   For example (one command):

   ```
   open /Applications/ClientSession.appclient_session -branding-
   package my_custom_branding.bp
   -branding-hash
   cbc3fd3c6d001a1e1f06342bcccf2a62bd748c3cf1dd2e4c9c29561ea07bd089
   <other-params>
   ```

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.

Related: Controlling USB support on WindowsMac PCoIP agents
For information about USB bridging configuration on WindowsMac PCoIP agents, see one of the following guides:

- *Teradici PCoIP® Graphics Agent 2.11 for Windows Administrators' Guide*
- *Teradici PCoIP® Standard Agent 2.11 for Windows Administrators' Guide*

Important: 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.

Important: Windows clients require an additional package
To enable USB support on Windows clients, you must install the Client USB package.

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* on page 30.

Note: 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 occured. The most likely cause is insufficient privileges to install drivers, or another permissions issue.

Uninstaller Return/Exit Codes		
0	–	
1	Success	Expected.

Uninstaller Return/Exit Codes		
2	–	
-1	Error, general.	An unexpected error occured. The most likely cause is insufficient privileges to uninstall drivers, or another permissions issue.

Installing the Client USB Package for Windows

USB support in Windows applications is provided by the Client USB package. The Client USB package installs a filter driver that attaches to USB hub driver instances. This filter driver enables the PCoIP client software to take ownership of USB devices, so they can be forwarded to the host.

The Client USB package must be installed if you want to integrate USB features into your client.

To install the Client USB package:

1. Open a command line window.
2. Enter the following command (all as one line):

```
{working directory}\pcoip_client_sdk_
session\product\windows\x86\Release\usb\PCoIP_Client_USB_
installer.exe /noreboot /autoclose /D={path}
```

Where:

- `{working directory}` is the location where the pcoip_client_sdk_session was unpacked
- `/noreboot` suppresses the recommendation to reboot after install
- `/autoclose` automatically closes the installer after installation
- `/D={path}` specifies the base path for installation.

Note: An uninstaller is automatically created
An uninstaller will be placed at `{path}\USB\`.

Tip: Silent mode
To install or uninstall without pop-up messages, use the `\S` parameter when running the installer.

For example, this command will install the USB package under `C:\Program Files (x86)\Teradici\PCoIP Client\`:

```
{working directory}\pcoip_client_sdk_
session\product\windows\x86\Release\usb\PCoIP_Client_USB_installer.exe
/noreboot /autoclose /D="C:\Program Files (x86)\Teradici\PCoIP Client"
```

PCoIP Support Bundler Tool

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

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

To create a support file:

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

```
pcoip_client_sdk_session/product/osx/x86_
64/Release/ClientSession.app/Contents/Resources/pcoip-client-support-
bundler
```

```
<root_dir>/products/windows/x86/Release/ClientSupportBundler.exe
```

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

Using the Broker Client Example

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

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

The sample broker client is located here:
`<dev root>/product/<os>/<arch>/<Debug|Release>/`

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

- `broker_client_example.exebroker_client_example`
- `login_info.txt`

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

Info: About `login_info.txt`

The broker client example uses a small local text file to supply session input values.

The following is a sample `login_info.txt` file (one line):

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

In this example:

- the **FQDN** of the host server is `sal-w2k8-ch605.autolab.local`
- the **domain** is `autolab`
- the **user** is `autorunner`
- the **password** is `mypassword`
- the **host name** is `sal-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 client's user interface.

Initiate Broker Connection Flow

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

To initiate the broker connection using the broker client example:

1. Open `login_info.txt` in a text editor.
2. Add the following information, in this order, separated by spaces:
   - the *FQDN* of the host server
   - the server *domain*
   - the *user name*
   - the *user password*
   - the *host name*
3. Save the text file.
4. Open a command prompt and change directory to `/product/<os>/<arch>/<debug|release>/`.
5. Open a Windows command line tool and type:

   ```
   # broker_client_example.exe  login_info.txt
   ```

   The broker client example will display a status message similar to the one shown next:

