Avaya Aura® Contact Center
&
Avaya Contact Center Select

Contact Control Service SDK (Java)
Release 7.1.2

Release Notes & User Guide

25 Sep 2021

This document contains information on SDK versioning, compatibility & user guide that is specific to this Avaya Contact Center SDK.

Table of Contents

Purpose

This document contains information on SDK compatibility and change history specific to this SDK. Please refer to the product documentation for more details.

Publication History

Issue	Change Summary	Date
01	Initial release of the CCS SDK	19/12/2016
02	Release of the CCS SDK to align with Contact Center 7.0.2.0	20/11/2017
03	Release of CCS SDK to align with CC 7.1.0.1	23/08/2019
04	Release of CCS SDK to align with CC 7.1.0.2	11/10/2019
05	Routine release of CCS SDK with various bugfixes	9/06/2020
06	Release of CCS SDK to align with CC 7.1.1.0	06/10/2020
07	Release of CCS SDK to align with CC 7.1.2.0	25/09/2021

Change History

SDK Version 7.1.2

The CCS SDK has been released to support Avaya Aura Contact Center and Avaya Contact Center Select release 7.1.2.

A new method is added to User: 'retrieve'. The method can be used by a client to pull a contact. Some detailed error messages could be received from server side if something happened during pull operation.

The release also contains a number of fixes:

- CC-23776 Stuck contact after EndPoint Unregistered event
- CC-23314 The label of Workcard displays incorrectly when Agent process the contact
- CC-24225 Unsubscribing from CCT notifications for logged out session
- CC-24476 CCS SDK doesn't show all custom fields

SDK Version 7.1.1

The CCS SDK has been released to support Avaya Aura Contact Center and Avaya Contact Center Select release 7.1.1.

A new property (and a getter for that property) is added to Interaction: 'canObserveDn' (CC-22397). The property can be used by a client to check if supervisor of an interaction is allowed to observe non-skillset voice calls.

New ContactType is added to support OpenQ contacts.

The release also contains a number of fixes:

- CC-21550 Avaya CRM connector does not display agent/customer details
- CC-21592 CCS API reconnect attempts from multiple clients may overload IIS and disrupt AACC performance
- CC-21880 WebSocket closed due to CloseStatus=MessageTooBig error

JavaFx Ref Client fixes:

- CC-21624 Cannot do conference calls. They always behave as transfers. Issue happens with Java Reference Client 7.1.0.3-1.
- CC-21595 cannot do hold/unhold at refclient 7.1.0.3
- CC-21549 during AES HA failover as CRM Connector getting CTI Error and call card in stuck

SDK Version 7.1.0.3

A defect in client reconnect logic was resolved. Previously, a client could get into an eternal reconnect/disconnect loop if certain conditions were met (specifically, if an authentication-related disconnect was issued from CCS on a non-null client Session: CC-21274).

A 'Language' property and associated *getLanguage()* and *setLanguage()* methods were added to the User object (CC-20810)

Added a new property (and a getter for that property) to Interaction: '*webChatCustomerConnected*'. A customer in a web chat may temporarily lose their connection, for example if their internet connection drops. In this case, an Interaction Update will be sent to the client with *WebChatCustomerConnected* set to false (CC-19537)

Fixed Intrinsics display in java ref client.

Added a getClient() method to SessionI interface (a getClient() method was already in Session.java but had been omitted from the interface).

SDK Version 7.0.2.0-1

The CCS SDK has been released to support Avaya Aura Contact Center and Avaya Contact Center Select release 7.0.2.0.

A new Profile has been added for server applications that require control of multiple Contact Center Users within a ClientI instance.

No changes were made to the existing interfaces.

The JDK version for compilation of the Contact Control Service SDK API and Reference Client has been up issued to line up with the Java version installed on the Contact Center server in the 7.0.2.0 release.

SDK Version 7.1.0.2

The CCS SDK has been released to support Avaya Aura Contact Center and Avaya Contact Center Select release 7.1.0.2

This version of the API offers feature parity with Workspaces on AACC v. 7.1.0.2; with the following exceptions:

- Agent signature (get, set, update)

New features include Suggested responses(email), suggested phrases(web chat), page push urls, customer history, customer history media transcript, ad-hoc email, screenpops (basic implementation only).

Interface changes: none

SDK Version 7.1.0.1

The CCS SDK has been released to support Avaya Aura Contact Center and Avaya Contact Center Select release 7.1.0.1

Support has been added for the email and web chat contact types with new interfaces added. The Java ref client has been updated to have basic Web chat and Email functionality.

