Cisco Remote Expert Mobile Version 11.5(1)

Developer's Guide

First Published: 2016-08-10
Last Modified: 2016-12-15

Cisco Systems, Inc. www.cisco.com

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

All printed copies and duplicate soft copies are considered un-Controlled copies and the original on-line version should be referred to for latest version.

Preface

Change History

Changes	Date
Initial release	2016-08-10
Updated Features section New section for Co-browse only (with code) added	2016-10-31
New sections under iOS and Android sections for starting co-browse only session	2016-11-1
Moved section Using UUI in CSDK for Web Added section Establishing RE Mobile Session to include previous sections With Voice and Video, Co-browse only (with correlation ID) and Co-browse only (with code) as subsections.	2016-11-15

About this guide

This document outlines the steps to develop mobile and web applications that leverage Cisco Remote Expert Mobile (RE Mobile).

Developers using this guide should have experience in JavaScript, Objective C or Java depending on the application type.

- Web—It is assumed that the developer will have a familiarity with JavaScript, HTML and CSS.
- iOS—It is assumed that the developer will have a familiarity with iOS, Xcode and Objective-C
- Android—It is assumed that the developer will have a familiarity with Android, Java, and the Android SDK

This guide also assumes that you are familiar with basic contact center and unified communications terms and concepts.

Successful deployment of Remote Expert Mobile also requires familiarity with the information presented in the *Cisco Collaboration Systems Solution Reference Network Designs (SRND)*. To review IP Telephony terms and concepts, see the documentation at the preceding link.

Organization of This Guide

This guide includes the following sections:

Introduction	Introduction and brief overview of Remote Expert Mobile.
Technologies	Description of the technologies used by Remote Expert Mobile.
Developer Overview	Describes the general components of the solution, provides an understanding and sequencing for Establishing a RE Mobile Sessions, Using Anonymous and Restricted Client Access as well as invoking co-browse only mode.
CSDK Security Highlights for Developers	A general overview of security concerns for developers.
Remote Expert Mobile—CSDK for Web (JavaScript)	Quick start and specifics for embedding CSDK in web applications. Also includes details for masking and hiding sensitive information within co-browse sessions.
Remote Expert Mobile—CSDK for iOS (Objective C)	Quick start and specifics for embedding CSDK in Apple iOS applications. Also includes details for masking and hiding sensitive information within co-browse sessions.
Remote Expert Mobile—CSDK for Android (Java)	Quick start and specifics for embedding CSDK in Android applications. Also includes details for masking and hiding sensitive information within co-browse sessions.
Acronym List	Lists some common industry and Cisco-specific acronyms relevant to Remote Expert Mobile.

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, using the Cisco Bug Search Tool (BST), submitting a service request, and gathering additional information, see *What's New in Cisco Product Documentation* at: http://www.cisco.com/c/en/us/td/docs/general/whatsnew/whatsnew.html.

Subscribe to *What's New in Cisco Product Documentation*, which lists all new and revised Cisco technical documentation, as an RSS feed and deliver content directly to your desktop using a reader application. The RSS feeds are a free service.

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco website at www.cisco.com/go/offices.

Documentation Feedback

To provide comments about this document, send an email message to the following address: contactcenterproducts_docfeedback@cisco.com.

We appreciate your comments.

Conventions

This document uses the following conventions.

Convention	Indication
bold font	Commands and keywords and user-entered text appear in **bold** font.
italic font	Document titles, new or emphasized terms, and arguments for which you supply values are in *italic* font.
[ ]	Elements in square brackets are optional.
{x | y | z }	Required alternative keywords are grouped in braces and separated by vertical bars.
[x | y | z]	Optional alternative keywords are grouped in brackets and separated by vertical bars.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.
`courier font`	Terminal sessions and information the system displays appear in `courier` font.
< >	Nonprinting characters such as passwords are in angle brackets.
[ ]	Default responses to system prompts are in square brackets.
!, #	An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.

Introduction

Cisco Remote Expert Mobile is a software solution that enables personal and actionable customer interactions within mobile and web applications. These interactions range from simple click-to call to a complete voice, video and Expert Assist customer engagement session interconnected to a full contact center environment. For example, Cisco Remote Expert Mobile can connect individual investors to the next available financial advisor within a mobile trading app (B2C—Business to Consumer) or a field employee's mobile app routing into an internal helpdesk (B2E—Business to Employee).

Features

With Cisco Remote Expert Mobile developers can deliver voice, video and Expert Assist co-browse and application sharing in mobile or web applications. Cisco Remote Expert Mobile is designed specifically for remote collaboration services provided through Cisco Unified Communications Manager, Cisco Unified Contact Center Enterprise (Unified CCE) and / or Cisco Unified Contact Center Express (Unified CCX). Remote Expert Mobile offers the following features and options that are pre-sized within core components. Core component features are:

- In-app voice and video communications (Over-the-Top WebRTC communications)
  - High definition video and audio
  - Bi-directional or one-way video
  - Mute audio, video or both
  - Client side call control
- WebRTC to SIP gateway (trunking into Cisco Unified Border Element and Unified Communications Manager)
- Expert Assist
  - Web co-browse
  - Mobile app sharing
  - Remote app control
  - Expert form editing and completion
  - Annotation by expert
  - Expert document push

- Expert URL sharing
- Protect sensitive data with field and data masking

- Media Handling:

  - STUN server (RFC 5389) for client external IP identification
  - UDP port multiplexing
  - Media encryption / decryption
  - Bidirectional audio
  - High definition video (H.264 or VP8 in CIF (352x288), nHD (640x360), VGA (640x480), 720p (1280x720)
  - High definition and narrowband audio codec support (Opus, G.711 ulaw or G.711 alaw)
  - Opus, G.711 ulaw, G.711 alaw and G.729a audio transcoding into the enterprise network
  - H.264 and VP8 video transcoding

Remote Expert Co-browse

With Cisco Remote Expert Co-browse (previously known as Meet Me), developers can deliver Expert Assist co-browse and application sharing in mobile or web applications. In this case, the key components are:

- Expert Assist (with all its elements as above)

In this case, there should be an existing communication channel between the two parties (consumer and agent), which might be a voice and video call, a chat session, or anything else.

SDKs

Cisco Remote Expert Mobile includes Software Development Kits (SDKs) to provide voice over IP, video over IP and expert assist (app share and web co-browse, annotation and document push) features within pre-existing mobile and web applications. Whether placing or receiving calls in client web applications, RE Mobile's Client SDK for Web supports every major browser such as: Google Chrome, Mozilla Firefox, Opera, Internet Explorer (Windows desktop only, not tablet) and Apple Safari. With WebRTC at its core, in-app communications are enabled without the need for plugins. Where WebRTC is yet to be supported in Internet Explorer and Safari, WebRTC plugins are provided for voice and video. Cisco Remote Expert Mobile also delivers integrated communications in iOS and Android apps through native libraries.

Technologies

WebRTC

WebRTC is a standards-based approach for enabling real time communications through a common set of APIs. These APIs were created as part of HTML5 standards and are simple for web developers to embed communications within web sites and mobile applications without knowing the complexities of Voice over IP. WebRTC defines a way for browsers and mobile apps to implement technologies like video conferencing in a way that is both interoperable with other clients and does not require the use of a plugin. WebRTC leverages a variety audio and video codecs such as G.711, Opus, H264 and VP8.

Expert Assist

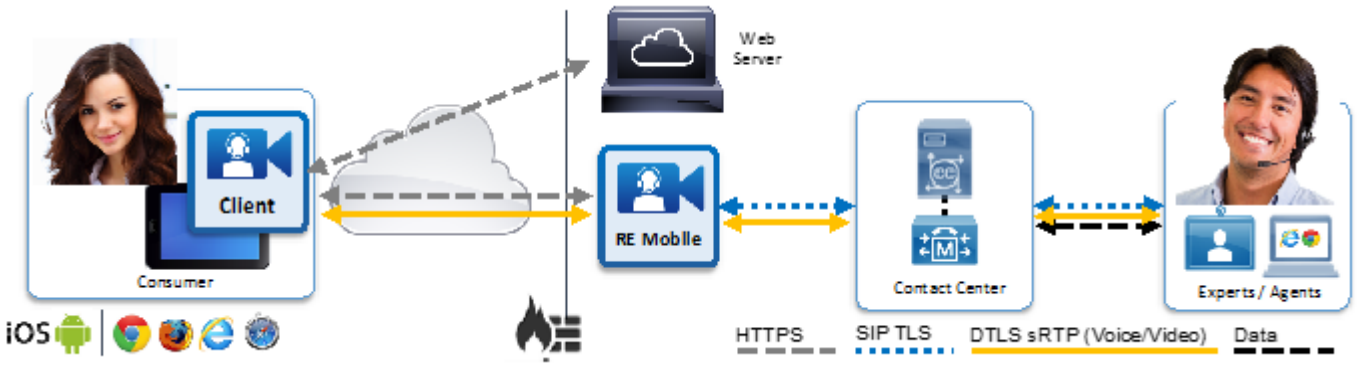
With Expert Assist, the remote user of an application can share the screen of their tablet, smartphone or browser tabs with an expert. Experts are more knowledgeable of the problem at hand and provide guidance through to successful task completion. And for sensitive information, fields and regions of a web page or application can be masked to shield the agents view.

The expert can also control the app or web site of the user through simple point and click. Remote control allows the advisor to traverse through menus, jump to specific information, complete a form or walk others through an important process. The expert can also move the live video window to ensure it doesn't interfere with elements of the screen.

Unlike most co-browsing technologies, Expert Assist **does not** share the Document Object Model (DOM) between the user and the expert. Expert Assist technologies ensure that inconsistencies between browsers are not reflected during a session. In addition, Expert Assist supports native iOS and Android apps.

Developer Overview

With Remote Expert Mobile, consumers are connected to an Expert or Agent. Developers embed the Remote Expert Mobile Client SDKs (CSDK) in a native iOS, native Android or web based application code.



Note: For detail architecture diagrams, protocol flows please refer to the "Cisco Remote Expert Mobile Design Guide"

■ **Consumer**—A consumer is the user of that connects to an Expert. The Client Application is installed on the consumer's mobile iOS or Android phone or tablet or accessed via their web browser.

■ **Client or Client Application**—With only a few lines of code, a developer can integrate WebRTC voice/video and Expert Assist into a new or existing application. These Client Applications (Clients) can then connect Consumers to Experts with voice over IP, video over IP and expert assist (app share and web co-browse, annotation and document push) features within pre-existing mobile and web applications. Each connection from a client to an expert is referred to as a session or call.

Whether placing or receiving calls, Cisco Remote Expert Mobile supports every major browser.. With WebRTC at its core, in-app communications are enabled without the need for plugins. Where WebRTC is yet to be supported in Internet Explorer and Safari, WebRTC plugins are provided. Cisco Remote Expert Mobile also delivers integrated communications in iOS and Android apps thru native libraries.

- CSDK for Web - JavaScript for web browser applications (Chrome (Desktop and Android), Firefox (Desktop and Android), Opera (Desktop and Android), Microsoft Internet Explorer (Windows desktop only, not tablet) and Apple Safari (Desktop)
- CSDK for iOS - Objective C for Apple iOS native applications (iPhone, iPad, iPod touch)
- CSDK for Android - Java for Android (Phone, Phablet and Tablets)

■ **Remote Expert Mobile (RE Mobile)**—RE Mobile also includes server software components that run on the 'company standard' Virtual Machine (VM) hardware platform for ease of management and deployment within an existing data center. These server component provide the call signaling to establish calls, the ability to handle voice, video and expert assist media as well as WebRTC and associated firewall-traversal.