   ```
   Connected Successfully.
   Desktop ID : sal-w7p64-sa15.autolab.local
   ip_addr : 10.64.60.147
   port : 4172
   connect_tag:
   SCS1fw0Zbk+Eu7q2iz0/M7mxfEE52au/3Jedtgp16L/rA8iB00+Er+YJd0yIL0xd9M
   v5V0CDLSDmUNkOCwyyV1+u3w1aA7hXxEWmzhAA
   session_id : 2305843009213693954
   sni : SAL-W7P64-SA15
   URI: "teradici-pcoip://10.64.60.147:4172?session-
   id=2305843009213693954&sni=SAL-W7P64-SA15", PARAMETERS: "connect-
   ```

```
tag=SCS1fw0Zbk%2bEu7q2iz0%2fM7mxfEE52au%2f3Jedtgp16L%2frA8iB00%2bE
r%2bYJd0yIL0xd9Mv5V0CDLSDmUNkOCwyyV1%2bu3w1aA7hXxEWmzhAA"
```

Launching the Session Client from Broker Client Example

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

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

1. Open a command prompt and change directory to `/product/<os>/<arch>/<debug|release>/`.
2. Run the command line client, providing the `login_info.txt` file as an argument:

```
broker_client_example.exe login_info.txt l
```

Passing Customization Parameters to the Session Client

When using the `l` parameter to automatically pass session information to the session client, you can pass additional session client parameters by enclosing them in double quotes. This enables you to demonstrate session client functionality without racing to build a command line string within the session client's 60-second window.

For example, to invoke the session client with the menu bar disabled, type:

```
broker_client_example.exe login_info.txt l "disable-menubar"
```

The Broker Client Example

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

Warning: Code is an API demonstration only

The sample session client, described in the following sections, demonstrates a simple connection scenario using the supplied `broker_client_library`. The example unrealistically assumes that all requests and calls succeed as expected, and performs only basic error handling.

An actual client implementation is likely to be far more complex; for example, you will need to handle failed broker certificate verification, account for other authentication steps beyond a simple user ID and password combination, and any other circumstances dictated by your system requirements.

The Broker Client Example Sequence

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

Custom broker client library implementations

PCoIP clients interact with PCoIP-compatible brokers and PCoIP agents using an abstraction layer called a **broker client library**.

The following example uses the supplied broker client library. You may, however, choose to write your own broker client library to meet specific requirements, or use a third-party 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.