Interface changes: Contact Center codes (*ContactCenterCodeI*)have been converted from *Integer* to *String*. This will break backwards-compatibility, but the previous behavior was defective and did not align with AACC CC Code type (which is *String*).

Versions and Compatibility

What is Compatible

CCS SDK Version	AACC/ACCS 7.1.2	AACC/ACCS 7.1.1	AACC/ACCS 7.1.0.3	AACC/ACCS 7.1.0.2	AACC /ACCS 7.1.0.1	AACC /ACCS 7.0.1.0	AACC/ACCS 7.0.0.0
7.0.1.0-1	N	N	N	N	N	Y	N
7.1.0.1	N	N	N	N	Y	N	N
7.1.0.2	Y	Y	Y	Y	N	N	N
7.1.0.3	Y	Y	Y	Y	N	N	N
7.1.1	Y	Y	Y	Y	N	N	N
7.1.2	Y	Y	Y	Y	N	N	N

Getting Started: Contact Control Service SDK

The Java version of the CCS SDK is packaged in the *ccs-sdk-java-package.zip* file and is the starting point for accessing the Contact Control Service SDK. This packaged SDK zip file contains all the contents and resources required to use the CCS SDK.

Supported Java Version

The CCS Reference Client and CCS API have been compiled and tested with the following version of OpenJDK.

1.8.0_242 (recommended)

Compatibility Matrix

The below table is a compatibility matrix detailing the solutions, configurations and features that are supported or not supported through the CCS API.

	AACC AML	AACC SIP	ACCS
Solutions Supported?	N	Y	Y
Features, Configurations & Use Cases that are Supported?			
POM Contact Type	N/A	Y	N
Voice Contact Type	N/A	Y	Y
All Other Contact Types (including Email, EWC, IM, WebComms etc)	N/A	Y	Y
Multiple applications simultaneously controlling the same Agent *	N/A	N	N
Offsite Agent	N/A	N	N

The above table is not an exhaustive list of features supported through the CCS API.

*If a CCS API developed application is controlling an Agent or Supervisor (known as User in the Contact Control Service), it must be the sole application controlling this Agent or Supervisor. No other application can control this Agent or Supervisor in parallel with the CCS API developed application instance. This restriction includes multiple instances of a CCS API developed application.

The applications referred to above include, Avaya Agent Desktop, any Contact Center SDK Reference Client or any application developed against the Contact Center SDK's listed below;

- CCT Open Interfaces SOAP SDK
- CCT Open Interfaces REST SDK
- CCT .NET SDK
- CCMM Web Communications SDK (all)
- CCMM Agent Web Services SDK
- Enterprise Web Chat SDK
- Contact Control Service SDK's

Supported Features

Listed below are the supported features that can be invoked through CCS API 7.1.0.3 based on Contact Type.

Features Supported	Voice	POM	Email	Web Chat
User Features				
Login*[1]	Y	Y	Y	Y
Logout	Y	Y	Y	Y
Ready	Y	Y	Y	Y
Not Ready	Y	Y	Y	Y
Not Ready with Code	Y	Y	Y	Y
After Call Work	Y	N	Y	N
Resource Features				
Call Supervisor	Y	N	N	N
Originate/Initiate *[2]	Y	N	N	N
Interaction Features				
Answer *[3]	Y	N	Y	Y
Hold	Y	Y	Y	Y
Un-Hold	Y	Y	Y	Y
End	Y	Y	Y	Y
Initiate Supervised Transfer	Y	Y	N	N
Initiate Single step transfer	Y	Y	Y	Y
Initiate Supervised Conference	Y	Y	N	N
Complete Transfer	Y	Y	Y	Y
Complete Conference	Y	Y	N	N
Join Conference	Y	N	N	N
Emergency Call	Y	N	N	N
Play DTMF	Y	Y	N	N