- Remote Expert Mobile Application Server (REAS)
- Remote Expert Mobile Media Broker (REMB)

■ **Contact Center**—Typically calls from client applications, passing through the RE Mobile server software are then placed in queue as part of a contact center solution, such as Cisco Unified Contact Center Enterprise (Unified CCE) or Cisco Unified Contact Center Express (Unified CCX) and a UC solution such as Cisco Unified Communications Manager (Unified CM). The contact center solution then routes the call to the best and available expert to handle the consumer's issues.

■ **Expert or Agent**—Experts are connected to consumers. Experts and contact center agents typically have an agent desktop application such as Cisco Finesse and a VoIP or video capable endpoint (ex. Cisco DX70). With the agent desktop and endpoint, the expert resolves consumer issues by communicating with the consumer and viewing the consumer's mobile app or browser. Experts and contact center agents may take advantage of Remote Expert Mobile within their Finesse agent user interface or use the Expert Assist Web Agent Console in UC specific deployments.

Quick JavaScript Example Overview

CSDK for Web allows a consumer on a web site to be connected over their Wi-Fi or Internet connection to an agent or expert and engage in live communications. Developer's can easily embed voice, video and or Expert Assist sessions in their web application with JavaScript. While there are several options that can be configured through various parameters, for the most part developer's will require two simple lines of JavaScript code:

#1—including the `assist.js` JavaScript library to enable the website:

```
<script src=" https://<REAS
IP>:8443/assistserver/sdk/web/consumer/assist.js"></script>
```

#2—establish an Expert Assist session in connection with a link or image:

```
<a title="Expert Assist" onclick="AssistSDK.startSupport({destination :
'agent1'})"><img src="img/support.png"></a>
```

Once the image is clicked, the consumer call will be routed to an agent who has logged in as 'agent1' into the Expert Assist Web agent console or Finesse gadget. More details for the JavaScript CSDK is provided in "Remote Expert Mobile—CSDK for Web (JavaScript)".

Note: This starts a call with voice and video. For co-browse only sessions, see Co-browse only (with code) or Co-browse only (with correlation ID)

Establishing a RE Mobile Session

The application can establish a session for voice and video to include co-browsing, or it can establish a session for co-browsing only. The difference between establishing a co-browse only session and establishing a session with voice and video is a matter of how the session is started. In both cases, the application calls `startSupport`, but provides different parameters in the configuration object for the call, using whatever way of creating the configuration object is appropriate for the platform.

Note: Once established, voice and video sessions and co-browse only sessions are identical as far as the Expert Assist developer is concerned. All the Expert Assist functionality (co-browsing, document push, annotations, screen sharing, remote control, and form completion) is available in the same way using the same API functions. For control of the voice and video call in a voice and video session, see the Advanced Developers Guide.

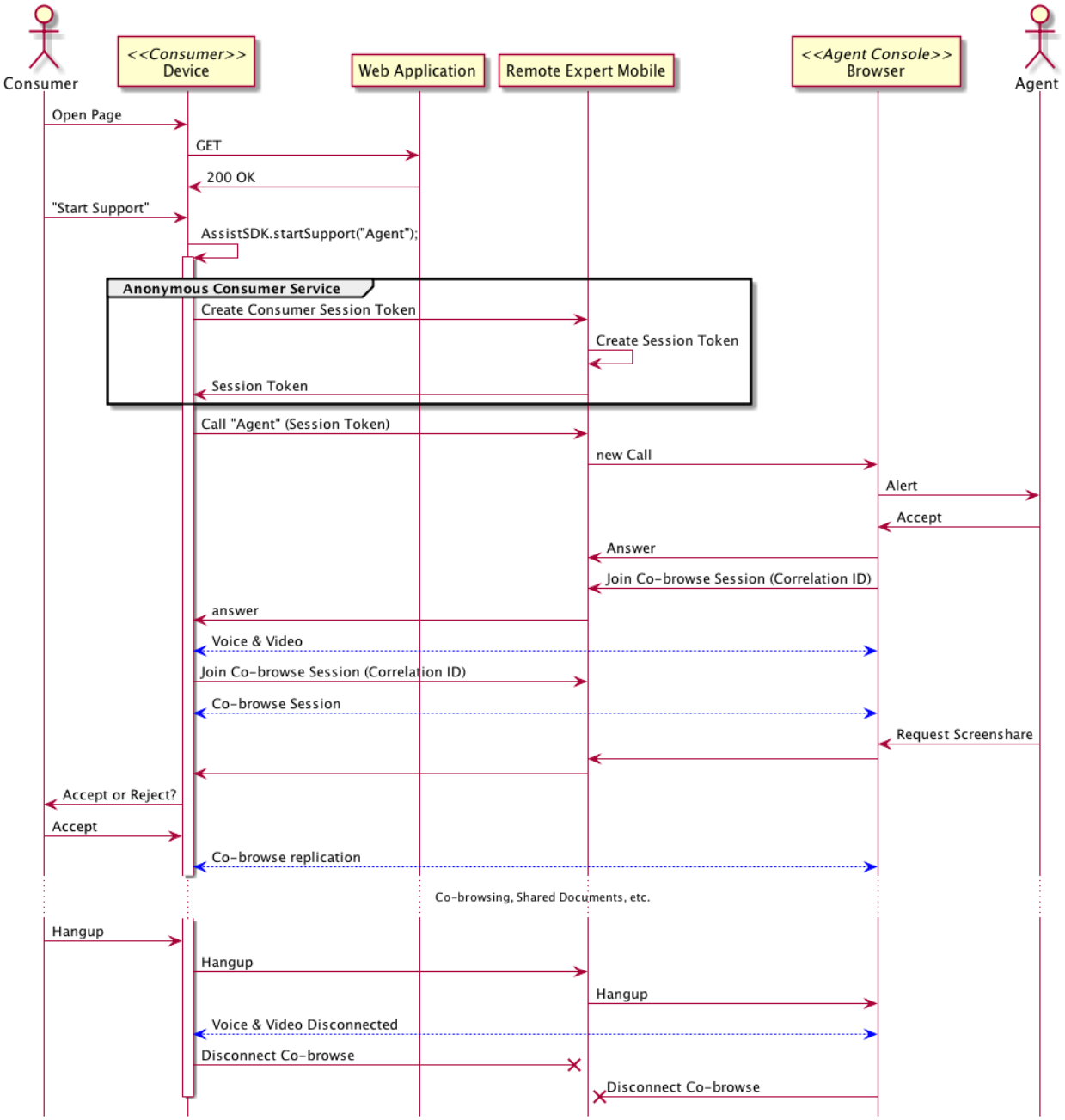
With Voice and Video

This section describes the sequence of events that establishes a voice and video support session between a Consumer and an Agent when using the integrated anonymous access.

Note: For increased Security, Anonymous Agent Access is disabled by default. Please refer to the Install and Configuration guide for details on how to enable Anonymous Agent Access

When voice and video is enabled, the call is used to transport an identifier that correlates the Consumer and the Agent so they can join the same co-browse session automatically (Correlation ID). The Correlation ID is transported as the username part of the Consumer's *From* SIP address and allocated by the CSDK.

When an Agent is available, the Consumer can request support as the following sequence diagram depicts:



Trusted Anonymous Consumer Access Mode

UUI data can be passed using the uui property. It requires Trusted Anonymous Consumer Access Mode. Please note the following:

- *Trusted* Anonymous Consumer Access mode should only be used where it can be asserted/trusted that the client (Web/iOS/Android) has not been tampered with, or where attempts to modify the Correlation ID and/or UUI have no negative impact to the system or other users of that system. As Javascript is in plain text in the browser, it is open to abuse and as such sensitive values such as the Correlation ID and UUI could be modified by a malicious user; the issue is less likely on iOS and Android as they're compiled binaries but the risk should not be discounted completely. Anonymous Consumer Access is in Enabled mode by default.

- UUI is typically used to transfer Customer specific information such account numbers, username, etc. The correctness/validity of this value should only be relied upon if the server-side application is managing the Session Tokens, or due-diligence has been undertaken for enabling trusted mode and either it is deemed secure enough (e.g. in a private network, used by *trusted* employees) or where the correctness of the UUI is not critical.

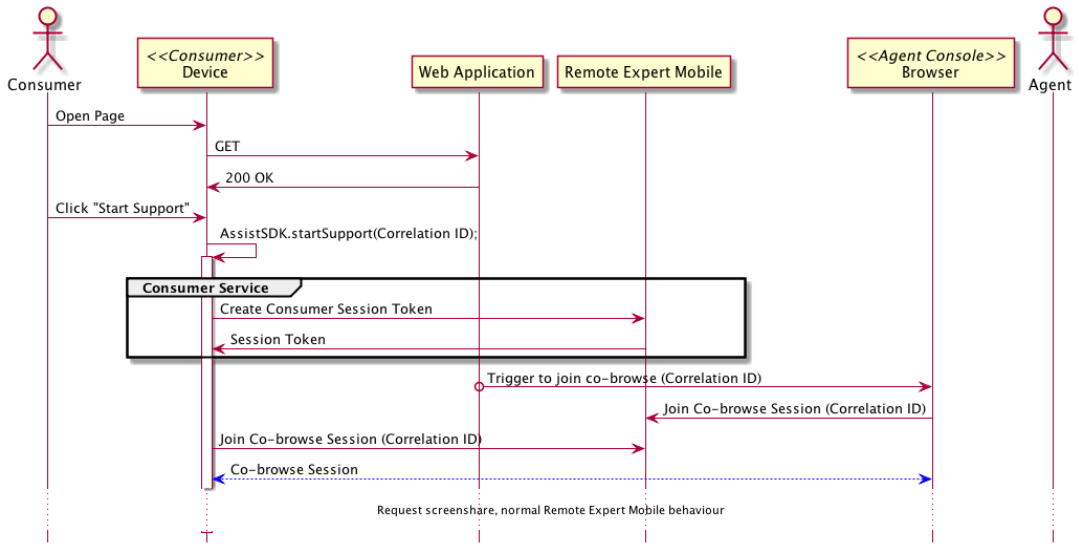
Co-browse only (with correlation ID)

CSDK can be used for co-browsing with the voice and video disabled. When only co-browsing is required, a call is not present and as such it is the responsibility of the developer to decide how to allocate a Correlation ID and signal it, if necessary, to the appropriate Agent; this typically requires deeper integration into an existing environment.

The correlation ID is included in the configuration provided to the `startSupport` call (see the sections on the specific SDs for information on how to do this, because the details differ between JavaScript, Android, and iOS SDKs).

Initiation

The following diagram provides a generalised sequence of events to establish a co-browsing session.



In the sequence diagram above, the Web Application that triggers the Agent to join a co-browse session; however, this signal may come from somewhere else within the infrastructure and is only shown here for illustrative purposes.

When selecting a suitable Correlation ID the following must be considered:

— **Uniqueness**—Different Customers must not be using the same Correlation ID at the same time.