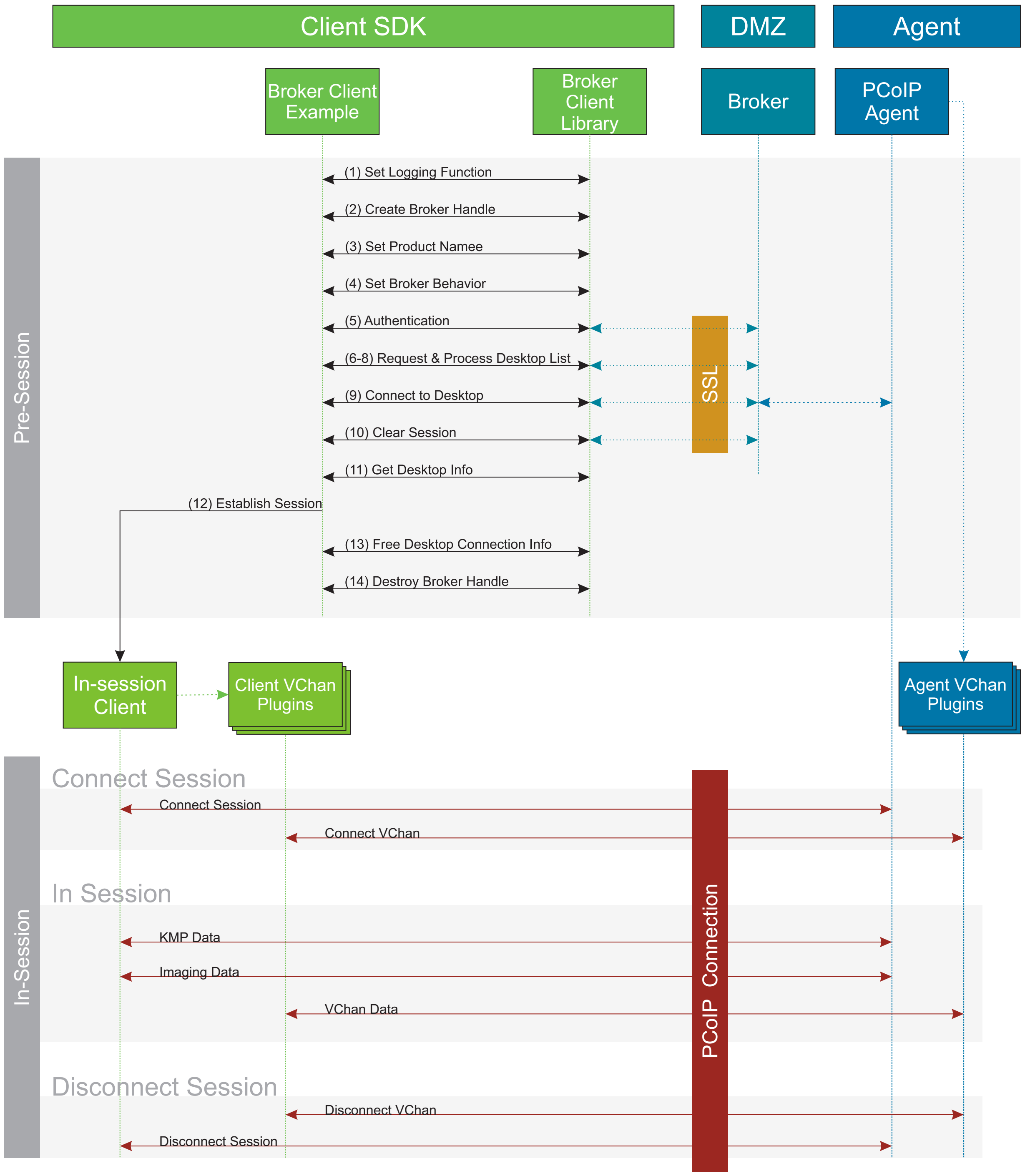
PCoIP Session-Creation Steps and Actors

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`

Note: 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.



The 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.

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.

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

2. **Create a broker handle**

   Create a handle for the broker client instance.

3. **Set client information**

   This information identifies your client to the broker. It should include the client name, client version, and client platform.

4. **Set broker address and behavior on unverified certification**

   This step identifies the address of the broker you want to connect to, and specifies error handling in the event the broker identity cannot be verified.

5. **Authentication between the broker and the client**

   This step requests an authentication method from the broker, and then submits the user's authentication information to the broker using the supplied authentication method.

   The client must implement all the authentication methods required by the broker.

6. **Request desktop list**

   Once the client is successfully authenticated by the broker, request a list of host servers (desktops) that the authenticated user is allowed to access.

7. **Retrieve desktop info**

   Loop through each desktop in the list acquired in step 6, requesting the name and ID of each desktop.

8. **Process the desktop list**

   Perform any processing required on the desktop list, and provide it to the user interface for selection.

Note: Desktop selection presentation can be customized
At this stage, you can also customize the dialogue and interface the user will use to select a desktop.

9. **Connect to selected host server**

   This step asks the broker to set up the PCoIP session. The broker then contacts the agent, which supplies the necessary information (most notably the session tag) the client will need to establish the connection later.

   The PCoIP session is not established yet at this stage.

10. **Clear session with broker**

    On a successful connection, clear the broker session. This effectively disconnects the client from the broker.

11. **Get desktop connection information and launch the session**

    Request the connection and security properties from the desktop (for example, its IP address, its port number, or a session ID), and handle errors if any of the required properties are not returned.

12. **Proceed with established session**

    This step invokes `client_session`, and implements the actual PCoIP connection.

    For specific instructions regarding establishing PCoIP connections, see *How to Establish a PCoIP Session* on page 40.

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.

How to Establish a PCoIP Session

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

- IP address
- Port number
- Session ID

- Server name indication (SNI)
- Connection tag

This information can then be passed to the provided in-session client to establish a PCoIP session programmatically. See the example code for specific call syntax.