Features Supported	Voice	POM	Email	Web Chat
Activity Codes	Y	N	N	N
Intrinsics	Y	N	Y	N
UUI	Y	N	Y	Y
Attached Data	Y	N	Y	Y
Agent Notes	N/A	Y	Y	Y
Get Zones	N/A	Y	N/A	N/A
Wrap-Up	N/A	Y	N/A	N/A
Wrap-Up with Code	N/A	Y	N/A	N/A
Extend Wrap-Up	N/A	Y	N/A	N/A
Create Callback	N/A	Y	N/A	N/A
Preview Dial	N/A	Y	N/A	N/A
Preview Cancel	N/A	Y	N/A	N/A
Redial	N/A	Y	N/A	N/A
Change Conference Owner	N/A	Y	N/A	N
End Conference *[4]	N/A	Y	N/A	N/A
Update Customer Details	N/A	Y	Y	Y
Add To DNC	N/A	Y	N/A	N/A
View Customer Details	N/A	Y	Y	Y
View Interaction details	Y	Y	Y	Y
Supervisor Features				
Force Logout	Y	Y	N	N
Force Not Ready	Y	N	N	N
Force Ready	Y	N	N	N
Observe	Y	N	N	N
Barge-In	Y	N	N	N
Whisper	Y	N	N	N
Web Chat Features				
IsTyping	N/A	N/A	N/A	Y
Suggested Phrases	N/A	N/A	N/A	Y
Push Url	N/A	N/A	N/A	Y
Receive Url	N/A	N/A	N/A	Y
Agent Label	N/A	N/A	N/A	Y
Customer Label	N/A	N/A	N/A	Y
Comfort Messages	N/A	N/A	N/A	Y
Send/Receive Message	N/A	N/A	N/A	Y

Features Supported	Voice	POM	Email	Web Chat
Email transcript to customer	N/A	N/A	N/A	Y
Transcript of transferred chat	N/A	N/A	N/A	N
Email Features				
Read Incoming Email (HTML & Plain text)	N/A	N/A	Y	N/A
View Email details	N/A	N/A	Y	N/A
Suggested responses	N/A	N/A	Y	N/A
Reply	N/A	N/A	Y	N/A
Reply all	N/A	N/A	Y	N/A
CC/BCC	N/A	N/A	Y	N/A
Attach Files *[5]	N/A	N/A	Y	N/A
Insert / edit Signature	N/A	N/A	N	N/A
Originate Email	N/A	N/A	Y	N/A

*[1] The SSO (Signle Sign On) login mode is not supported by the SDK.
*[2] The Originate feature is not applicable for POM. The Preview Dial or Redial features would be the closest equivalent POM feature. Initiate
*[3] The Answer feature is not applicable for POM. POM interactions are automatically answered.
*[4] End Conference is a POM specific feature. In Voice, a User can drop themselves out of a conference by invoking the End feature.
*[5] Attachments are not sent through CCS API. Rather, on an email Reply, an upload URL is sent to the client, which the client can use to upload attachments using whatever mechanism it sees fit.

Getting Started: Contact Control Service SDK API

Information about the CCS API is available in the Java documentation overview. The javadocs provide the following;

1) A description of the Contact Control Service (CCS) & CCS SDK
2) A description of the Contact Control Service API
3) A list of third party dependencies
4) The structure of the CCS API
5) How to get started developing a Contact Control Service API compliant application

The javadocs for the CCS API are located in *documentation\api-javadocs* directory of the SDK package. To launch the javadocs, browse to the *documentation\api-javadocs* directory and open the *index.html* file. This will launch a browser and load the overview page of the CCS SDK.

The third party libraries that are distributed in the SDK are the versions of these libraries that were tested and verified with the CCS API and CCS API Reference Client. Use of other versions of these libraries at runtime may cause unexpected behavior and are not supported.

Getting Started: Contact Control Service SDK Reference Client

Within the SDK package, you will also find a reference client implementation.

The goal of the Reference Client (RefClient) is to provide a sample implementation of how the Java CCS API could be used. The RefClient is an example of how to develop an application against the CCS API but it should not be taken as the only way to use the CCS API.

The GUI of the RefClient has been implemented with the primary goal of understanding how to use the CCS API. There are many buttons which are disabled most of the time. This was done to make it easier to see when a capability becomes enabled as the button, wherever possible, are driven solely from capabilities provided in CCS API events.

For a complete understanding of every operation possible through the RefClient please refer to the Java CCS API java documentation referred to in the section Getting Started: Contact Control Service SDK API.

Creating a Project for the Reference Client in Eclipse IDE

The reference client requires the use of the Java FX library for the user interface implementation. The Java FX library is an extension to Java SE.

In Eclipse to add the Java FX libraries, install the e(fx)clipse IDE plugin.