— **Randomness**—The value should be suitably random and difficult to guess. As the *CSDK* does not authenticate a user, if the Correlation ID was easily guessable it could be possible for an attacker to guess another Consumer's Correlation ID and eavesdrop on their co-browse session.

Destruction

Both the Agent and Consumer can Join and Leave the co-browsing session independently of each other; the co-browsing session will remain open for as long as there is at least one active connection.

Co-browse only (with code)

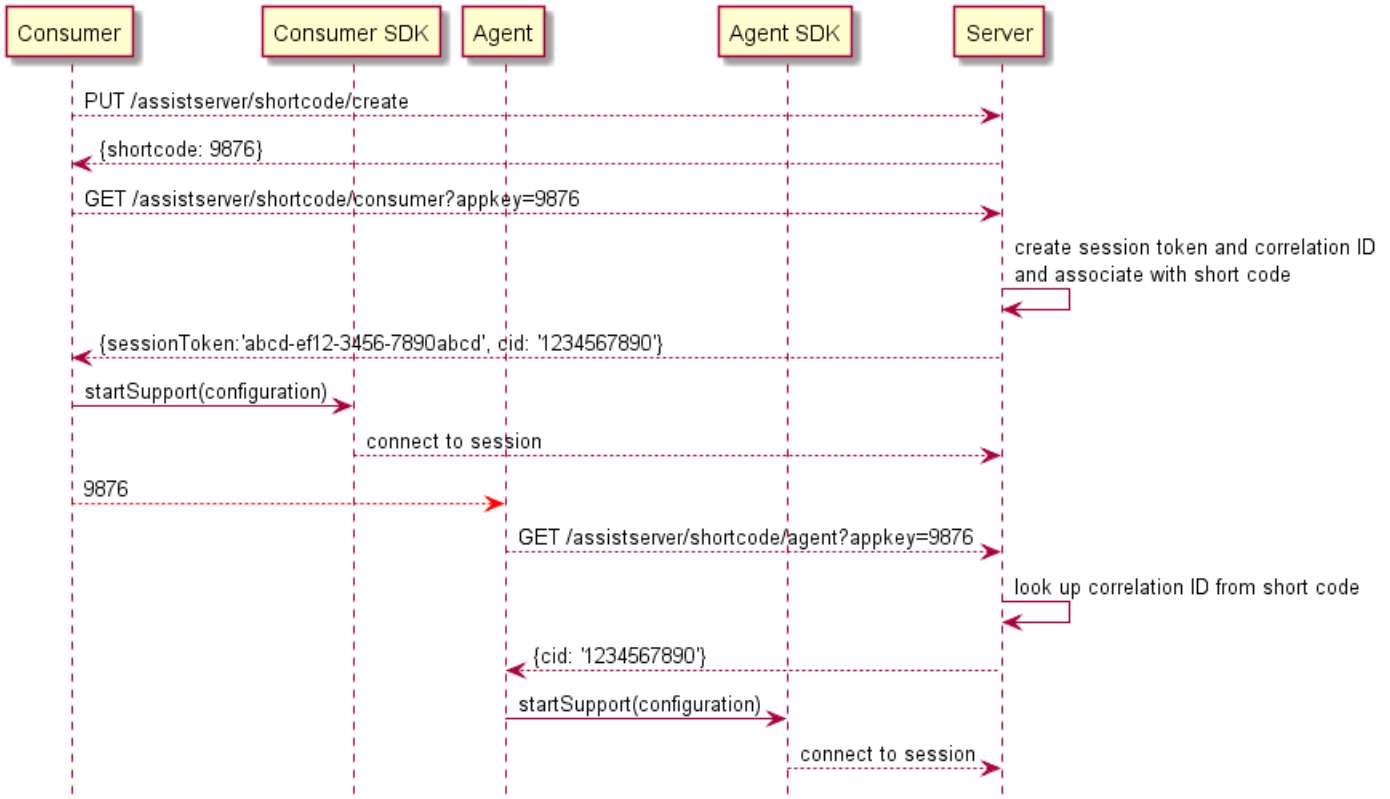
The Expert Assist server exposes short code REST services to help developers communicate the correlation ID in a couple of scenarios. The short code services can create a correlation ID and session token, and associate them with a short code, which can then be communicated to the other party.

The advantage of communicating a short code, rather than communicating the correlation ID directly, is that the short code generated by the server will be guaranteed to be both unique while the correlation ID is being communicated, and short enough to be easily communicated by voice (or whatever other out-of-band communication channel is in use) without error.

Note: Once a short code has been used by both agent and consumer to communicate a correlation ID, it is discarded, and can be used by a different agent and consumer to communicate a different correlation ID; it will also expire after 5 minutes. The short code should therefore be used as soon as possible after being created.

Co-browse only

In this scenario, a communication channel already exists between the two parties who need to set up a co-browse session. The consumer uses the REST service to create the short code, and then uses another REST service to create the session token and the correlation ID, which are associated with the short code; the consumer uses these in the configuration in their call to `startSupport`. The consumer then communicates the short code to the agent using the out-of-band mechanism, who uses it to connect to the same session by retrieving the correlation ID associated with the short code.

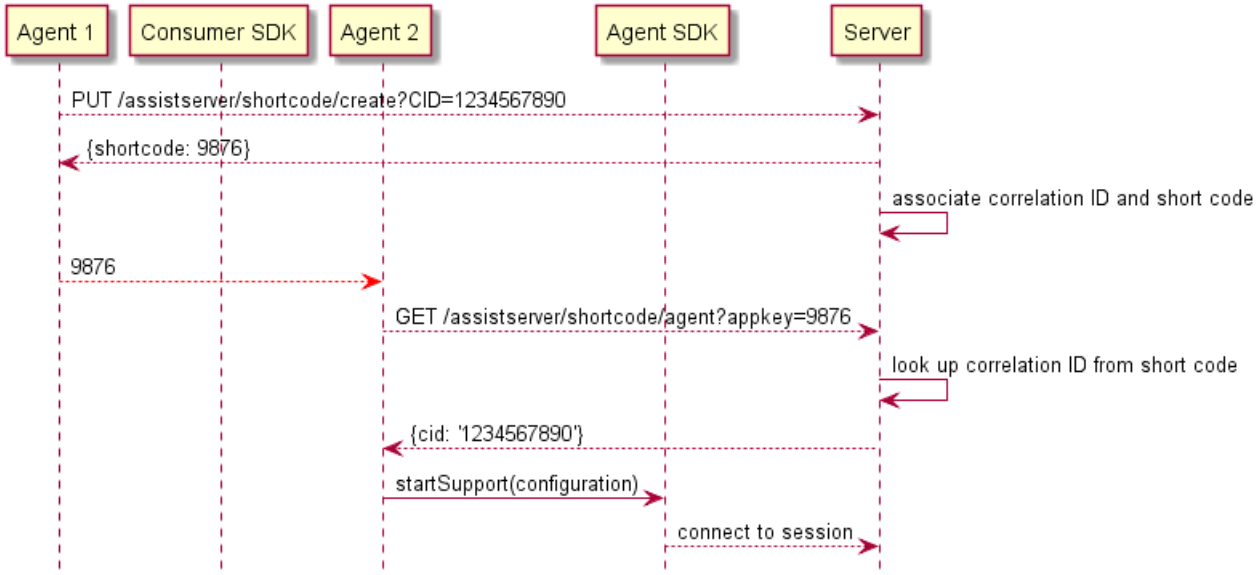


Escalating a call to co-browse

The sequences here are the same as the diagram above, except that the session token is already known to the consumer.

Including a second agent

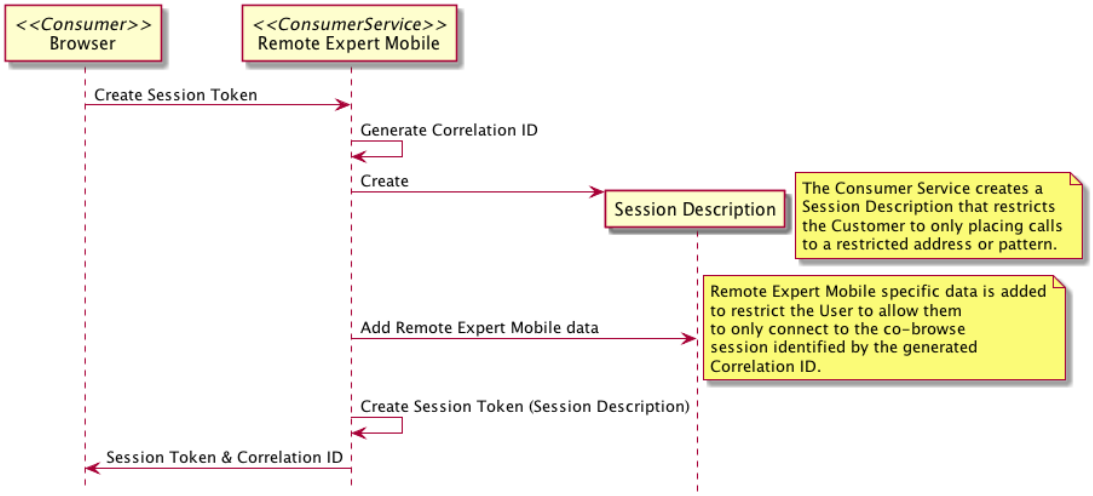
In this scenario, a consumer is in a co-browse session with an agent, and the agent wishes to include another agent into the same co-browse (similar to a consultation call). The first agent already knows the correlation ID in use for the co-browse session, and can use the CID parameter in the initial call to the REST service to associate that correlation ID with a newly created short code:



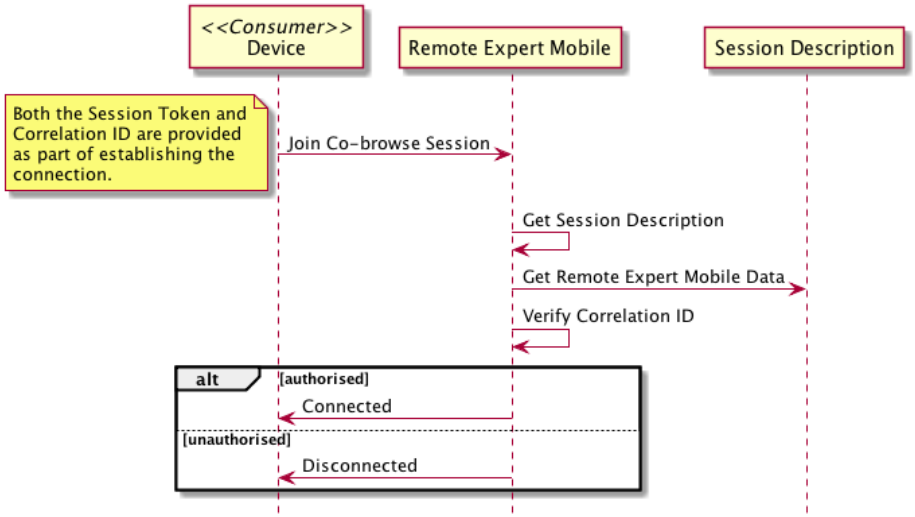
Anonymous Client Access (default)

By default CSDK provides client applications anonymous access to Remote Expert Mobile allowing any user to use voice, video or expert assist from a developer's application. The CSDK generates a Correlation ID that is suitably unique and random, in the form of "assist-" followed by a 25 character alphanumeric string. (e.g. `assist-m2v7r3jpb0jsk5j28ok4b5o4s`)

The diagram below shows the sequence of events:



When the session is established from the Consumer's device to CSDK, CSDK verifies that the specified Session Token is authorized for the Correlation ID, rejecting the connection if not.