In terms of programming interface, there are two ways that the connection and security information can be presented to `ClientSession.appclient_session`:

- **Pass the pieces of information individually to the executable.**
  The following command invokes `ClientSession.appclient_session` to establish a PCoIP session and passes the connection and session information as parameters, where:
  - **Connection tag**: `SCS1WsopFJ3iz1l48PTJMXFkcD4b6M9aiakHXH3ellLhUROBceWAifSSn%2b4AV1FC8IihWVmsISmYFKeA25AtzFrdMpdaCtqlic0zfxAA`
  - **IP address**: `10.64.60.115`
  - **Session ID**: `2305843009213693961`

  ```
  open /Applications/ClientSession.app client_session.exe -i
  connect-tag=
  SCS1WsopFJ3iz1l48PTJMXFkcD4b6M9aiakHXH3ellLhUROBceWAifSSn%2b4AV1FC
  8IihWVmsISmYFKeA25AtzFrdMpdaCtqlic0zfxAA address=10.64.60.115
  session-id=2305843009213693961
  ```

- **Encode all information into a string container (URI) and then pass to the executable.**
  The following command invokes `ClientSession.appclient_session` to establish a PCoIP session and passes the connection tag as a parameter and a URI encapsulating the IP address and Session ID in a string container, where:
  - **Connection Tag**: `SCS1WsopFJ3iz1l48PTJMXFkcD4b6M9aiakHXH3ellLhUROBceWAifSSn%2b4AV1FC8IihWVmsISmYFKeA25AtzFrdMpdaCtqlic0zfxAA`
  - **URI**: "teradici-pcoip://10.64.60.115:4172?session-id=2305843009213693961"

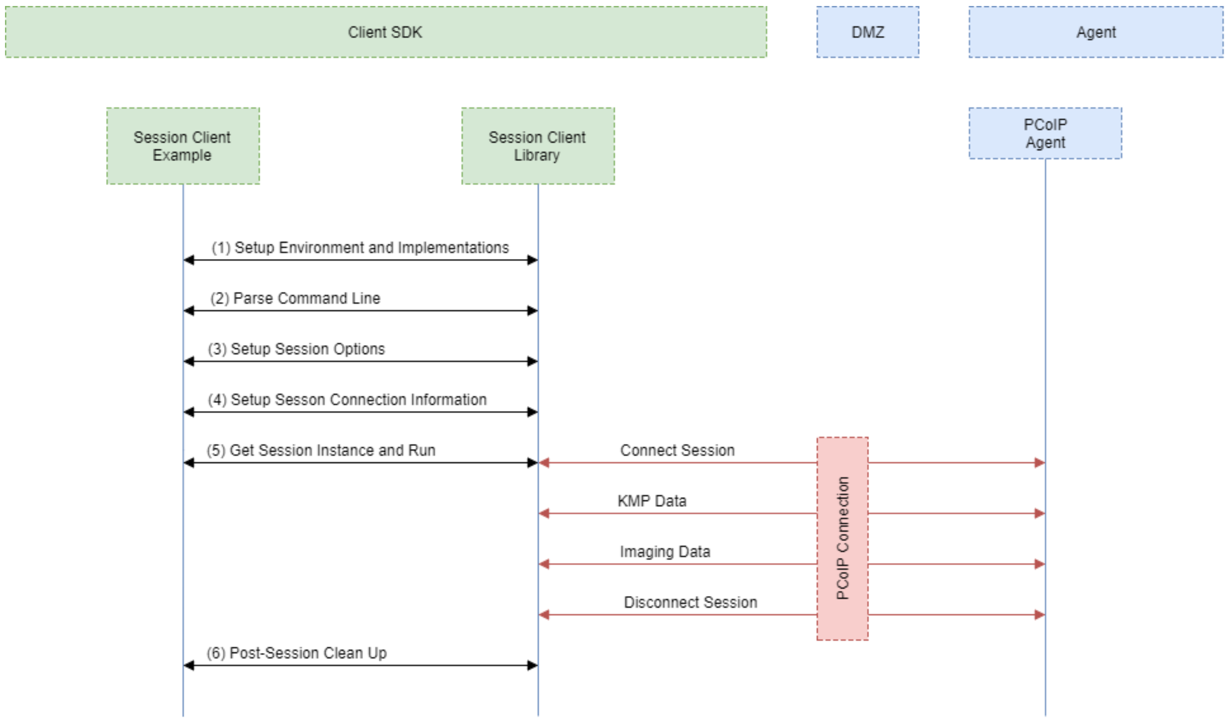
  Note: URI format documentation
  There is a document describing the URI format in the root of the SDK.

  ```
  open /Applications/ClientSession.appclient_session connect-
  ```

```
tag=SCS1WsopFJ3iz1l48PTJMXFkcD4b6M9aiakHXH3ellLhUROBceWAifSSn%2b4A
V1FC8IihWVmsISmYFKeA25AtzFrdMpdaCtqlic0zfxAA "teradici-
pcoip://10.64.60.115:4172?session-id=230584300921369396"
```