To create an Eclipse project for the reference client;

1) In Eclipse, click File -> New -> Other -> JavaFX Project
2) Give the project a name, *CcsSdkReferenceClient* for example
3) Un-tick 'Use default location'
4) Browse to the '*reference-client'* directory *o*f the SDK package
5) Ensure that the appropriate JRE version is selected, please see the Supported Java Version section of this guide for guidance
6) Click finish

Note: The class path of the project should already be setup and there should be no errors in the project. If there are errors due to missing libraries, ensure that all jars in the *reference-client\lib* directory of the SDK package are added as libraries to the Java Build Path of the project.

The javadocs for the CCS API can be attached to the ccs-api.jar file in the project. To do this;

1) Right click on the *CcsSdkReferenceClient* project, click Properties
2) Select Java Build Path. Select the libraries tab. Click the drop down for the *ccs-api.jar*
3) Select 'Javadoc Location' and click edit
4) Browse to the '*documentation\api-javadocs*' directory of the SDK package, click validate, click ok

This is useful particularly when developing an application against the CCS API or if source code modifications are being made to the CCS SDK Reference Client for test.

Note: other IDE's can be used for the reference client if preferred.

Building the Reference Client Application

To build the reference client, Ant is utilized. Having Ant installed and setup correctly on your development machine is a prerequisite to build the reference client. Ensure that Ant is on your development machines path environment variable.

The version of Ant that was used to build both the CCS API (located in the SDK package at *api**ccs-api.jar*) and the sample distribution of the CCS Reference Client (located in the *reference-client**sample-distribution* directory of the SDK package) was;

- 1.9.4

The Ant build.xml file for the reference client is located in the *reference-client* directory of the SDK package.

There are two main ant targets in the build.xml. The makejar target will simply build the *JavaFXReferenceClient.jar* file and output the jar to the *reference-client* directory. The generate-distribution target builds a standalone distribution for the reference client which is outputted to the *reference-client**distribution* directory. To run the reference client from different location, this entire distribution directory can be copied to another location on the file system.

There are two ways to build the reference client;

Using Ant from command line

To build the reference client from command line;

1) Through a command prompt shell, browse to the *reference-client* directory of the SDK package.
2) Execute the command *ant generate-distribution*
3) The built reference client distribution is outputted to the *reference-client**distribution* directory

Using Ant through Eclipse

To build the reference client through Eclipse using Ant;

1) Open the Ant view within Eclipse
2) Drag the build.xml file into the Ant view
3) Double click on the generate-distribution ant target
4) The built reference client is outputted to the *reference-client**distribution* directory

Launching the Reference Client

A prebuilt sample distribution is included in the SDK package. To launch this sample;

1) Browse to the *reference-client**sample-distribution* directory of the SDK package
2) Run the LaunchClient.bat file

If you have previously built your own version of the reference client following the steps in Building the Reference Client Application, you can launch this built version of the reference client by;

1) Browsing to the *reference-client**distribution* directory of the SDK package
2) Running the LaunchClient.bat file

The client launcher will run the JavaFXRefClient.jar that is located in the directory that the application was launched from and the reference client application will launch. A *CcsSdk.log* file will be outputted for the reference client in the directory that the application was launched from (sample-distribution or distribution). This log file will contain both the CCS API and the CCS Reference Client outputted logging.

The LaunchClient.bat file specifies the classpath requirements for the Reference Client and CCS API developed applications. The .bat file is an example of a Windows launcher for the Reference Client.

Important Note: Multiple reference client instances should not be launched simultaneously from the same directory. To launch multiple reference client instances simultaneously, please make a copy of the sample-distribution or distribution directory for each reference client instance.

Reference Client User Guide

Detailed below is a guide providing instructions on how to use the Reference Client.

Prerequisites:

1. A fully functional AACC/ACCS 7.0.1.0 or higher server (7.1.0.1 for Email/Web Chat).
2. Have successfully launched the RefClient as described in the section Launching the Reference Client

Signing in to the Reference Client

Below is a step by step guide on how to sign in to the Contact Center from the Reference Client.