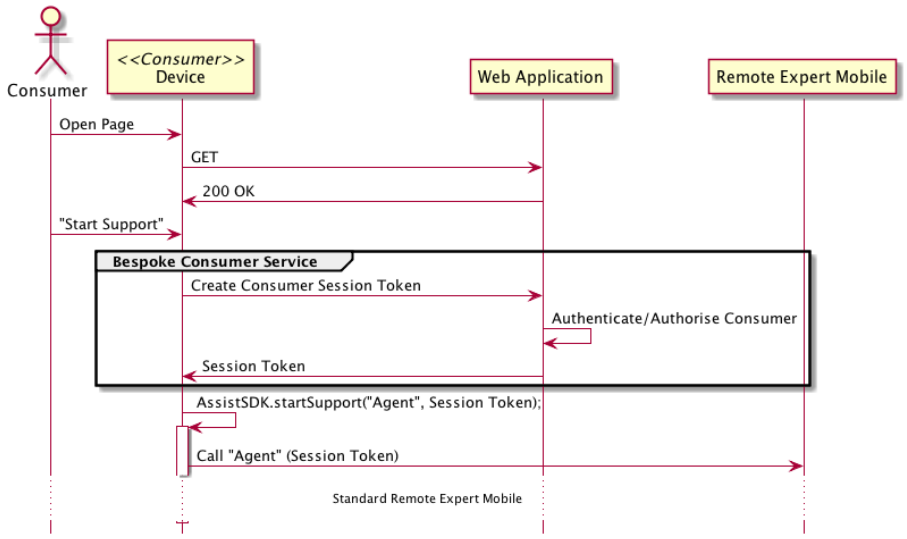

<<Consumer>>
Device
Remote Expert Mobile
Session Description
Both the Session Token and Correlation ID are provided as part of establishing the connection.
Join Co-browse Session
Get Session Description
Get Remote Expert Mobile Data
Verify Correlation ID
alt
[authorised]
Connected
[unauthorised]
Disconnected

Restricted Client Access

It may not be appropriate in all use cases for clients to have anonymous access to experts. Consequently, it is possible to disable anonymous and create a custom implementation. It becomes the responsibility of this implementation to create and manage Session Tokens, which are subsequently provided by the application to the CSDK for it to use when establishing sessions.

For example, Expert Assist may only be available to users that are logged in to the a CSDK for Web application. The server-side web application can validate a Consumer to ensure they're logged in, create a Session Token on the REAS with the appropriate details and provide that token back to the client-side code on the Consumer's device. That token is then passed to the CSDK, which will use it to establish the Expert Assist session.

When the implementation is restricted for consumers, the sequence of events differs slightly from the Anonymous Access, as per the following:



The key difference is that it's the responsibility of the developer's Application to create the Session Token using the REAS; the generated token is provided to CSDK for it to use. Due to the presence of the Session Token, CSDK will not attempt to create a token itself and will only use the one provided.

CSDK Security Highlights for Developers

Security within CSDK is achieved through the following mechanisms:

- Socket Security—HTTPS, Secure Web Sockets (WSS).
- Session Tokens—Session Tokens created with restricted policies.
- Configuration—Behaviour restricted by System Configuration administrated by an administrator.

CSDK by default provides anonymous access to Consumers and Experts and does not implement user authentication or authorization; however, it does take precautions to allow only the appropriate participants to join a session. If client access is restricted, the following items must be considered:

- JavaScript is in Plain Text when running in a User's browser. Consequently, values such as the Correlation ID are readily accessible. As discussed earlier, a suitably unique and random value should be used if specifying one manually.
- REAS Session Tokens can be created that allow varying degrees of restriction. It is recommended that as much restriction as possible be applied.
- The Session Token API exposed by the REAS must be accessed by a Server Side Web Application and that API must not be exposed publicly through a Reverse Proxy.

> **Note**: For more details on security and certificate setup, please refer to the "Installing and Configuring Cisco Remote Expert Mobile" guide.

Callbacks

The consumer web application, iOS, and Android apps can provide certain callback functions to the CSDK. The application executes these functions when certain things happen. They enable the application to respond to events, and in some cases, to tell the CSDK what to do next.

Web

onCobrowseActive and onCobrowseInactive

The application can implement these callback functions to define what happens when co-browsing starts and ends. For example:

```
AssistSDK.onCobrowseActive = function() {
    // Display indicator, log, etc.
}
AssistSDK.onCobrowseInactive = function() {
    // Display indicator, log, etc.
}
```

If an application does not provide these callbacks, then the CSDK provides the following default banner at the top of the window: "This page is currently being shared."

onConnectionEstablished

A consumer application can use this callback to receive a notification when a consumer first joins an Expert Assist session. Once the consumer has joined the Expert Assist session, it is possible for the Agent, or Agent Console, to request permission to co-browse. The request for permission is by default presented to the user by Expert Assist; however, it is possible to override this behavior using the `onScreenshareRequest` callback.

```
AssistSDK.onConnectionEstablished = function() {
    console.log("Connection Established");
};
```

onEndSupport

When an Expert Assist Session is terminated, for example, when the voice and video call ends, or the consumer application calls `AssistSDK.endSupport`, the application can be notified of this event using the `onEndSupport` function.

```
AssistSDK.onEndSupport = function() {
    console.log("Live Assist session ended");
};
```

onInSupport

A consumer can navigate between Expert Assist-enabled pages whilst maintaining the active Expert Assist session across those page navigations. When the target page loads, Expert Assist detects the active session and notifies the application through this callback. This allows the application to present UI, and log events, for example.

```
AssistSDK.onInSupport = function() {
    // Show user extra UI as they are in an Expert Assist session
    showCobrowseUI();
};
```

onScreenshareRequest

Co-browsing by default does not start until the agent requests permission, and the consumer accepts. This process can be overridden from either end of the call using functions exposed by the CSDK, and this callback for the consumer application.

See also the *Cisco Remote Expert Mobile - Advanced CSDK Developers Guide*.

The function assigned to `onScreenshareRequest` returns a Boolean—true indicates that the User grants permission to co-browse, and false indicates that they deny permission to co browse.

```
AssistSDK.onScreenshareRequest = function() {
    // Automatically accept the cobrowse without prompting the user.
return true;
};
```

onWebcamUseAccepted

Using the default calling capabilities of Expert Assist to establish a Voice and Video call, the User is prompted to grant permission for the application or browser to use their webcam. Once the user has granted permission, this callback is triggered. The application could use this notification to cancel or hide a prompt or warning asking the user to accept the webcam permissions to make Voice and Video calling possible.

```
AssistSDK.onWebcamUseAccepted = function() {
    // Hide the warning
    hideWebcamWarning();
};
```

Android

Co-browsing Visual Indicator

The SDK provides a means to customize the visual indication displayed while screen sharing an application. The default implementation displays a notification icon in the notification bar during the application screen share. Place the notification icon in the `drawable` folder, and name it `launcher_icon.png`. A toast (small popup) is also displayed when the sharing begins and ends.

To override this behavior, provide an implementation of the `AssistCobrowseListener` to the `AssistConfigBuilder`.

```
public class CobrowsingListener implements AssistCobrowseListener {
@Override
public void onCobrowseActive(){
    ..implementation code
}
@Override
public void onCobrowseInactive(){
```

```
    ..implementation code
}}
```

- The `onCobrowseActive()` is fired when the application sharing begins with the agent.
- The `onCobrowseInactive()` is fired when the application is no longer being shared with the agent.

onConnectionEstablished

- Not supported in Android

onSupportEnded and onSupportError

When an Expert Assist Session is terminated, for example, when the voice and video call ends, or the consumer application calls `Assist.endSupport()`, the application can be notified of this event if they pass in an AssistListener implementation when they start support via Assist.startsSupport(AssistConfig config, Application application, AssistListener listener). By implementing this listener they will also be notified of any errors that can cause the support session to end.

```
public AssistListenerImpl implements AssistListener {

    @Override
    public void onSupportEnded(boolean endintiatedLocally){
        ...code to run on support ended
    }

    @Override
    public void onSupportError(AssistError errorType, String message) {
        ...code to run on support error
}
```

onInSupport

- Not supported in Android

onCobrowseAuthListener

Co-browsing by default does not start until the agent requests permission, and the consumer accepts. This process can be overridden from either end of the call using functions exposed by the CSDK, and this callback for the consumer application.

See also the *Cisco Remote Expert Mobile - Advanced CSDK Developers Guide*.

To do this supply an `AssistCobrowseAuthListener` implementation to the `AssistConfig` before you start a call. This will allow the application to accept or reject the co-browse request.

```
@Override
Public void onCobrowseAuthListener(AssistCobrowseAuthEvent event) {
event.acceptCobrowse();
// or event.rejectCobrowse();
}
```

onWebcamUseAccepted

- Not supported in Android

Allowing the application to decide whether to accept screen sharing

When an agent requests screen sharing, the default behavior on the consumer side is to pop up a `UIAlertView`, presenting the user with options to accept or reject this screen-share request. It is possible for a consumer-side application to override this behavior, using whatever method they choose.

To do this, have a consumer application class conform to the following protocol:
`ASDKScreenShareRequestedDelegate`

This protocol is available in the header file `ASDKScreenShareRequestedDelegate.h`

For example, in the consumer application:

```
@interface ConsumerApp <ASDKScreenShareRequestedDelegate>

@implementation
….
(void) assistSDKScreenShareRequested:(void (^)(void))allow deny: (void (^) (void))deny
{
    // Application logic – at some point, application calls either
    // allow() or deny() etc.
    allow();  // here, always immediately allow screen-sharing.
}
```

Finally, set the `screenShareRequestedDelegate` attribute in the configuration passed in to `startSupport`.

For example:

```
NSDictionary *config = [NSDictionary dictionary];
config[@”screenShareRequestedDelegate”] = self;
…….
……
[AssistSDK startSupport:server supportParameters:config];
```

Controlling How a Consumer Accepts or Rejects Shared Resources

When the agent wants to share a document or image with the consumer, by default, the consumer SDK prompts the user to see if they would like to view the shared resource. Only if the consumer chooses to allow the resource is it downloaded and displayed on the screen.

The consumer app developer can override this behavior to change the prompt shown to the user, or to provide another method of determining whether to accept or reject the shared resource. To do so, the app developer passes the `pushDelegate` configuration property (in the configuration passed to the startSupport call). The value of this property should be an object conforming to the `ASDKPushAuthorizationDelegate` protocol.

The `ASDKPushAuthorizationDelegate` protocol defines the method `displaySharedDocumentRequested:allow:deny:`, which takes a shared document object and two closures, `allow` and `deny`. The `allow` closure should be executed if the application determines that the shared resource should be shown to the user. The `deny` closure should be executed if the application determines that the shared resource should not be shown to the user.