Using the Client Session API

Use of the Client Session C++ API is demonstrated in <*pcoip_client_sdk_session/modules/client_session/src/client_session_main.cpp*>. It includes the steps illustrated in the diagram below:



PCoIP Session Client API Sequence Diagram

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

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

Setting Up the Development Environment

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

To set up your client development environment:

1. Choose and populate a directory for third-party libraries, currently only OpenSSL is required.

   Either unpack `session_client_third_party*.tar.gz` into this location, or download and manually build the latest version of OpenSSL.

   Note: Library names and versions
   Library names and specific versions are encoded in the path names within the third-party tree.

   Note: OpenSSL version
   `session_client_third_party*.tar.gz` only contains the version of OpenSSL available at the time of distribution. You need to obtain and update any future security updates that may be issued.

2. Set an environment variable called `LOCAL_THIRD_PARTY` pointing to the location chosen in step 1. For OS-specific instructions, see *Windows Build Procedure* on page 46*Using a text editor, add the following lines to ~/.bash_profile:* on page 48.
3. Choose or create a working directory and unpack the SDK archive (`pcoip_client_sdk_windows*.<rev_info>`) into it. Choose or create a working directory and unpack the SDK archive (`pcoip_client_sdk_session_OSX.<rev_info>`) into it.

Tip: USB functionality in Windows client applications
If you will be incorporating USB devices into your Windows client application, you must also install the Client USB package. For instructions, see *Supporting USB Devices* on page 28.

The macOS application includes the required USB kernel drivers.

Updating SDK Components

Updating the SDK to a new version can be done by replacing the old binaries with new versions in place. There is no special upgrade path.

Upgrading components will not break compatibility with existing APIs.

Windows Prerequisites

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

- Python 2.7 Select 32-bit

  https://www.python.org/downloads/windows/

  The 32-bit version of Python must be used even when installing on a 64-bit OS. Make sure Python is in the system PATH.

- MS Visual Studio 2013 with Update 4.

  https://www.visualstudio.com/downloads/

- CMake 3.4.1

  https://cmake.org/download/

  Make sure CMake is in the system PATH; CMake does not automatically add itself to the system PATH during installation.

  Important: CMake version requirement
  The PCoIP Client SDK does not support versions of CMake higher than 3.4.3.

Building the SDK for Windows

This section includes instructions for building and packaging the SDK for Windows systems.

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.

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

Windows Build Procedure

1. From the Windows Control Panel, set an environment variable called `LOCAL_THIRD_PARTY` pointing to the third party libraries.

   Note: Windows 7 procedure
   This procedure is correct for Windows 7. Other versions of Windows may use slightly different methods to define environment variables.

   a. Open **System** in *Control Panel*.
   b. Click on **Advanced system settings**.
   c. On the *Advanced* tab, click **Environment Variables**.
   d. In the lower box labeled *System Variables*, click **New**.
   e. In the *Variable Name* field, type `LOCAL_THIRD_PARTY`.
   f. In the *Variable Value* field, enter the full path to root of the third-party libraries.
   g. Click **OK** on each of the three open dialogs.

2. From the Windows Control Panel, set an environment variable called `NO_SOFTSTORE` to 1.
3. Open a Windows cmd.exe shell and change directory to the root of the unpacked `pcoip_client_sdk_windows*.<rev_info>` source directory.
4. Run **python gencmake.py**. This will invoke CMake and generate the required Visual Studio 2013 solution and project files.
5. Open the `.\build\client.sln` solution file in Visual Studio 2013
6. Build the solution.

Packaging

All files needed to run the client, except the Microsoft Visual Studio 2013 redistributable, are contained in the `product\WINDOWS\x86\[Debug|Release]` directory.

macOS Build Prerequisites

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

- **Xcode 6.0 or 7.0 for macOS 10.11 and Xcode 7.3.1 for macOS 10.12**

  https://developer.apple.com/xcode/downloads/
- **macOS 10.7 SDK**

  For macOS 10.7 SDK, you must locate the SDK and install it into Xcode here:

  `/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs`
- **Python 2.7**

  https://www.python.org/downloads/mac-osx/
- **CMake 3.4.1**

  https://cmake.org/files/v3.4/

  Important: CMake version requirement
  The PCoIP Client SDK does not support versions of CMake higher than 3.4.3.

