CEC
2.5.0.0.0

Agent Integration--CC-Gateway Development Guide

Issue	01
Date	2024-03-01

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: https://www.huawei.com

Email: support@huawei.com

Security Declaration

Vulnerability

Huawei's regulations on product vulnerability management are subject to the *Vul. Response Process.* For details about this process, visit the following web page:
https://www.huawei.com/en/psirt/vul-response-process
For vulnerability information, enterprise customers can visit the following web page:
https://securitybulletin.huawei.com/enterprise/en/security-advisory

Contents

1 Overview 1
1.1 Introduction 1
1.2 Network and Development Mode 1
1.3 Connection Mode 2
1.3.1 Modes of Obtaining Agent Events 2
1.3.2 C4 Agent Operation Interface Authentication 3
1.3.3 Detection of Agent Connection Timeout 4
1.3.4 Agent Event Push Mode 4
2 Developing Applications for the First Time 5
2.1 Instance Introduction 5
2.2 Prerequisites 8
2.3 Implementation Process 8
2.4 Implementation Example 11
2.5 Response Message 21
3 Basic App Development 23
3.1 Viewing Interface Invoking Logs 23
3.2 Basic Functions and Configurations 23
3.2.1 Signing In 23
3.2.2 (Optional) Querying Skills Configured for an Agent 24
3.2.3 Signing In to a Skill Queue 24
3.2.4 Obtaining Agent Events 24
3.2.5 Sign Out an Agent 24
3.3 Basic Voice Calls 24
3.3.1 Making an Outgoing Call 24
3.3.2 Answering a Call 26
3.3.2.1 Specifying Whether to Enable Automatic Answering 26
3.3.2.2 Specifying Whether to Enable Manual Answering 27
3.3.2.3 Querying Call Information Before Answering a Call 28
3.4 Functions Used After Call Setup 28
3.4.1 Releasing a Call 28
3.4.2 Holding or Unholding a Call 29
3.4.2.1 Holding a Call 29

3.4.2.2 Unholding a Call....30
3.4.3 Transferring a Call....30
3.4.4 Asking for Internal Help....32
3.4.5 Three-Party Call....33
3.4.6 Muting....34
3.4.6.1 Starting Muting....34
3.4.6.2 Stopping Muting....35
3.5 Real-time Quality Check....35
3.5.1 Inserting....35
3.5.2 Listening On....36
3.5.3 Intercepting....37
3.6 Voice Playback....37
3.6.1 Playing Voice in Non-calling State....37
3.6.2 Playing Voice in Calling State....39
3.7 Basic Multimedia Calls....40
3.7.1 Message Sending and Receiving....41

1 Overview

1.1 Introduction

1.2 Network and Development Mode

1.3 Connection Mode

1.1 Introduction

The CC-Gateway provides REST-style web service interfaces for secondary development engineers to develop agent service functions (REST stands for Representational State Transfer). This document describes the web services provided based on the Representational State Transfer (REST) style. The interfaces provided in this document enable you to develop service capabilities such as audio and video call processing and text call processing for agents. In addition, you can apply for other development packages to integrate various call center businesses.

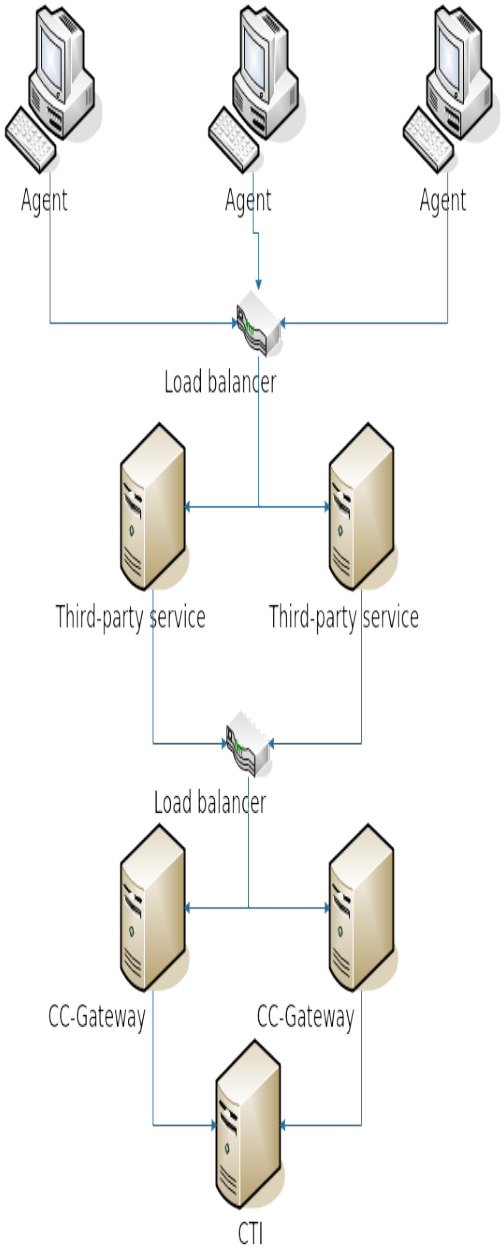
The CC-Gateway interface provides the following functions:

- Enables you to process calls and text chats more easily, which lowers your R&D cost.
- Allows you to use software on an agent to directly access interfaces.

1.2 Network and Development Mode

The CC-Gateway uses the server-to-server connection mode, as shown in **Figure 1-1**.

Figure 1-1 CC-Gateway network



1.3 Connection Mode

1.3.1 Modes of Obtaining Agent Events

Agent events can be obtained in polling mode or push mode.

The two modes are distinguished based on the **callBackUri** and **serviceToken** parameters in the agent sign-in interface.

If both parameters are transferred during agent sign-in, the push mode is used.

If only one parameter or neither of the parameters is transferred, the polling mode is used.

For details, see **Signing In**.

For details about the event polling mode, see **Obtaining Events of One Agent in Polling Mode**.

For details about the event push mode, see **Maintaining the Heartbeat by an Agent** and **Calling Back Pushed Events**.

1.3.2 C4 Agent Operation Interface Authentication

The interface authentication mode can be static or dynamic authentication. The dynamic authentication mode is used by default.

NOTICE

- The static authentication mode is used only at historical sites that are compatible with earlier versions. The new sites need to use the dynamic authentication mode.
- When the static authentication mode is used, GUIDs may be stolen by guessing and used for spoofing, which is insecure. Exercise caution when using this mode. At historical sites, upgrade the third-party system as soon as possible and use the dynamic authentication mode.

Static Authentication

- Modify the configuration in **agentgateway/WEB-INF/config/basic.properties**:
  ```
  AUTHMODE = 1
  ```
- After invoking the sign-in interface to sign in successfully, the agent can obtain the **Set-GUID** header from the response header of the request, for example, **JSESSIONID=27************f5.AgentGateway0**. In the header, **27************f5.AgentGateway0** is the GUID, which is the authentication information of the agent. When invoking another interface, the agent needs to set the **guid** header in the HTTP request header to the obtained GUID. The CC-Gateway will obtain the GUID from the HTTP request header to compare with the value of the **Set-GUID** header delivered previously. If they are the same, the agent is authenticated. If they are different, the agent fails to be authenticated, and code 100-006 or 000-003 is returned.

Dynamic Authentication

Dynamic authentication is an enhancement of static authentication. The GUID updates periodically.

- Modify the configuration in **agentgateway/WEB-INF/config/basic.properties**:
  ```
  AUTHMODE = 2
  ```


- When the GUID is changed, obtain the **Set-GUID** header from the response header of the HTTP request of the event obtaining interface, for example, **JSESSIONID=27***************f5.AgentGateway0**. In the header, **27***************f5.AgentGateway0** is the GUID, which is the authentication information of the agent.
- The **basic.properties** file for dynamic authentication has the following two configuration items, which are not included in the **basic.properties** file for static authentication:

  ```
  GUIDINTERVAL = 60000 //GUID update interval
  GUIDTIMEOUT = 120000 //GUID timeout interval
  ```

  The units of both configuration items are millisecond. The value of **GUIDINTERVAL** must be less than the value of **GUIDTIMEOUT**.

1.3.3 Detection of Agent Connection Timeout

Modify the configuration in the **agentgateway/WEB-INF/config/basic.properties** file as follows:

```
TIMEOUT_FLAG = ON  //Whether to enable detection of agent connection timeout
MAXTIME = 120000 //Timeout interval when the event polling mode is used
EVENT_PUSH_HANDSHAKE_MAXTIME = 120000 //Timeout interval when the event push mode is used
```

If an agent does not send a request to the server within the timeout interval, the agent is forcibly signed out.