See the following example code for a class that sets the `pushDelegate` configuration property to itself, and provides the `displaySharedDocumentRequested` method required by the `ASDKPushAuthorizationDelegate` protocol:

```
@interface ViewController () {
    UIAlertView *pushAuthAlertView;
    void (^pushAuthAllow)(void);
    void (^pushAuthDeny)(void);
}

[…]

@implementation ViewController
[…]
- (IBAction) startLiveAssist:(id)sender {
   NSString *server = [CXLAConfiguration getServerHost];

   NSLog(@"Starting LiveAssist with server %@", server);
   NSMutableDictionary *config = [NSMutableDictionary
                                  dictionaryWithDictionary:@{@"destination" :
@"agent1",
@"acceptSelfSignedCerts" : @YES,
@"videoMode" : @"full",
@"correlationId" : @"100001",
@"pushDelegate" : self}];
   [AssistSDK startSupport:server supportParameters:config];
}

- (void)displaySharedDocumentRequested:(id)sharedDocument allow:(void (^)(void))allow
deny:(void (^)(void))deny {
    pushAuthAllow = allow;
    pushAuthDeny = deny;
    pushAuthAlertView = [[UIAlertView alloc] initWithTitle:@"Shared Document"
message:@" The agent wishes to send you a shared document.  Would you like to view it?"
delegate:self cancelButtonTitle:@"Cancel" otherButtonTitles:@"OK", nil];
    [pushAuthAlertView show];
}

- (void)alertView:(UIAlertView *)actionSheet
clickedButtonAtIndex:(NSInteger)buttonIndex {
    if (actionSheet == pushAuthAlertView) {
        if (buttonIndex == 0) {
            NSLog(@"Cancel button clicked.  Will not show shared document.");
            pushAuthDeny();
        } else {
            NSLog(@"OK button clicked.  Showing shared document...");
            pushAuthAllow();
        }
    }
}

[…]

@end
```

In this example, the prompt shown to the user is similar to the one shown by default if no `pushDelegate` is defined. However, the `displaySharedDocumentRequested:allow:deny:` method can be changed to show a different prompt to the user, or to determine in some other way whether the resource should be shown to the user.

Application Delegation

It is possible for an application to be notified of and to take further control of certain actions that occur within *Remote Expert Mobile.* The following sections discuss what notifications the application can receive and what operations are subsequently available.

The *CSDK* supports multiple delegates for any of the following protocols. Delegates registered with the CSDK must conform to at least one of the protocols below. If they do not, then they are *not* added to the delegate set. The delegates

are not ordered and as such the order in which they are messaged is not defined. Multiple delegates can support the same or different protocols. The registered delegates are managed by the application, calling the following methods:

```
+ (BOOL) addDelegate:delegate;
+ (BOOL) removeDelegate:delegate;
```

Each `NSNotification` name is the capitalized form of the method name without the trailing colon, for example `@selector(assistSDKDidDoSomething)` yields a name of `@"AssistSDKDidDoSomething"`. `NSNotification` object and userInfo properties vary as detailed below.

Documents

```
@protocol AssistSDKDocumentDelegate <NSObject>
@optional
- (void) assistSDKDidOpenDocument:(NSNotification*)notification;
- (void) assistSDKUnableToOpenDocument:(NSNotification*)notification;
- (void) assistSDKDidCloseDocument:(NSNotification*)notification;
@end
```

The document delegate is notified whenever a shared document is opened and closed, or if it could not be opened. In each case, the notification's object is the `ASDKSharedDocument` object. If the document cannot be opened, the `userInfo` is populated with an `NSString` `reason` value.

The `ASDKSharedDocument` has a `close` method that allows the document to be programmatically closed.

Annotations

```
@protocol AssistSDKAnnotationDelegate <NSObject>
@optional
- (void) assistSDKWillAddAnnotation:(NSNotification*)notification;
- (void) assistSDKDidAddAnnotation:(NSNotification*)notification;
- (void) assistSDKDidClearAnnotations(NSNotification*)notification;
@end
```

Annotations are collections of SVG path elements; currently only specific attributes are supported. The `AssistSDKAnnotationDelegate` `will` and `did` notifications' object is an `NSMutableDictionary` including the following keys and values. The `userInfo` dictionary is used for ancillary data.

Key	Value Class
kASDKSVGPathKey	UIBezierPath
kASDKSVGLayerKey	CAShapeLayer
kASDKSVGPathStrokeKey	UIColor
kASDKSVGPathStrokeWidthKey	NSNumber
kASDKSVGPathStrokeOpacityKey	NSNumber

The `CAShapeLayer` is created and initialized with the available attributes between the `will` and `did` calls; any values changed in the `will` call are used in the initialization. To affect the display of the annotation in the `did` call, the `CAShapeLayer` must be manipulated directly.

The `clear` notification's object is an array of `CAShapeLayer` instances. The dictionary used to create the layer is no longer available but the layer could contain metadata placed on it in the `did` call.

Errors

```
@protocol AssistSDKDelegate <NSObject>
```

```
- (void) assistSDKDidEncounterError:(NSNotification*)notification;
@end
```

The `object` of this notification is an `NSError`. No keys are defined for the `userInfo` dictionary, but error reporters may add additional details that could be useful for observing delegates.

Each reported `NSError` has its `code` attribute set to one of the constants provided in the supplied `ASDKErrorCodes.h` file.

A class can act as an error listener by conforming to this protocol, implementing the specified method, and by adding the class as a delegate listener prior to the call to `[AssistSDK startSupport…]`

For example:

```
[AssistSDK addDelegate:self];
```

Co-browse indicator

An application can implement the method `(void) cobrowseActiveDidChangeTo:(BOOL)active` and be informed of when an application's screen is shared to an agent, and also when no agents can see the application's screen (for example, when the application is backgrounded, or when an agent leaves a co-browsing session).

There are two ways that an application can use this method:

- by implementing it on the main application delegate class;
- by conforming to the `AssistSDKDelegate` protocol on a class which implements the method, and adding the class as a delegate listener prior to the call to `[AssistSDK startSupport….]:`

For example:

```
[AssistSDK addDelegate:self];
```

Annotations

The `AssistAnnotationListener` offers the following functionality:

- Notification of new annotations received
- Notification of annotations cleared
- Provides an `AnnotationContext` per support session, that can be used to manually clear annotations and turn off the auto-clearing of annotations in the SDK.

The `AssistAnnotationListener` must be set on the `AssistConfigBuilder`.

Notification of New Annotations

Implement the following method on the `AssistAnnotationListener`:

```
@Override
public void newAnnotation(Annotation annotation, String sourceName){
    ..implementation code
}
```

The received `Annotation` object contains the following values:

- `Path getPath()`—this is the standard Android library Path object that is used to draw the new Annotation
- `int getStrokeColour()`
- `int getStrokeWidth()`
- `float getStrokeOpacity()`

The received `sourceName` is the name of the source that created the annotation, for example the agent's name.

Notification of Annotations Cleared

Implement the following method on the `AssistAnnotationListener`:

```
@Override
public void annotationsCleared(){
    ..implementation code
}
```

This method is called every time the annotations are cleared on the screen, regardless of how the clearance is triggered. For example, the method is called if the Agent clears annotations, or the `AnnotationContext` is used by the developer

For further information, see AnnotationContext below.

AnnotationContext

Use the `AnnotationContext` to take control of clearing Annotations manually. A new context is created for every support session. The `AssistAnnotationListener` allows the developer to obtain the context. The listener also lets them know when it is no longer valid, and should be discarded. The `AnnotationContext` throws an `Exception` if any methods are called on it after the context has ended.

```
@Override
public void annotationContextInitialised(AnnotationContext annotationContext){
    ..store AnnotationContext to be able to clear annotations later
}
@Override
public void annotationContextEnded(AnnotationContext annotationContext){
    ..discard the annotationContext at this point
}
```

Use the following method to manually clear Annotations:

```
annotationContext.clearAnnotations();
```

Automatic clearing of Annotations in the SDK can be turned off by calling the following on the `AnnotationContext`:

```
annotationContext.setClearAnnotationsOnScreenChange(false);
```

Automatic clearing can be set at any time during the support session.

Note: Changing **Activity** within the application clears annotations regardless of this setting.

Typically, this is set to `false` to allow the drawing of other views on top of the support screen without the Annotation being cleared—see the following example.

Shared document callbacks

To receive notifications when an agent shares a document with the consumer, an `AssistSharedDocumentAuth` implementation should be set in the `AssistConfig`. This will allow the application to accept or reject a document. If the `AssistSharedDocumentAuth` implementation is not supplied the behavior will default to showing the user an `AlertDialog` prompting them to accept or reject the document.

```
new AssistSharedDocumentAuth() {
@Override
public void handleSharedDocumentAuth(AssistSharedDocumentAuthEvent
docAuthEvent) {
      // code to accept or reject the document using.
      // docAuthEvent.acceptDocument() or
      // docAuthEvent.rejectDocument()
}
};
```

Provide an `AsssitSharedDocumentReceivedListener` implementation to be notified about successful and unsuccessful incoming shared documents. This also handles document closure notifications and provides an API for

programmatically closing currently opened documents. To programmatically close a document, call `close()` on the `AssistSharedDocument`.

```
new AssistSharedDocumentReceivedListener(){

@Override
public void onError(AssistSharedDocument doc) {
         // code to log error
      }

@Override
public void onDocumentReceived(AssistSharedDocument doc) {
   // can hold on to the doc and close later
}

@Override
public void onDocumentClosed(AssistSharedDocumentClosedEvent doc) {
      // document closed
}
};
```

Remote Expert Mobile—CSDK for Web (JavaScript)

This is an overview of the tasks and development required to integrate RE Mobile with a pre-existing web application using the CSDK for Web. Developer's can easily embed voice, video and or Expert Assist sessions in their website or web application with JavaScript.

Getting started

As highlighted previously, CSDK for Web allows a web site user to be connected over their Wi-Fi or Internet connection to an agent or expert and engage in live communications. Once a session is active, a configurable pop-up will keep call and session active across many pages.

While there are several options that the be configured through the method's various parameters, for the most part developer's will require two simple lines of JavaScript code:

#1—including the assist.js JavaScript library to enable the website:

```
<script src="https://<REAS IP>:8443/assistserver/sdk/web/consumer/assist.js"></script>
```

Note: It is recommended that the 'assist.js' script reference be included prior to the last `<body>` tag in the HTML document.

#2—invoking an Expert Assist session in connection with a link or image:

```
<a title="Expert Assist" onclick="AssistSDK.startSupport({destination : 'agent1'})"><img
src="img/support.png"></a>
```

More details for JavaScript CSDK are as follows.

Self-signed Certificates and Internet Explorer

Using Internet Explorer and HTTPS to connect to a REM installation that has a self-signed certificate causes the call to fail—the REM popup window appears, but it is blank (that is, it has no content at all).

To resolve this, use HTTP instead, or add the self-signed certificate to the IE trust store to use HTTPS.

Enabling websites: including assist.js

In addition to being able to invoke support, every page that is to allow support to continue must include the following script from the CSDK. CSDK is made available by default via the REAS component and can be included as such:

```
<script src="<path_to_server>/sdk/web/consumer/assist.js"></script>
```

It is suggested that this line be added to the template for the site if available.

> **Note**: the JavaScript SDK requires cookies, a pop-up and JavaScript to be enabled in the browser, as well as the use of `<!DOCTYPE html>` as the declared '`doctype`' for the page. This consideration should be taken into account when developing with the SDK.

Packaging JavaScript

The JavaScript SDK is available as part of the RE Mobile OVA within the REAS. As such, the necessary JavaScript library to load the SDK can be included on a web page directly from the server.