1. In the bottom left hand corner of the RefClient the following message should be observed "DISCONNECTED",
    a. If this is not the case select *Session* -> *Disconnect*
    b. Once "DISCONNECTED" is observed proceed to step below
2. Select *Session*, in the top left hand corner of the screen.
    a. A drop down menu will be presented with the following options
        i. *SignIn* – this should be enabled
        ii. *Disconnect* – this should be disabled
        iii. *Exit* – this should be enabled
3. From the *Session* drop down menu select *SignIn*
4. A *SignIn* dialog should be presented, enter the required details:
    a. *Server* – This is the fully qualified domain name of the AACC/ACCS server
    b. *Username* – The username of Windows User Account that is associated with the User (Agent/Supervisor) that you want to connect to the Contact Center. This is the same username which would be used for other Contact Center client applications, i.e. Avaya Agent Desktop. The format required is Domain\Username
    c. *Password* – The password of the Windows User Account that is associated with the User (Agent/Supervisor)
    d. *AutoReconnect* – Checkbox which defaults to selected.
    e. A full description on these items can be found in the Java documentation for the CCS API in the *ClientI* interface.
5. Click the *SignIn* button
    a. The bottom left hand status should transition to "AUTHENTICATED", depending on timing you may also observe the intermediary states "CONNECTING" and "CONNECTED". These are not important as long as the final state is "AUTHENTICATED"
    b. If the state does not transition to "AUTHENTICATED" please refer to the Java documentation in combination with the RefClient logs to determine any possible errors. The most likely causes of connection issues are DNS resolution of the AACC/ACCS server or domain credentials issues so these should be checked first.

6. After successful authentication the RefClient is now connected to Contact Center but the User is not logged into the Contact Center. Observe the current state of the User is "LoggedOut" and the *Login* button is enabled.
    a. If the state is different in your client this may be because the User was already logged in, possibly using a different client.
    b. If required it should be possible to *Logout* the user to return to a "LoggedOut" state
7. It is not necessary for the User to login to Contact Center to receive a direct call (personal call).

Handling the first Interaction

Below is a step by step guide to handle a Voice Interaction in the Reference Client.

Prerequisites:

1. Have successfully signed in to the RefClient as described in the section Signing in to the Reference Client

Steps:

1. From a phone set or another client application place a call directly to this User's number.
2. A new Interaction should appear in the table in the middle of the RefClient
    a. Observe that the Interaction is in the "Ringing" state
    b. Observe that the *Answer* button is now enabled
    c. Intrinsic's for the Interaction are displayed in the left hand pane of the RefClient
3. Click the *Answer* button
    a. Observe that the Interaction transitions to the "Active" state
    b. Observe that several Interaction buttons now become enabled
4. Click the *End* button to release the Interaction

Viewing the Details Forms

To the right of the *Session* menu you can find another menu option called *View*. This menu is used to open the *Supervisor Window* and also open several details forms. The details forms are used to show the current properties and capabilities of CCS API objects.

Prerequisites:

1. Send a direct call to this User as described in the section Handling the first Interaction, but do not answer the Interaction yet.

Steps:

1. Select *View -> Interaction Details*
2. A new *Interaction Details* form will be displayed
    a. The left hand side of the form contains the *Interaction* properties and their associated values, note some of these are blank.

   b. The right hand side of the form displays all the *Interaction* capabilities and their current state. If the capability name is Green then the capability is currently enabled. If the capability name is Red then the capability is currently disabled.
3. Note in this scenario the *canAnswer* capability is enabled. In the main form window the *Answer* button is driven from this capability and hence is enabled.
4. Click the *Answer* button
   a. In the main window note the Interaction buttons which are enabled change
   b. In the *Interaction Details* form note the changes in capabilities and properties
5. Click the *End* button
   a. In the *Interaction Details* form note the properties have all been blanked and the capabilities have all been disabled. When there is no Interaction to display the form displays a blank template.

The other details forms work in a similar way. For more details on the meaning and possible usage of the properties and capabilities of the CCS API object please refer to the relevant Java documentation.

Reference Client Object Model

The Reference Client is a Java FX application with the entry point to the application being the JavaFXRefClient class. This class is responsible for construction and handling all of the JavaFX GUI components, I will refer to it as the main form to distinguish it from the other forms in use. This class is also responsible for the thread context in which most of the GUI components will run. The default name of the main UI thread has been changed to JavaFX. This name is visible in java debuggers as well as the SDK log file.

The main form is composed of several areas, some of which are subdivided;

- Top
  - Contains the menu bar
- Left
  - Contains a side bar with Interaction related content
- Center
  - The main content area, which is further subdivided:
    - User Controls
    - Resource Controls
    - Interaction Table
    - Interaction Controls
    - POM Controls
- Bottom
  - Contains a status bar and clock

The GUI components are decoupled from the CCS API to avoid thread contention and maintain a responsive interface. This has been implemented by not holding references to CCS API objects, the only exception being the ClientI object through which all other CCS API objects can be retrieved.

