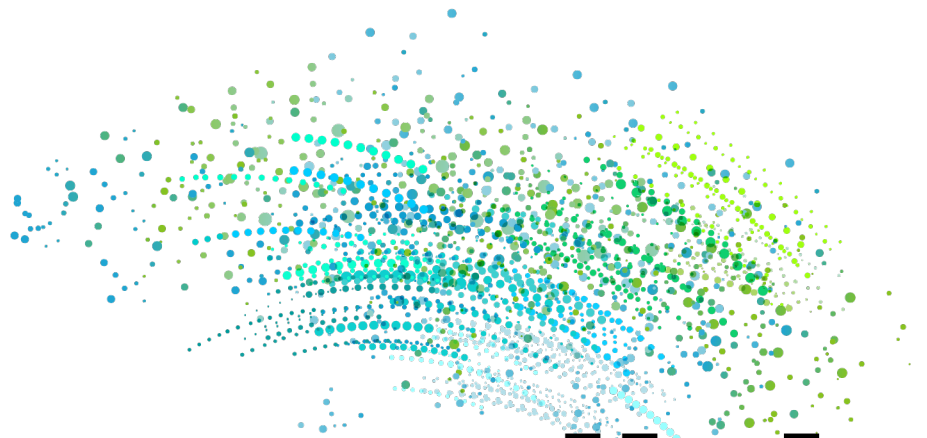


# PCoIP® Client Software Development Kit for Windows

Version 3.3

---

Developers' Guide



**teradici**®

TER1608003-3.3

# Contents

<b>Who Should Read This Guide</b> .....	<b>4</b>
<b>What's New in this Release?</b> .....	<b>5</b>
Enhanced Audio and Video Synchronization (Technology Preview) .....	5
Automatic Uninstall of Previous Versions .....	6
<b>About PCoIP Sessions</b> .....	<b>7</b>
System Actors .....	7
PCoIP Session Phases .....	7
About Brokered and Non-Brokered Connections .....	8
<b>About the PCoIP Client Software Development Kit for Windows</b> .....	<b>9</b>
Supported Build Platforms .....	9
What Can You Build With the PCoIP Client SDK? .....	9
<b>Customizing the PCoIP Session</b> .....	<b>11</b>
Disable Session Menu Bar Visibility .....	11
Disable Hot Keys .....	11
Windowed or Fullscreen Mode .....	11
Set Host Resolution .....	12
Custom Client Branding .....	12
Image Scaling .....	12
Maintain Aspect Ratio .....	13
USB Auto-Forward .....	13
USB Vendor ID/Product ID Auto-Forward .....	13
Locale .....	13
Log Level .....	14
Log File Name .....	15
<b>Setting Up a PCoIP Agent Test Environment</b> .....	<b>16</b>
Connecting To Your PCoIP Test Agent .....	16
Establishing a PCoIP Connection Using a Teradici PCoIP Software Client .....	17
<b>Setting Up the Development Environment</b> .....	<b>19</b>
Updating SDK Components .....	19
Windows Prerequisites .....	19
<b>Using the Broker Client Example</b> .....	<b>21</b>
Establishing a New PCoIP Session .....	22
Automatically Passing Session Parameters to the Session Client .....	23
Passing Additional Parameters to the Session Client .....	24

<b>Using the Session Client</b> .....	<b>25</b>
<b>Supporting USB Devices</b> .....	<b>26</b>
Manually Bridging USB Devices .....	26
Updating the Client USB Package .....	26
Exit Codes for Programmatic USB Installations and Uninstallations .....	27
Installing the Client USB Package for Windows .....	28
<b>The Example Broker Client</b> .....	<b>30</b>
The Example Broker Client Sequence .....	30
PCoIP Session-Creation Steps and Actors .....	31
How to Establish a PCoIP Session .....	34
<b>Building the SDK for Windows</b> .....	<b>37</b>
Configuring CMake for PDB Files .....	37
Windows Build Procedure .....	37
Packaging .....	38
<b>Debugging a Crash in Windows</b> .....	<b>39</b>
The Application Dump File .....	39
Files Required for Debugging .....	39
Debugging a Crash .....	39
<b>Frequently Asked Questions</b> .....	<b>41</b>

## 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 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 Windows](#)
- [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 Windows:

### Enhanced Audio and Video Synchronization (Technology Preview)



**Warning: This information is subject to change**

This preview program involves pre-release software and rapidly evolving technology. Features at technology preview may have stability, compatibility or other issues. Technology preview releases are for early adopters and are made available for daily use, development, or compatibility testing.