Alternatively, the contents of the `expert_assist_web_consumer_SDK-n.n.n.zip` package can be included in your web site and used to source the SDK.

Connecting via JavaScript—invoking a support session

Expert Assist is invoked using the `AssistSDK.startSupport()` method, passing in a configuration object. The configuration object supports the following properties:

Property	Default Value	Description
`destination`		Username of agent or agent group if that agent or agent group is local to the web gateway; otherwise full SIP URI of agent or queue
`videoMode`	`"full"`	Sets whether and from which parties video should be shown. Allowed values are "`full`", "`agentOnly`", and "`none`".
`correlationId`		ID of the co-browsing session
`url`		Base URL of the Remote Expert Mobile Applications Server (REAS), including only scheme, hostname, and port number.
`sdkPath`		URL of the base directory of the consumer SDK, if not deducible from src attribute of HTML script tag which loads the "`assist.js`" file.
popupCssUrl		Set URL of CSS stylesheet containing styles for the Expert Assist popup window in the web page
sessionToken		Web gateway session token (if required)
uui		Passes the specified value in the SIP User-to-User Interface header (see Using UUI, below, for details.)

Connecting Voice and Video Sessions

With Voice and video

Invoking a support session in JavaScript takes the form:

```
AssistSDK.startSupport({destination : "agent1"})
```

Where the parameter specified (in this case '`agent1`') is the address of the queue or remote party to call within the REAS for support. For example, to add a clickable link one could use the following:

```
<a title="Expert Assist" onclick="AssistSDK.startSupport({destination : 'agent1'})"><img
src="img/support.png"></a>
```

When a customer clicks this link, they will establish a session with the support agent designated ‘agent1’.

Typically, customer support services provide a queue system which is serviced by any number of support agents or experts. The destination parameter can be used to specify a queue instead of an individual agent, e.g.

```
AssistSDK.startSupport({destination : "customer-support"})
```

With voice and one-way video from expert

To invoke support without video from the consumer side, but with video from the agent side and with audio from both sides, include the configuration parameter “videoMode” and set it to “agentOnly”. For example:

```
AssistSDK.startSupport({destination : "agent1", videoMode : "agentOnly"});
```

With voice only

To invoke support with voice but without video, include the configuration parameter "videoMode" and set it to "none". For example:

```
AssistSDK.startSupport({destination : "agent1", videoMode : "none"});
```

Using UUI

The value specified will be placed in the SIP User-to-User Interface header exactly as it is passed. The application will need to ensure that the encoding is correct.

Code example:

```
// Triggered by clicking a button, link, etc.
// 5465737420555549 = Hex encoded String "Test UUI"  AssistSDK.startSupport({"destina-
tion":"agent1", "uui":"5465737420555549"});
```

Connecting Co-browse only sessions

Co-browse only mode—Expert Assist with no voice or video by using Correlation ID

REM Mobile sessions do not need a voice or video call to be made using CSDK and can support existing infrastructure such as a traditional phone system (PSTN). If the client web application does not want the JavaScript CSDK to place a call using the application, the developer can provide a unique ID that will correlate the consumer to the agent. The “Correlation ID” provides a way to join agent/consumer sessions without prior knowledge of the domain specific way sessions are identified. This allows an application to leverage the features of Expert Assist (co-browsing, document push, annotations, remote control) without voice/video.

Invoking only the Expert Assist functionality in JavaScript, without a voice/video call, takes the form:

```
AssistSDK.startSupport({correlationId : "your_correlation_ID"})
```

Where the parameter specified is the unique ID used to correlate agent and consumer sessions. For example, add a click to support link the following would be appropriate:

```
<a title="Expert Assist" onclick="AssistSDK.startSupport({correlationId :
"your_correlation_ID"})"><img src="img/support.png"></a>
```

Escalating a Call to Include Co-browse

Typically, a call using Live Assist uses voice and video from the start, so implementing a co-browse is easily accomplished.

Note: Escalating a call to include co-browsing is not relevant to co-browse only session, though it in fact uses the same mechanisms; see Co-browse only (with code) or Co-browse only (with correlation ID)

There may be a situation where a customer already has a voice call in progress with an agent, that they need escalate to include co-browsing. To match the customer's screen to the agent's co-browse, the customer can display a short code on their screen—when they tell the agent this code, the agent can connect to co-browse the customer's screen.

The sample applications include code that lets you see this functionality in the UI. You can copy/paste the application code that displays the button as it is from the sample apps, or change it to suit your purposes as required.

The sample presents the following options when a customer clicks the *Help* button:

- Share my screen with support agent
- Call support and then share
- Already on the call, want to share
    - Selecting this option displays a short code on the screen (for example, 962013) that the customer can read out to the agent. The agent then enters this code and connects to co-browse the customer's screen.

The typical scenario for this functionality is as follows:

1. Agent prompts consumer to click the button
2. Consumer reads the generated short numeric code to the agent
3. Agent enters the code into their console
4. Consumer and agent are connected to the same co-browse—agent can now request co-browsing.

With the URL for the REAS

Ordinarily, the URL for the REAS is derived from the URL for the client web page, using the same scheme, host, and port as the client. However, sometimes developers may need to specify a different URL for the application server. In that case, you can do so by including a "`url`" property in the configuration object passed to `startSupport()`. For example:

```
AssistSDK.startSupport({destination : "agent1", url : "https://myserver.test.com:8443"});
```

The URL should only include a scheme, hostname, and port number. Similarly, when the URL configuration parameter is present, documents that are shared with the consumer by the agent are resolved against this URL.

Specifying path to CSDK

Normally, the CSDK will automatically detect its path to the SDK based on the URL used to include the `assist.js` JavaScript. However, in some circumstances it may be necessary to specify the path to the JavaScript CSDK manually. This can be accomplished by providing the "`sdkPath`" parameter, for example:

```
AssistSDK.startSupport({destination : "agent1", sdkPath : "http://myserver.com/sdk/"});
```

Excluding Elements from Co-browsing through CSS

To limit the areas of the page the Agent can see while co-browsing with the consumer, a CSS class can be added to HTML elements to instruct the CSDK to mask those areas. Instead, the Agent will see a black box by default, or a colored or transparent box if specified.

To hide an area from the Agent, add the assist-no-show CSS class to any appropriate HTML elements, e.g.

```
<div id="sensitive-details" class="assist-no-show">content</div>
```

By default excluded elements will be rendered as black boxes consuming the same space on the page as the original element, however, the color of the box rendered to the agent can be configured by altering the color attribute of the special `assist-no-show-agent-console` CSS class in your stylesheet. (The only attribute of the assist-no-show-agent-console class that has any effect is the color attribute.) This only affects the rendering of the boxes on the agent console, and does not affect the display of the elements on the consumer pages. For example, the following CSS code will make elements marked with the assist-no-show class render in orange on the agent console:

```
.assist-no-show-agent-console {
      color: orange;
}
```

To make elements marked with the assist-no-show class not appear at all on the agent console, use the following CSS code:

```
.assist-no-show-agent-console {
	color: transparent;
}
```

Customizing the JavaScript CSDK popup window

The popup window in the browser used to maintain traversal off pages can be customized with regard to colors, fonts, and images. To customize it, create a CSS file defining styles for the body tag and for the #title, #logo, and #status elements. When invoking, you must pass the CSS file URL as the `popupCssUrl` configuration parameter. For example:

```
var config = {destination: "agent1", popupCssUrl: "/assistsample/css/popup.css"};
AssistSDK.startSupport(config);
```

To customize the background of the window, specify background attributes for the body tag:

```
body {
	background-color: #0000FF;
	background-image: url('/assistsample/img/foo.jpg');
}
```

To customize the logo, specify a background image for the #logo element, along with width and height attributes:

```
#logo {
	background-image: url('/assistsample/img/newlogo.png');
	width: 64px;
	height: 64px;
}
```

Fonts can be customized by specifying font attributes for the `#title` and `#status` elements.

Remote Expert Mobile—CSDK for iOS (Objective C)

This is an overview of the tasks and development required to integrate RE Mobile with a pre-existing web application using the CSDK for iOS. Developer's can easily embed voice, video and or Expert Assist sessions in their website or web application with Objective C.

iOS and Xcode Supported Versions

The officially supported versions of Xcode and iOS for REM 11.5(1) are Xcode 6 and iOS 8—to use Xcode 7 and run on iOS 9, rebuild the sample app using the instructions below, and re-submit to the app store.

Existing application binaries built with earlier versions of Xcode should continue to work without modification, although you may be prompted to trust the application/developer.

1. New or existing projects loaded into Xcode 7 requires changes before they build and run:
2. Disable the generation of bitcode – `Enable Bitcode = NO`.

Add entries to your application's `plist` file to disable the new iOS 9 Application Transport Security feature—see the following for further information: https://developer.apple.com/library/prerelease/ios/technotes/App-Transport-Security-Technote/

Note: The iOS sample application has been modified accordingly.

Embedding the CSDK library

In order for CSDK to work on iOS you need to link your code against the libraries that are needed. The libraries are found in the `expert_assist_iOS_SDK-n.n.n.zip` archive.

And it is necessary to add them to the XCode project:

- Add "libAssistSDK.a" in the "Target -> Build Phases -> Link Binary With Libraries" section of your XCode project.

It is also necessary to add the following standard libraries:

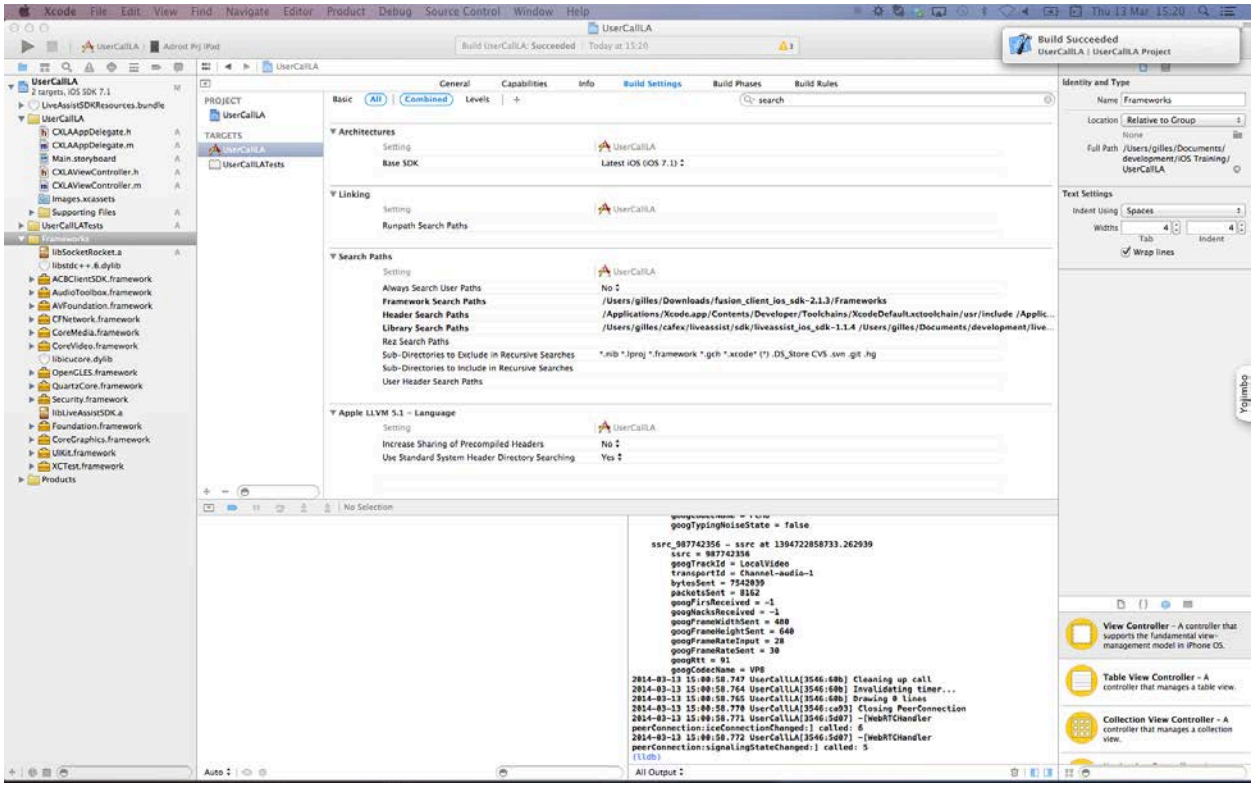
- ImageIO.framework
- MobileCoreServices.framework

This is achieved by selecting "Target -> Build Phases -> Link Binary With Libraries" in your XCode project, and then clicking the "+" button to select and add the required libraries.