1.3.4 Agent Event Push Mode

When the callback address is pushed in HTTPS mode, certificate authentication is used by default.

Configure the trust certificate directory in the **agentgateway/WEB-INF/config/basic.properties** file as follows:

```
eventpush.ssl.trustAll=false  //Whether to trust all certificates. The default value is false, indicating that certificate authentication is used.
eventpush.ssl.cert.file=  //Trust certificate directory in event push mode. It can only be a relative path of the user home directory.
eventpush.ssl.cert.type= //Certificate type.
```

NOTE

If **eventpush.ssl.trustAll** is set to **true**, all certificates are trusted, which may cause security risks. Exercise caution when using this value. The default value **false** is recommended.

Push callback URLs are controlled by a trustlist. Configure them in the **agentgateway/WEB-INF/config/basic.properties** file as follows:

```
EVENT_PUST_WHILTELIST_CALLBACKURI =  //Callback URLs. Use semicolons (;) to separate multiple URLs.
```

Configure the push failure interval and total failure time in the **agentgateway/WEB-INF/config/basic.properties** file as follows:

```
EVENT_PUSH_FAIL_INTERVALTIME = 10000   //Interval at which an event fails to be pushed.
EVENT_PUSH_FAIL_TOTALTIME = 120000   //Total time of event push failures.
```

If the event fails to be pushed, the system pushes the event again when the interval expires. If the event fails to be pushed all the time, the system accumulates the failure duration and forces the agent to sign out when the failure duration exceeds the total time. If the push is successful, the failure duration is cleared and accumulated again upon the next failure.

2 Developing Applications for the First Time

2.1 Instance Introduction
2.2 Prerequisites
2.3 Implementation Process
2.4 Implementation Example
2.5 Response Message

2.1 Instance Introduction

This instance involves only the login, forceLogin, resetSkill, and getAgentEvent interfaces.

The following figure shows the business logic.