In general, be alert for the following types of issues:

- The user interfaces described in this document may change. Fields and buttons shown here may not exist or may have a different appearance by the time you see them.
- As Teradici and its technology partners progress towards an official release, compatibility requirements and prohibitions will change. This should normally trend towards increasing stability and expanded compatibilities, but regressions are possible.
- Unexpected issues or conflicts may cause this process to break. You may or may not see a helpful error message if this happens. Administrators with a sophisticated understanding of virtual machine deployments in general, and Azure deployments in particular, may be able to overcome some of these issues.

This document may be updated as these procedures evolve.

There are improvements in video playback, a reduction in delays between the audio and video channels, and improved lip synchronization for movie playback. This feature is disabled by default. To enable the audio and video synchronization technology:

1. Navigate to %APPDATA%\Teradici.
2. Add the following line to the Teradici PCoIP Client.ini file: `enable_enhanced_avsync=1`.
3. Restart the client.

To disable this feature, remove the line from the **Teradici PCoIP Client.ini** file.

## Automatic Uninstall of Previous Versions

This feature enables the client installer to uninstall the previous client before installing a new version.

## 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.

### 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. For a step-by-step walkthrough of the pre-session phase, with examples, see [The Example Broker Client Sequence on page 30](#).

- **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 11](#).

## 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.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.



# About the PCoIP Client Software Development Kit for Windows

The PCoIP Client Software Development Kit (SDK) for Windows 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:

- Libraries
- Programming Guide
- Code Examples

## Supported Build Platforms

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

### Windows

- Windows 10 (64-bit)
  - Windows 7 (32-bit and 64-bit)
- 

## 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 *session* phases of a PCoIP connection. For example, the following customizations are all typical use cases:

Pre-Session Customizations	Session Customizations
<ul style="list-style-type: none"> <li>• Customize the client user interface to create a branded, end-to-end solution using company assets such as:               <ul style="list-style-type: none"> <li>◦ Corporate logos</li> <li>◦ Slogans, trademarks, or other text</li> <li>◦ Corporate colors and iconography</li> </ul> </li> <li>• Developing customized authentication workflows, either directly or using a broker</li> <li>• Automatically connecting users to specific desktops or applications, based on an identified user type or task</li> <li>• Embedding a remote workload into an application</li> </ul>	<ul style="list-style-type: none"> <li>• Client branding, including:               <ul style="list-style-type: none"> <li>◦ Menu item labels</li> <li>◦ Window titles</li> <li>◦ Application icons</li> <li>◦ Company logos</li> </ul> </li> <li>• Automatic bridging of USB devices</li> <li>• In-session menu bar visibility</li> <li>• Disabling hot keys</li> <li>• Client display size in windowed and full-screen mode</li> <li>• Configuring resolutions</li> <li>• Securely using local and bridged USB devices</li> </ul>

For even more complex deployment scenarios, such as adding support for custom peripherals or embedding third-party ecosystems, 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](#).

# 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
- Locale
- Log Level
- Log File Name



## 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) 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 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. 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`.

## 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.

## 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:

## 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.

## Log Level

`log-level` is the log level parameter. It is possible to over-ride the default log-level by specifying a different log-level parameter. The following parameters apply:

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



**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.

# Setting Up a PCoIP Agent Test Environment

Before developing your custom client, you must 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 2.10 for Windows Administrators' Guide](#)
- [Teradici PCoIP® Graphics Agent 2.10 for Windows Administrators' Guide](#)
- [Teradici PCoIP® Standard Agent 2.10 for Linux Administrators' Guide](#)
- [Teradici PCoIP® Graphics Agent 2.10 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](#)

## 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:

Host Address: 123example NEXT

\*Connection Name: Setup SAVE

*\*optional: a name to identify your profile*

Connect to PCoIP Remote Workstation card

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

example.com example \*\*\*\*\* LOGIN

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.

### Pre-Session Connection

Instructions for configuring and connecting with PCoIP Software Clients can be found in the [Teradici PCoIP<sup>®</sup> Software Clients 3.3 Administrators' Guide](#).

# 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.
3. Choose or create a working directory and unpack the SDK archive (`pcoip_client_sdk*.<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 26](#).

## 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.

# 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 example broker client:

- 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.



**Info: About login\_info.txt**

The example broker client 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.