Then, in "Build Settings", ensure that:

- "Header Search Paths" includes the location of "AssistSDK.h".

The project will then typically look similar to the following screenshot:



Using CSDK for iOS

In order to use CSDK in an application, import the header file:

```
#import <AssistSDK.h>
```

Invoking a Remote Expert Mobile Session

With Voice and Video

Call and start CSDK by adding the following line:

```
[AssistSDK startSupport: @”myserver.test.com” destination: @"agentXYZ"];
```

Normally this call will be called when the user clicks on some kind of “Help” or “Request support” button.

■ The first argument can be the fully qualified name of the server that hosts the REAS, or it can be a URL for the REAS including a scheme, host, and port, e.g. “https://myserver.test.com:8443”. If it is a URL, it should only include a scheme, hostname, and port number. If the application is behind a reverse proxy, a URL (scheme, hostname, and port) must be used, rather than just the server name.

■ The second argument is the name of the agent or the queue to be contacted. For example, to specify a queue, simply alter the destination parameter:

■ [AssistSDK startSupport: @”myserver.test.com” destination: @"queueXYZ"];

A more complete example is as follow:

```
#import "WelcomeController.h"
#import <AssistSDK.h>

@interface WelcomeController ()
@end

@implementation WelcomeController
...
- (IBAction) startLiveAssist:(id)sender {
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    NSString *address = [defaults stringForKey: @"serverAddress"];
    if ((address == nil) || ([address length] == 0)) {
        address = @" myserver.test.com";
    }

    // the single line to start the CSDK
    [AssistSDK startSupport: address destination:@"agent1"];
}
...
@end
```

So far we have covered the basic usage of the CSDK for iOS. The API allows more advanced scenarios to be covered. These scenarios are detailed in the next sections but they are all called from the same API entry point:

```
+ (AssistSDK*) startSupport: (NSString*) server supportParameters: (NSDictionary*) config
```

This API entry point requires you to provide the server along with the configuration parameters are by means of a dictionary.

To call an agent as previously described but by using this method would be as follows:

```
NSDictionary *laConfig = [NSDictionary dictionaryWithObjectsAndKeys:
                    @"agentXYZ", @"destination", nil];
[AssistSDK startSupport: @"myserver.test.com"
supportParameters:laConfig];
```

Note: This starts a call with voice and video. For co-browse only sessions, see Co-browse Only

The following configuration properties are supported:

Property	Type	Default Value	Description
destination	NSString		Address of agent or queue
videoMode	NSString	@"full"	Sets whether and from which parties video should be shown. Allowed values are @"full", @"agentOnly", and @"none".
correlationId	NSString		The correlation ID.
acceptSelfSignedCerts	NSNumber (@NO or @YES)	@NO	Whether self-signed certificates should be accepted. Set to @YES or @NO. Useful for development and demonstration purposes, however should not be used (or should be set to @NO) in production environments.
useCookies	NSNumber (@NO or @YES)	@NO	Whether cookies set up to be sent to the REAS should be sent on the web socket connection. Set to @YES or @NO.
hidingTags	NSSet		Set of numeric tags to be used for making content invisible on the agent console.
maskingTags	NSSet		Set of numeric tags to be used for masking content, making it appear as black or colored boxes on the agent console.
maskColor	UIColor		Color of boxes to be shown on agent console in place of masked content
timeout	NSNumber		Approximate number of seconds to wait to establish communication with the REAS. May be created from a float, for example: [NSNumber numberWithFloat:30.0];

Property	Type	Default Value	Description
sessionToken	NSString		Web Gateway Session Token (if required)
uui			Passes the specified value in the SIP User-to-User Interface header (see Using UUI, below, for details.

Using UUI

The value specified will be placed in the SIP User-to-User Interface header exactly as it is passed. The application will need to ensure that the encoding is correct.

Code example:

```
NSString *uui = @"5465737420555549" // Hex encoded String "Test UUI"

NSDictionary *laConfig = @{@“destination”:@“agent1”, @“uui”:uui};

[AssistSDK startSupport:server supportParameters:config];
```

Note: Setting the UUI has no effect in co-browse only sessions; see Co-browse only (with code) or Co-browse only (with correlation ID)

Co-browse Only

The correlation ID and session token need to be placed in the support parameters as received from the short code service in response to the call to the shortcode/consumer service (see Co-browse only (with code)). The data object of the response is a JSON object which can be converted into an NSDictionary.

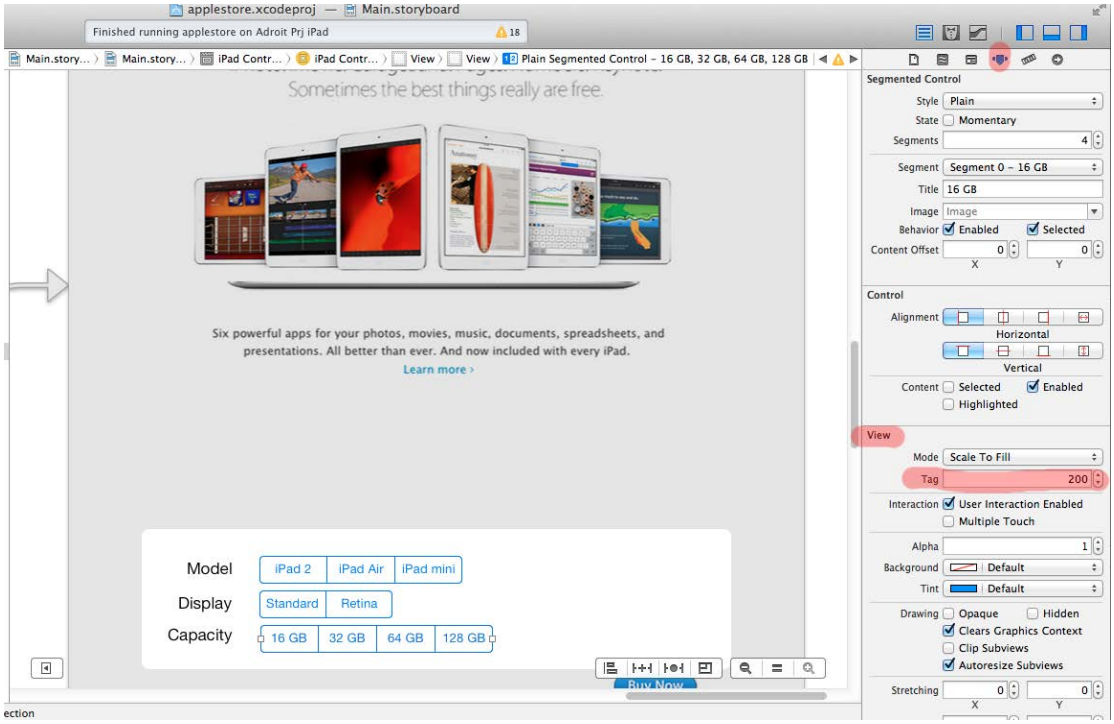
Code example:

```
NSError* jerror = nil;

NSDictionary* dictionary = [NSJSONSerialization JSONObjectWithData:data options:0 error:&jerror];

NSString *cid = dictionary[@”cid”];

NSString* sessiontoken = dictionary[@”session-token”];

NSDictionary *laConfig = @{@“correlationId”:cid, @“sessionToken”:sessiontoken};

[AssistSDK startSupport:server supportParameters:config];
```

Excluding elements from app sharing

One use of the advanced API is to allow iOS UI Elements from being excluded from the co-browse. This ensures the agent can only see the information they are authorized to see.

In the CSDK for iOS, this is done by way of marking the UI element with a specific tag value. This is achieved in XCode by showing the Attribute Inspector and then opening the “view” panel for the UI elements that are to be hidden.



There are then two options.

Option 1—Use of a unique tag value to hide elements

In that scenario, the same tag value is used to mark all the elements that need to be hidden. In that case, the single tag value is submitted as an argument of the “`hidingTags`” dictionary argument:

```
NSSet *tag = [NSSet setWithObjects:
                    [NSNumber numberWithInteger:100], nil];
NSDictionary *laConfig = [NSDictionary dictionaryWithObjectsAndKeys:
                                @"agent1", @"destination",
                                tag, @"hidingTags",
                                nil];
[AssistSDK startSupport: @”myserver.test.com”
supportParameters:laConfig];
```

The exact tag value does not really matter. It just needs to match between what is specified in the UI editor and what is passed as an argument to the “`startSupport: supportParameters:`” method.

Option 2—Use of multiple tag values to hide elements

Some applications already use the `UIView` tag value and they require each UI component to use a unique value. In that case the pre-existing tag values will be used and they will be submitted as argument when calling the "`startSupport`" method.

```
// Here the components to hide have the tag values 100 and 200
NSSet *tags = [NSSet setWithObjects:
                  [NSNumber numberWithInteger:100],
                  [NSNumber numberWithInteger:200],
                  nil];
NSDictionary *laConfig = [NSDictionary dictionaryWithObjectsAndKeys:
                              @"agent1", @"destination",
                              tag, @"hidingTags",
                              nil];
[AssistSDK startSupport: @"myserver.test.com"
supportParameters: laConfig];
```

Masking Elements

In addition to hiding elements, you can also mask elements using the "`maskingTags`" configuration property, in which case the masked elements appear as solid rectangles in the agent console. By default, the rectangles appear black in the agent console, however, this color can be changed using the "`maskColor`" configuration property. Like "`hidingTags`", the value of the "`maskingTags`" property must be an object of type `NSSet`, which contains objects of type `NSNumber` which are constructed from integers. The value of the "`maskColor`" property must be an object of type `UIColor`.