Figure 2-1 Agent event polling mode

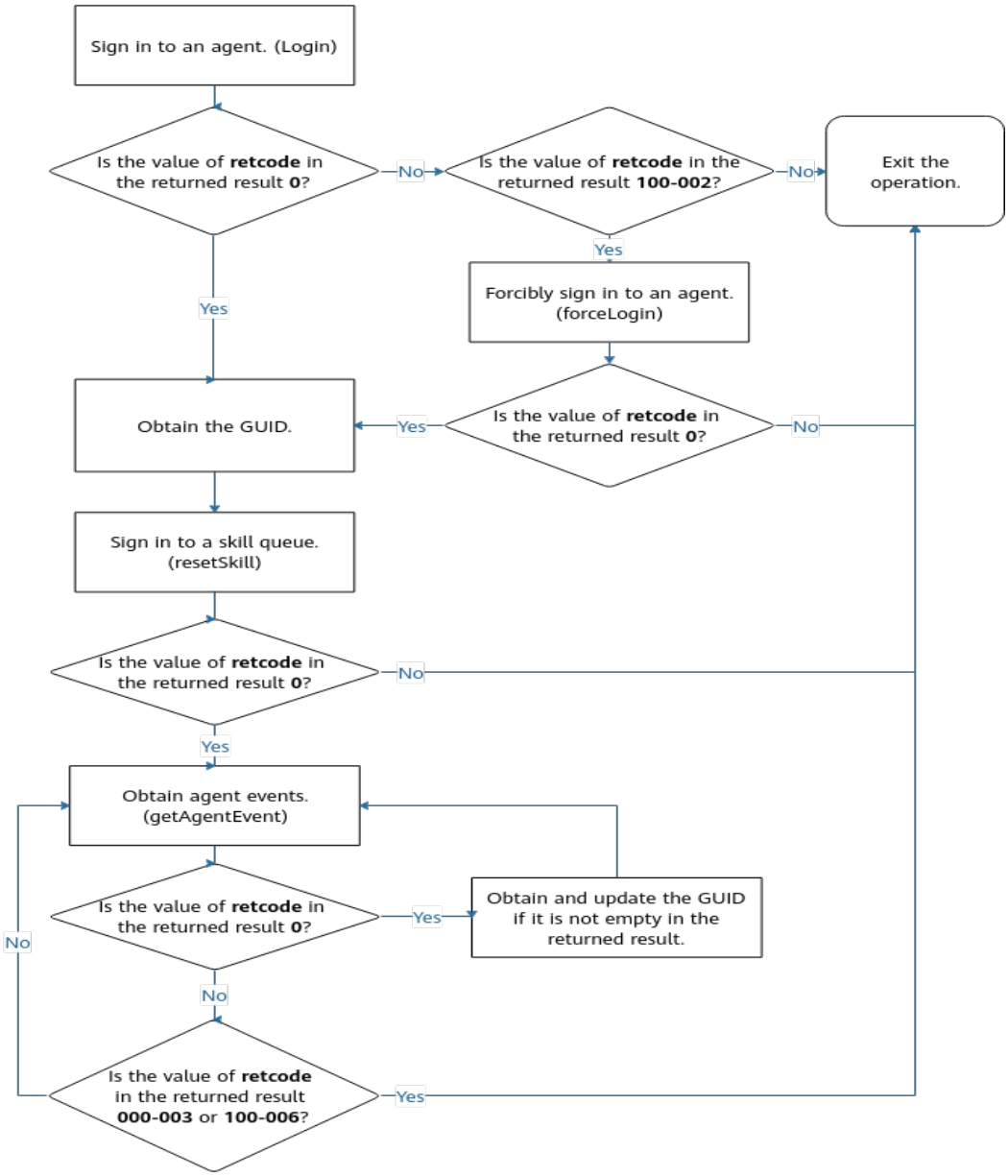
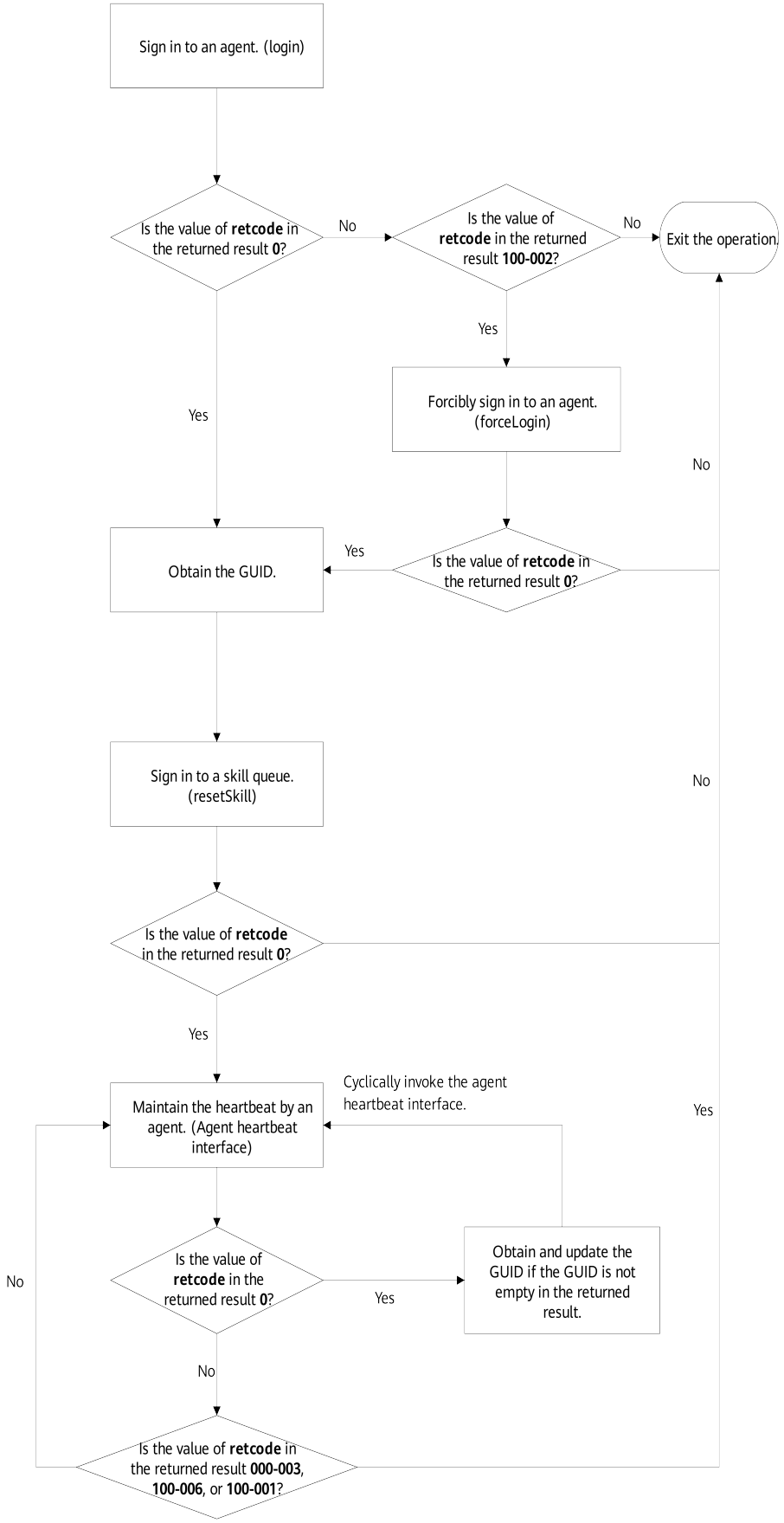