## Establishing a New PCoIP Session

To establish a new session 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 establish a new PCoIP 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%2fM7mxEE52au%2f3Jedtgp16L%2frA8iB00%2bEr%2bYJd0yIL0xd9Mv5V0CDLSDmUNkOCwyyV1%2bu3w1aA7hXxEWmzhAA"
```

6. Take the URI line and the connect-tag parameter from the status message, and pass them to the session client. The URI contains the IP address, the port number, the session ID, and the SNI; the connect tag is in the PARAMETERS string.

Concatenate the URI and connect tag and call `client_session.exe` with them as shown next:

```
# client_session.exe teradici-pcoip://10.64.60.147:4172?session-id=2305843009213693954&sni=SAL-W7P64-SA15 connect-tag=SCS1fw0Zbk%2bEu7q2iz0%2fM7mxEE52au%2f3Jedtgp16L%2frA8iB00%2bEr%2bYJd0yIL0xd9Mv5V0CDLSDmUNkOCwyyV1%2bu3w1aA7hXxEWmzhAA
```



### Important: Broker client example invocation limits

The following restrictions apply each time you invoke the broker client example:

- You have 60 seconds to pass the connect tag to the session client.
- You can only invoke the session client with a given session tag once.

## Automatically Passing Session Parameters to the Session Client

You can also use the `l` switch (lowercase L, for launch) to automatically invoke the session client with the supplied parameters. 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 `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. Run the command line client, providing the `login_info.txt` file as an argument:

```
broker_client_example.exe login_info.txt 1
```

## Passing Additional Parameters to the Session Client

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

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

```
broker_client_example.exe login_info.txt 1 "disable-menubar"
```



## Using the Session Client

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

- `<working directory>\pcoip_client_sdk\product\windows\x86\Release\client_session.exe`

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.

**To view a list of supported in-session Windows client options:**

1. Open a command line prompt.
2. Change directory to `product\<os>\<arch>\<release|debug>`.
3. Type `client_session help`.

## 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 Windows PCoIP agents

For information about USB bridging configuration on Windows PCoIP agents, see one of the following guides:

- [Teradici PCoIP® Graphics Agent 2.10 for Windows Administrators' Guide](#)
- [Teradici PCoIP® Standard Agent 2.10 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 28](#).



**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 occurred. 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 occurred. 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\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` 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\product\windows\x86\Release\usb\PCoIP_Client_USB_installer.exe  
/noreboot /autoclose /D="C:\Program Files (x86)\Teradici\PCoIP Client"
```

## The Example Broker Client

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 Example Broker Client Sequence

This section describes how the example broker client 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.

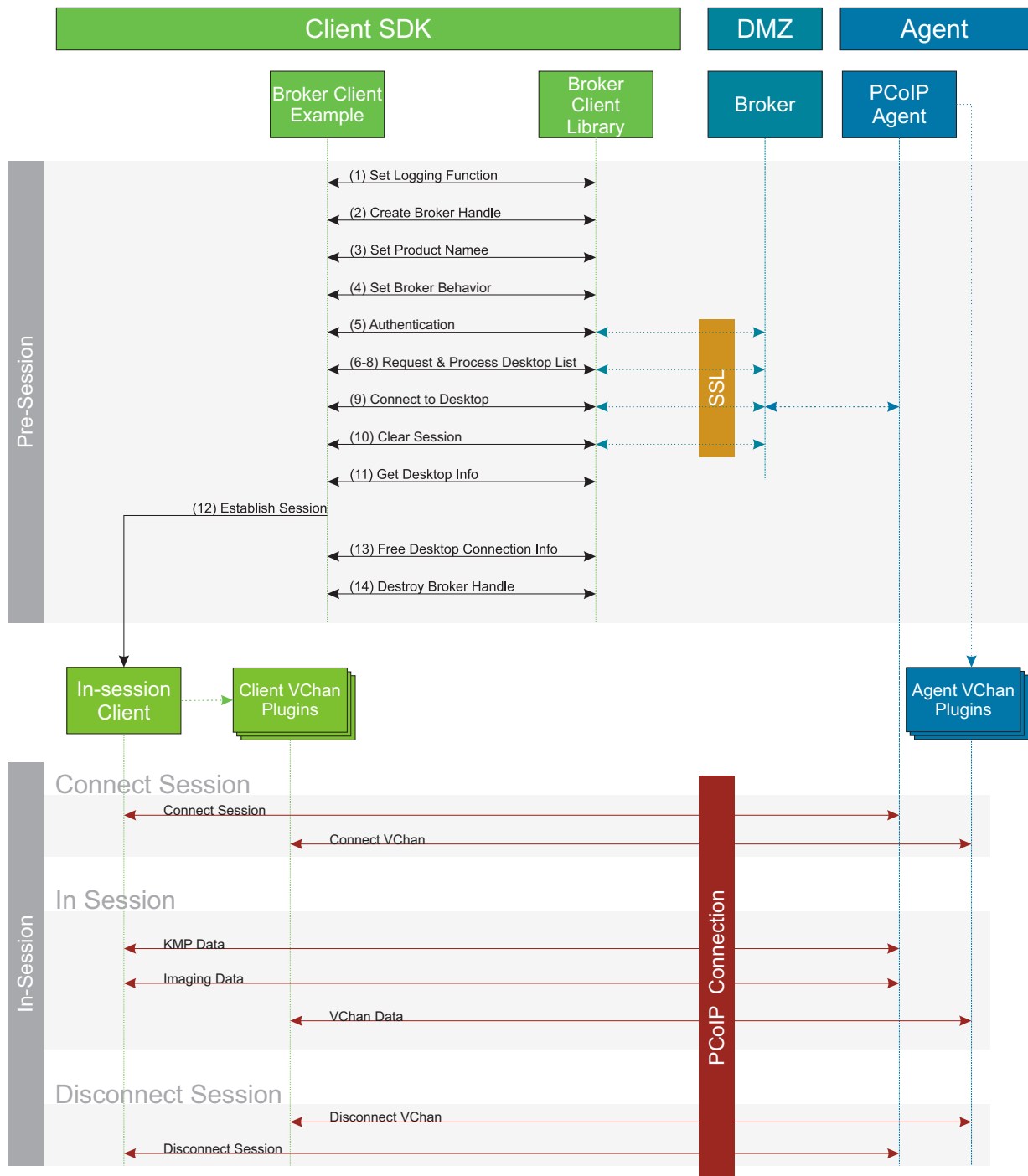
The example C++ code can be found in the SDK package located here:

```
pcoip_client_sdk/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 long 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 34](#).

**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 `client_session.exe`:

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

- **Connection tag:**  
`SCS1WsopFJ3iz1148PTJMXFkcD4b6M9aiakHXXH3ellLhUROBceWAifSSn%2b4AV1FC8IihWVmsISmYFKeA25AtzFrdMpdaCtqliC0zfxAA`
- **IP address:** `10.64.60.115`
- **Session ID:** `2305843009213693961`