For example, the following code masks elements with tags 150 and 151 using red rectangles:

```
// Here the components to mask have the tag values 150 and 151
NSSet *tags = [NSSet setWithObjects:
                  [NSNumber numberWithInteger:150],
                  [NSNumber numberWithInteger:151],
                  nil];
NSDictionary *laConfig = [NSDictionary dictionaryWithObjectsAndKeys:
                              @"agent1", @"destination",
                              tag, @"maskingTags",
                   [UIColor redColor], @"maskColor",
                              nil];
[AssistSDK startSupport: @"myserver.test.com"
supportParameters: laConfig];
```

Co-browse only mode—Expert Assist with no voice or video by using Correlation ID

Like the CSDK for Web client capabilities, the iOS SDK can also be commanded to start a sharing session without initiating an Audio/Video call.

This is for the scenarios where the PSTN could be used to provide the call, or where a chat session would be used instead of an audio/video session.

So, if the client application does not want to place a call, the application can provide a unique identifier to correlate the consumer and agent side. This "correlation ID" provides a way to join agent/consumer sessions without prior knowledge of the domain specific way sessions are identified. This allows an application to leverage the features of Expert Assist (co-browsing, document push, annotation, remote control) without voice/video.

Invoking the Expert Assist functionality in the iOS client in situations where CSDK should not create a voice/video call, takes the following form:

```
NSDictionary *laConfig = [NSDictionary dictionaryWithObjectsAndKeys:
@"mycorrelationid_xyz", @"correlationId",
nil];
[AssistSDK startSupport: @"myserver.test.com"
supportParameters:laConfig];
```

Accepting Self-Signed Certificates

By default, self-signed security certificates are rejected by the iOS SDK. If you wish to allow self-signed certificates to be accepted, you must set the "`acceptSelfSignedCerts`" configuration parameter to `@YES`.

It is however recommended that you restrict this mode to your DEBUG builds only. So here is an implementation suggestion:

```
NSLog(@"Starting Assist with server %@ and agent %@", server, agent);

NSMutableDictionary *config = [NSMutableDictionary
                dictionaryWithDictionary:@{@"destination" : agent}];

#ifdef DEBUG
   NSLog(@"DEBUG MODE - Allowing self signed certificates");
   config[@"acceptSelfSignedCerts"] = @YES;
#endif

[AssistSDK startSupport:server supportParameters:config];
```

Warning: The inclusion of this code will result in denial of submission to the application store. This is ONLY recommended for development applications or applications that allow the user to accept the risks of using the self-signed cert. This is not recommended for Production Applications.

Enabling Web Socket Cookie Support

By default, CSDK does not include cookies on the Web Socket connection it opens to the REAS. If however it is required for the Web Socket to include all the appropriate cookies for the Web Socket URL then they can be enabled by providing a "`useCookies`" configuration parameter set to `@YES`. e.g.

```
NSDictionary *config = @{@"destination" : @"agent1",
 @"useCookies" : @YES};
[AssistSDK startSupport: @"myserver.test.com" supportParameters:config];
```

CSDK uses the cookies stored within the `NSHTTPCookieStorage` class provided by iOS. Consequently, cookies that should be applied to the Web Socket must be present in the `NSHTTPCookieStorage` singleton before invoking `startSupport`. CSDK uses the `cookiesForURL` method of `NSHTTPCookieStorage` to obtain the collection of applicable cookies for the Web Socket.

Considerations

iOS Alerts / System Dialog Boxes

Due to limitations imposed by iOS, it is not possible for CSDK to replicate any iOS generated dialog boxes such as Alert boxes, the iOS keyboard or menus generated by HTML (e.g. The popup menu generated by the HTML `<select>` element). Consequently, it should be considered whether it is necessary for the Agent to see those elements and possibly consider alternative implementations, such as JavaScript and CSS.

Password Fields

When entering a password into a text field on a mobile device such as iOS or Android, the device will momentarily display the letter that has been entered before masking it. As a user's device screen is being replicated and displayed to an Agent, it is possible for that Agent to see the password as it is being entered. Consequently, it is recommended that fields that can contain sensitive information are masked using the built-in masking capabilities provided by each CSDK, described in earlier chapters.

Remote Expert Mobile—CSDK for Android (Java)

This is an overview of the tasks and development required to integrate RE Mobile with a pre-existing web application using the CSDK for Android. Developers can easily embed voice, video and or Expert Assist sessions in their website or web application with Java.

Embedding the CSDK library

The CSDK for Android contains three component parts that developers will need to integrate into their application in order to use RE Mobile:

Component	Description
`assets`	A folder containing assets needed by the SDK. The contents of this folder should be copied into the matching 'assets' directory of the application build.
`libs`	The artefacts needed to integrate an application with CSDK. The assist-android-sd.x.x.x.jar file and armeabi-v7a folder should be copied into the application's 'libs' build directory.
`res`	A folder containing resources needed by the SDK for the user interface. The contents of this folder should be copied into the matching 'res' directory of the application build.

Enabling the SDK via AndroidManifest.xml

The CSDK requires entries within the application AndroidManifest.xml in order to function.

Under the root `<manifest>` element the following feature entries should be enabled:

```
<uses-feature android:name="android.feature.CAMERA" android:required="true"
android:glEsVersion="0x00020000"/>

<uses-feature android:name="android.hardware.camera.autofocus"/>
```

Under the root `<manifest>` element the following permission entries should be enabled:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
```

Under the `<application>` element the following service entries should be enabled:

```
<service android:name="com.alicecallsbob.assist.sdk.core.AssistService"/>
```

Implementing the SDK with Java

The main activity of the application should usually extend the `AssistApplicationImpl` class in order to use CSDK.

However, should this be inconvenient, it's also possible to extend a different `android.app.Application` object whilst implementing the `AssistApplication` interface. To follow this pattern an `AssistCoreImpl` object needs to be constructed and terminated with the lifecycle of your application.

The following is an illustration of this pattern:

```
public class SampleAssistApplication extends Application implements AssistApplication
{
    private AssistCore assistCore;

    @Override
    public void onCreate()
    {
        super.onCreate();
        assistCore = new AssistCoreImpl(this);
    }

    @Override
    public void onTerminate()
    {
        super.onTerminate();
        assistCore.terminate();
        assistCore = null;
    }

    @Override
    public AssistCore getAssistCore()
    {
        return assistCore;
    }
}
```

Note: The previously mentioned classes can be resolved against the `com.alicecallsbob.assist.sdk.core` package.

Invoking a Remote Expert Mobile Session

With voice and video

RE Mobile is invoked via the `Assist.startSupport` static call whilst providing an `AssistConfig` configuration item, the Application object for the current application and an `AssistListener` object for receiving feedback on the state of the current session, such as errors.

The `AssistConfig` object supplies the configuration for the session to be initialised and be constructed using by using the `AssistConfigBuilder` class. For example, an `AssistConfig` object can be constructed as follows:

```
AssistConfig config = new
AssistConfigBuilder(getApplicationContext()).setAgentName(“agent1”).setServerHost(“
127.0.0.1”).build();
```

The configuration object supports the following properties:

Method	Mandatory	Default Value	Description
`setAgentName`	No	`“agent1”`	Username of agent or agent group if that agent or agent group is local to the web gateway; otherwise full SIP URI of agent or queue
`setConnectionSecurely`	No	`False`	Whether to connect over HTTP or HTTPS to the supplied server host
`setMediaMode`	No	Voice/video to and from the agent	Sets whether and from which parties video should be shown. Values include voice/video in both directions; voice in both directions but video from the agent only and voice only. The `AssistMediaMode` class provides the enum to control this parameter
`setCorrelationId`	No	Generated	ID of the co browsing session
`setServerHost`	Yes		Base URL of the REAS, including only the hostname or IP address
`setServerPort`	No	8080	Port for the http(s) connection to the REAS
`setSessionToken`			Web gateway session token (if required)
`setHostnameVerifier`	No	None	A `HostnameVerifier` object to validate connections made by the SDK to secure URLs (including pushed content such as documents)
`setTrustManager`	No	None	A `TrustMananger` object to validate connections made by the SDK to secure URLs (including pushed content such as documents)
`uui`	No	None	Passes the specified value in the SIP User-to-User Interface header (see Using UUI, below, for details.

Similarly, an ongoing session can be ended via the `Assist.endSupport()` method call.

Using UUI

The value specified will be placed in the SIP User-to-User Interface header exactly as it is passed. The application will need to ensure that the encoding is correct.

Code example:

```
String uui = "5465737420555549"; // Hex encoded String "Test UUI"
AssistConfigBuilder builder = new AssistConfigBuiler();
```

```
builder.setAgentName("agent1");
builder.setUUI(uui); // Set other desired properties on builder
Assist.startSupport(builder.build(), getApplication(), assistListener);
```

Note: This starts a call with voice and video. For co-browse only sessions, see Co=browse only (with code) or Co-browse only (with correlation ID)

Co-browse Only

The correlation ID and session token need to be placed in the support parameters as received from the short code service in response to the call to the shortcode/consumer service (see Co-browse (with code) or Co-browse (with correlation ID)). The response object of the onResponse method of the listener is a JSONObject containing correlation ID and session token.

Code example:

```
String cid = response.getString("cid");
String sessiontoken = response.getString("session-token");
AssistConfigBuilder builder = new AssistConfigBuiler();
builder.setCorrelationId(cid);
builder.setSessionToken(sessiontoken); // Set other desired properties on builder
Assist.startSupport(builder.build(), getApplication(), assistListener);
```

Excluding Co-browse Elements through tags

It is possible to exclude an area of the screen from being shared to the agent by adding a tag to the View object representing the area. For example:

```
view.setTag(Assist.PRIVATE_VIEW_TAG, true);
```

Other references

Cisco DevNet

https://developer.cisco.com/site/devnet/home/index.gsp

Internet Engineering Task Force (IETF®) Working Group

http://tools.ietf.org/wg/rtcweb/

W3C WebRTC Working Group

http://www.w3.org/2011/04/webrtc/

WebRTC Open Project

http://www.webrtc.org

Acronym List

Item	Description
CODEC	“Coder-decoder" encodes a data stream or signal for transmission and decodes it for playback in voice over IP and video conferencing applications.
CSDK	Remote Expert Mobile Client SDKs. Includes three distinct SDKs for iOS, Android and web/JavaScript developers.
G.711	PCMU/A 8-bit audio codec used for base telephony applications
G.729a	Low-bitrate audio codec for VoIP applications
H.264	Video codec. H.264 is the dominant video compression technology, or codec, in industry that was developed by the International Telecommunications Union (as H.264 and MPEG-4 Part 10, Advanced Video Coding, or AVC). Cisco is open-sourcing its H.264 codec (Open H.264) and providing a binary software module that can be downloaded for free from the Internet. Cisco will cover MPEG LA licensing costs for this module.
Opus	Low bit rate, high definition audio codec for VoIP applications. Opus is unmatched for interactive speech and music transmission over the Internet, but is also intended for storage and streaming applications. It is standardized by the Internet Engineering Task Force (IETF) as RFC 6716 which incorporated technology from Skype's SILK codec and Xiph.Org's CELT codec (www.opus-codec.org)
REAS	Remote Expert Mobile Application Server
REMB	Remote Expert Mobile Media Broker
UC	Unified Communications
VP8	Video codec—VP8 is a video compression format owned by Google. Google remains a staunch supporter of VP8 after buying On2 Technologies in 2010; Google then released VP8 software under a BSD-like license, as well as the VP8 bitstream specification under an irrevocable license, and free of royalties. VP8 is roughly equivalent in processor usage, bandwidth, and quality to H.264.
WebRTC	Web Real Time Communications for communications without plug-ins