Figure 2-2 Agent event push mode



1. An agent invokes the login interface. If the value of **retcode** is **0**, sign-in is successful. If the value of **retcode** is **100-002**, the sign-in fails because the agent has signed in from another client. The agent can invoke the forceLogin interface to forcibly sign in. If the value of **retcode** is **0**, forcible sign-in is successful.
2. When the agent has successfully signed in, the CC-Gateway adds the **Set-GUID** header to the response headers of the HTTP request. The value of the **Set-GUID** header can be, for example, **JSESSIONID=279fe21d-2caa-4437-84c3-ab9e9dae20f5.AgentGateway0**. In the value, **279fe21d-2caa-4437-84c3-ab9e9dae20f5.AgentGateway0** is the GUID, which is used by the CC-Gateway to authenticate the interfaces invoked by the agent.
3. After obtaining the GUID, the agent can invoke the resetSkill interface to sign in to a specified skill queue.
4. As shown in **Figure 2-1**, the agent event polling mode is used. After signing in to the skill queue, the agent periodically invokes the getAgentEvent interface to obtain agent events in polling mode. This interface is not only used to obtain agent events, but also the heartbeat line between the agent and CC-Gateway.

   As shown in **Figure 2-2**, the agent event push mode is used. After the agent sign-in is successful and the skill queue sign-in is complete, the event is directly pushed to the callback address transferred during sign-in. The agent only needs to invoke the agent heartbeat interface to maintain the heartbeat between the CC-Gateway and the third-party callback address.

   When the CC-Gateway uses dynamic authentication, the GUID is dynamically updated. After the GUID is updated, a **Set-GUID** header is added to the response headers of the HTTP requests of the getAgentEvent interface and agent heartbeat interface. The agent needs to obtain and update the GUID accordingly and uses the new GUID when invoking other interfaces.

2.2 Prerequisites

1. The CC-Gateway has been deployed and commissioned.
2. This instance uses Java as the development language and the Eclipse as the development tool.
3. Start the Eclipse and choose **File** > **New** > **Java Project** to create a Java project.
4. Right-click the Java project and choose **Folder** from the shortcut menu to create the **com.huawei.example** package.
5. Create a **MainTest.java** file in the **com.huawei.example** package.
6. Copy the code lines in section 3.4 "Implementation Example" to the **MainTest.java** file.

2.3 Implementation Process

Involved Interfaces

1. login

Request method: PUT

Request URL: https://ip:port/agentgateway/resource/onlineagent/{agentid}

For details, see **Signing In**.

2. forceLogin

   Request method: PUT

   Request URL: https://ip:port/agentgateway/resource/onlineagent/{agentid}/forcelogin

   For details, see **Forcibly Signing In**.

3. resetSkill

   Request method: POST

   Request URL: https://ip:port/agentgateway/resource/onlineagent/{agentid}/resetskill/{autoflag}?skillid={skillid}&phonelinkage={phonelinkage}

   For details, see **Resetting Skill Queues**.

4. getAgentEvent

   Request method: GET

   Request URL: https://ip:port/agentgateway/resource/agentevent/{agentid}

   For details, see **Obtaining Events of One Agent in Polling Mode**.