```
client_session.exe -i connect-tag=
SCS1WsopFJ3iz1148PTJMXFkcD4b6M9aiakHXXH3ellLhUROBceWAifSSn%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 `client_session.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:**  
`SCS1WsopFJ3iz1148PTJMXFkcD4b6M9aiakHXXH3ellLhUROBceWAifSSn%2b4AV1FC8IihWVmsISmYFKeA25AtzFrdMpdaCtqliC0zfxAA`
- **URI:** `"teradici-pcoip://10.64.60.115:4172?session-id=230584300921369396"`



### Note: URI format documentation

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

```
client_session.exe connect-
tag=SCS1WsopFJ3iz1148PTJMXFkcD4b6M9aiakHXXH3ellLhUROBceWAifSSn%2b4A
```

```
V1FC8IihWVmsISmYFKeA25AtzFrdMpdaCtqliC0zfxAA "teradici-  
pcoip://10.64.60.115:4172?session-id=230584300921369396"
```

## 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](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. Open a Windows cmd.exe shell and change directory to the root of the unpacked `pcoip_client_sdk*.<rev_info>` source directory.
3. Run **python gencmake.py**. This will invoke CMake and generate the required Visual Studio 2013 solution and project files.
4. Open the `.\build\client.sln` solution file in Visual Studio 2013
5. 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.

# 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_session.exe`

The SDK does not include files for debugging `client_session.exe` crashes. If a crash happens in `client_session.exe`, 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 11](#).

**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 `client_session`.

**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. 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

#101-4621 Canada Way, Burnaby, BC V5G 4X8 Canada

phone +1.604.451.5800 fax +1.604.451.5818

[www.teradici.com](http://www.teradici.com)

---

**teradici**<sup>®</sup>

The information contained in this documentation represents the current view of Teradici Corporation as of the date of publication. Because Teradici must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Teradici, and Teradici cannot guarantee the accuracy of any information presented after the date of publication.

This document is for informational purposes only. TERADICI MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Teradici Corporation.

Teradici may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Teradici, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property. Visit [Notice of Intellectual Property Rights](#) for more information.

© 2004-2017 Teradici Corporation. All rights reserved.

Teradici, PC-over-IP, and PCoIP are trademarks of Teradici Corporation and may be registered in the United States and/or other countries. Any other trademarks or registered trademarks mentioned in this release are the intellectual property of their respective owners.