Building the SDK for macOS

This section includes instructions for building and packaging the SDK for macOS systems as discussed next:

1. Open a Terminal window and change directory to the root of the unpacked `pcoip_client_sdk*.<rev_info>` package.
2. Using a text editor, add the following lines to ~/.bash_profile:

   ```
   export LOCAL THIRD_PARTY=<full path to root of the third party
   libraries>
   ```

   ```
   export NO_SOFTSTORE=1
   ```

   This adds an environment variable called `LOCAL_THIRD_PARTY` and points it to the third party libraries. It also turns off soft store which is not necessary for this build.
3. Run **./gencmake.py**.

   This will invoke CMake and generate the required Xcode files.

Note: File permissions
You may need to change the file permissions on gencmake.py in order to run it.

4. Open the `./build/client.xcodeproj` project file in Xcode.
5. Build the project.

   For a debugging build, use **ALL_BUILD** > **My Mac (64-bit).**

   For a release build, use **Xcode Menu** > **Product** > **Build For** > **Profiling.**

Debugging a Crash in Windows

This section describes the files and processes required to debug application crashes in Windows.

The Application Dump File

If a crash happens within `broker_client_example.exe`, an application generates a dump file with the .dmp extension.

Client application .dmp files are saved in the folder where all log files are stored. The path to the log files is `%LocalAppData%\Teradici\PCoIPClient\logs` for the session client. The .dmp file can also be saved manually during a debugging process in Visual Studio.

When the Visual Studio debugger stops at an exception, close the pop-up window, choose **Save Dump As** under the *DEBUG* tab, and save the dump file.

Note: Debugging crashes in client_sessionclient_session.exe
The SDK does not include files for debugging `ClientSession.appclient_session` crashes. If a crash happens in `ClientSession.appclient_session`, the `.dmp` file should be sent to Teradici for analysis.

Files Required for Debugging

- .exe files
- .dll files
- .pdb files
- .dmp file
- source files

The PCoIP Client SDK contains the `.dll` files and the `.pdb` files. The `.exe` file that caused the crash should also be in the same directory as the `.dll` files. You must use the `.pdb` files that were generated when the `.exe` and `.dll` files were built.

Debugging a Crash

To debug a crash in broker_client_example.exe:

1. Open Visual Studio 2013.
2. Select **File** > **Open** > **File**. The file selection dialog appears.
3. From the file selection dialog, browse to .dmp file and open it.

The file opens, all symbols are linked, and the minidump file summary window appears.

Tip: Setting the symbol path
If the `.pdb` files are not found, the path should be specified in the `_NT_SYMBOL_PATH` environment variable, or, locate the *Actions* box and choose **Set symbol paths** at the top-right corner of the summary page, and configure the path there.

4. Locate the *Actions* box at the top-right corner of the summary and choose **Debug with Native Only** to start debugging.
5. If Visual Studio cannot find the path to the source files, it will present an alert dialog and allow you to manually specify the path. If this occurs, provide the actual path to the source files.

   If you don't have the files, click **Cancel** to proceed without them.
6. Find the location of the crash under the *Call Stack* tab in the Visual Studio debugging window.

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.

Note that it is now possible to brand the in-session client using the branding tool see *Customizing the PCoIP Session* on page 14.

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 customPCoIP Virtual Channel plugins which stream data between clients and hosts. For more information, see the*PCoIP® Virtual Channel Software Development Kit 1.1 Developers' Guide*.

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

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

Q: **Is consulting offered by Teradici for the Client SDK?**

A: Cloud Access Platform customers receive support as described in the general support terms of your agreement. For additional consulting services, contact Teradici.

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.

Teradici Corporation

#301-4601 Canada Way, Burnaby, BC V5G 4X8 Canada

phone +1.604.451.5800 fax +1.604.451.5818

www.teradici.com

Intellectual Property Rights

Teradici Corporation
#101-4621 Canada Way, Burnaby, BC V5G 4X8 Canada
phone +1.604.451.5800 fax +1.604.451.5818
www.teradici.com