Incoming events are decoupled in the following way. When an event is received the data contained within it is copied into a cached data object which is then passed to the GUI components to update their display state. This has been done to ensure GUI components can be updated in a consistent manner. This may not be the case if the CCS API objects were being accessed directly as the same CCS API objects will be updated by CCS API threads as incoming events are received.

Outgoing requests into the CCS API are also decoupled from the GUI components. This has been implemented through the use of dedicated request objects for each CCS API request type.

The Executor class defines the incoming event and outgoing request executors.

- The incoming event executor ensures minimal processing is performed within the CCS API threads ensuring reliable delivery of events. There is only one executor to ensure correct ordering of messages to the RefClient.

- The outgoing request executor ensures processing is taken off the JavaFX GUI thread. This in turn ensures the GUI remains responsive.

Requests are separated into their own package: *com.avaya.ccs.javafxrefclient.request*.

The Request class provides the basis for all the requests. Every request must implement the call() method, which is the code which will run on the outgoing request executor. There are several other JavaFX methods which can optionally be implemented;

- setFailed() – executed if there is an exception thrown from within call()
- setSucceeded() – called by default if setFailed() or setCancelled are not called
- setCancelled() – called if the task is cancelled for any reason

Each of these methods operates in the JavaFX GUI thread and hence can directly update GUI components.

For most of the requests in the RefClient it can be seen that these methods are not implemented. This is due to the fact that most requests into the CCS API do not return a success or failure. Instead a separate incoming event is received.

The state of the Reference Client is driven by incoming events and not from outgoing requests; it is never assumed an outgoing request will succeed. The Reference Client instead relies solely on the incoming event to indicate a state change and update the display.

Below is an example illustrating the state caching and request operation for the interaction answer scenario outlined in Handling the first Interaction.

Log fragments below are provided for illustration only, actual logs may differ slightly. The log format includes a thread name within [], examples include;

- ccs-client-1 – CCS API thread
- JavaFX – Java FX UI thread
- incomingEvent – Incoming event executor, within which all events coming from the CCS API are handled
- outgoingRequest – Outgoing request executor, within which all requests to the CCS API are handled

1. From a phone set or another client application place a call directly to this User's number.
2. NEW Interaction message is received by CCS API.
    a. 2016-11-28 13:38:54,706 [ccs-client-1] TRACE c.a.c.c.ProtocolHandler - receivedMessage()
    NEW{"ObjectType":"Interaction"
    2016-11-28 13:38:54,709 [ccs-client-1] INFO c.a.c.c.Session - onInteractionNotification()
3. Message is received by RefClient incoming event task executor.

a. 2016-11-28 13:38:54,711 [incomingEvent] INFO c.a.c.j.InteractionEventTask - call(): type:NEW
   As part of the task execution a new InteractionData object is created. This acts as a cache for the InteractionI CCS API object.
b. In response to the incoming event a success handler triggers on the JavaFX UI thread
   2016-11-28 13:38:54,713 [JavaFX] DEBUG c.a.c.j.InteractionEventTask - SuccessHandler():
   This allows the UI thread to update the UI based on the new cached InteractionData object state.

4. The answer button is pressed.
   a. 2016-11-28 13:39:05,151 [JavaFX] INFO c.a.c.j.InteractionDataGUI - handleAnswerButton()
   b. 2016-11-28 13:39:05,151 [JavaFX] DEBUG c.a.c.j.InteractionExecutor - Answer()
   c. Answer request is created and executed on the outgoing request executor
      2016-11-28 13:39:05,151 [outgoingRequest] TRACE c.a.c.j.r.AnswerRequest - call()
      The request is processed by the CCS API and sent to the AACC/ACCS server.
      Note the RefClient UI is not updated due to the outgoing request.
5. Update message is received in CCS API
   a. 2016-11-28 13:39:05,277 [ccs-client-1] TRACE c.a.c.c.ProtocolHandler - receivedMessage()
      UPDATE{"ObjectType":"Interaction"
6. Update is processed by RefClient incoming event executor
   a. 2016-11-28 13:39:05,278 [incomingEvent] INFO c.a.c.j.InteractionEventTask - call(): type:UPDATE
   b. A new InteractionData object is created to cache the new interaction state. This InteractionData object is then used to update the previously cached InteractionData object.
7. The UI is updated based on the new InteractionData object contents
   2016-11-28 13:39:05,280 [JavaFX] TRACE c.a.c.j.InteractionDataGUI - update()