Sending an HTTP PUT Request

For details, see the **MainTest.java** file.

```
/**
    * Send http's PUT request
    * @param url  the address of the request
    * @param entityParams the paramters of entity
    * @param headers the field is used to set the header of http request
    * @return
    */
   public Map<string, string> put(string url, Map<string, Object> entityParams, Map<string, string> headers)
```

Sending an HTTP POST Request

For details, see the **MainTest.java** file.

```
/**
    * Send http's POST request
    * @param url the address of the request
    * @param entityParams the paramters of entity
    * @param headers the field is used to set the header of http request
    * @return
    */
    public Map<string, string> post(string url, Map<string, Object> entityParams, Map<string, string>
headers)
```

Sending an HTTP GET Request

For details, see the **MainTest.java** file.

```
/**
    * Send http's POST request
    * @param url the address of the request
    * @param entityParams the paramters of entity
    * @param headers the field is used to set the header of http request
    * @return
```



```
    */
   public Map<string, string> get(string url, Map<string, Object> entityParams, Map<string, string> headers)
```

Implementing the login Interface

For details, see the **MainTest.java** file.

```
   /**
    * log in
    * @param workNo  the work no of the agent
    * @param password the password of the agent
    * @param phoneNumber the phone number of the agent
    * @return
    */
   public Map<string, string> login(string workNo, string password, string phoneNumber)
```

Implementing the forceLogin Interface

For details, see the **MainTest.java** file.

```
/**
    * When agent has logged in, call the interface to forcibly log in
    * @param workNo  the work no of the agent
    * @param password the password of the agent
    * @param phoneNumber the phone number of the agent
    * @return
    */
   public Map<string, string> forceLogin(string workNo, string password, string phoneNumber)
```

Implementing the resetSkill Interface

For details, see the **MainTest.java** file.

```
/**
    * After log in, reset the skills
    * @param workNo the work no of the agent
    * @param autoFlag Is automatically signed into the skill queue
    * @param skillId the id of the skill. if has more than one skill that need to be sign, it's split by;
    * @param headers the field is used to set the header of http request
    * @return
    */
   public Map<string, string> resetSkill(string workNo, boolean autoFlag,
         string skillId, Map<string, string> headers)
```

Implementing the getAgentEvent Interface

For details, see the **MainTest.java** file.

```
/**
* Get the agent's event
* @param workNo workNo the work no of the agent
* @param headers the field is used to set the header of http request
* @return
*/
public Map<string, string> getAgentEvent(string workNo, Map<string, string> headers)
```


2.4 Implementation Example

NOTE

Replace the IP address in the following example with the actual CC-Gateway address when invoking the URL:

- https://*ip*:*port*/agentgateway

  In the URL, *ip* indicates the IP address of the CC-Gateway server, and *port* indicates the HTTPS port number of the CC-Gateway server.
- **WORKNO** indicates the agent ID, **PASSWORD** indicates agent password, and **PHONENUMBER** indicates the softphone number of an agent.
- **IF_TRUST_ALL** indicates whether to trust all certificates. The value can be **false** or **true**. The default value is **false**. You need to load the CC-Gateway client certificate **truststore.jks** and set **TRUSTSTORE_PASSWORD**. The value **true** may cause security risks. Exercise caution when using this value.
- The **truststore.jks** certificate must be stored in the same directory as the compiled **MainTest.class** file.

```
package com.huawei.example;

import com.alibaba.fastjson.JSON;
import lombok.extern.log4j.Log4j2;
import org.bouncycastle.crypto.BlockCipher;
import org.bouncycastle.crypto.engines.AESEngine;
import org.bouncycastle.crypto.prng.SP800SecureRandomBuilder;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import sun.security.provider.Sun;
import org.apache.commons.io.IOUtils;
import org.apache.http.Header;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpPut;
import org.apache.http.config.Registry;
import org.apache.http.config.RegistryBuilder;
import org.apache.http.conn.socket.ConnectionSocketFactory;
import org.apache.http.conn.socket.PlainConnectionSocketFactory;
import org.apache.http.conn.ssl.NoopHostnameVerifier;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.security.KeyStore;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.SecureRandom;
import java.security.Security;
import java.security.cert.CRL;
import java.security.cert.CertPathBuilder;
import java.security.cert.CertStore;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
```



```
import java.security.cert.CollectionCertStoreParameters;
import java.security.cert.PKIXBuilderParameters;
import java.security.cert.PKIXParameters;
import java.security.cert.PKIXRevocationChecker;
import java.security.cert.X509CertSelector;
import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.Collection;
import java.util.EnumSet;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;
import java.util.concurrent.TimeUnit;

import javax.net.ssl.CertPathTrustManagerParameters;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.TrustManagerFactory;
import javax.net.ssl.X509TrustManager;

public class MainTest {
    private static final String AGENTGATEWAY_URL = "https://ip:port/agentgateway";

    private static final String WORKNO = "100";

    private static final String PASSWORD = "xxxxxxxx";

    private static final String PHONENUMBER = "88880001";

    private static final boolean IF_TRUST_ALL = false;

    private static final String TRUSTSTORE_PASSWORD = "******";

    private static final long MAX_IDLE_TIME = 2L;

    private static final String HTTP = "http";

    private static final String HTTPS = "https";

    private static SSLConnectionSocketFactory sslsf = null;

    private static PoolingHttpClientConnectionManager cm = null;

    private static CloseableHttpClient httpClient = null;

    private static final int HTTPCLIENT_CM_MAXCONNECTION = 200;

    private static final int HTTPCLIENT_CM_MAXPERROUTECONNECTION = 20;

    private static final int CONNECT_TIMEOUT = 10000;

    private static final int CONNECTION_REQUEST_TIMEOUT = 10000;

    private static final int SOCKET_TIMEOUT = 20000;

    private static final int AES_KEY_BIT_LENGTH = 256;

    private static final int ENTROPY_SOURCE_BIT_LENGTH = 384;

    private static final boolean FORCE_RESEED = false;

    private static final boolean PREDICTION_RESISTANT = false;

    private static final FastSecureRandomUtil FAST_SECURE_RANDOM = new FastSecureRandomUtil();

    /**
```



```
 * @param args
 */
public static void main(String[] args) {
    MainTest test = new MainTest();
    Map<String, String> resultMap = test.login(WORKNO, PASSWORD, PHONENUMBER);
    if (resultMap == null) {
        System.out.println("Send http request to agentgateway failed");
        return;
    }
    String guid = resultMap.get("guid");
    String agwResultString = resultMap.get("result");
    HashMap<String, Object> agwResult = JSON.parseObject(agwResultString, HashMap.class);
    if (agwResult == null) {
        System.out.println("Parse json to map failed");
        return;
    }


    if ("0".equals(agwResult.get("retcode"))) {
        //log in successfully
        System.out.println("----login ok");
    } else if ("100-002".equals(agwResult.get("retcode"))) {
        //Agent has logged in, but Agent can forcibly log in
        System.out.println("----has login");
        resultMap = test.forceLogin(WORKNO, PASSWORD, PHONENUMBER);
        if (resultMap == null) {
            System.out.println("Send http request to agentgateway failed");
            return;
        }

        guid = resultMap.get("guid");
        agwResultString = resultMap.get("result");
        agwResult = JSON.parseObject(agwResultString, HashMap.class);
        if (agwResult == null) {
            System.out.println("Parse json to map failed");
            return;
        }

        if ("0".equals(agwResult.get("retcode"))) {
            //forcibly log in successfully
            System.out.println("----forceLogin ok");
        } else {
            System.out.println("----forceLogin failed");
            return;
        }
    } else {
        System.out.println("----login failed");
        return;
    }

    //Need to add guid to http request for Authentication after log in
    Map<String, String> headers = new HashMap<String, String>();
    headers.put("guid", guid);

    //After log in,when agent reset skills,  agent can receive call from customer request
    resultMap = test.resetSkill(WORKNO, true, "", headers);
    if (resultMap == null) {
        System.out.println("Send http request to agentgateway failed");
        return;
    }
    agwResultString = resultMap.get("result");
    agwResult = JSON.parseObject(agwResultString, HashMap.class);
    if (agwResult == null) {
        System.out.println("Parse json to map failed");
        return;
    }

    if ("0".equals(agwResult.get("retcode"))) {
        //Reset skills successfully
```



```
            System.out.println("----resetSkill ok");
        } else {
            System.out.println("----resetSkill failed");
            return;
        }

        /**
         *After log in and reset skill successfully. We need to start a thread to get the agent's event by interval.
         */
        Map<String, String> event = null;
        while (true) {
            event = null;
            resultMap = test.getAgentEvent(WORKNO, headers);
            if (resultMap == null) {
                System.out.println("Send http request to agentgateway failed");
                return;
            }

            //if agentgateway uses dynamic authentication mode, the guid will be updated by interval.
            //You can get the guid from the reponse of get the agent's event request
            guid = resultMap.get("guid");
            if (guid != null && guid != "") {
                headers = new HashMap<String, String>();
                headers.put("guid", guid);
            }

            agwResultString = resultMap.get("result");
            agwResult = JSON.parseObject(agwResultString, HashMap.class);
            if (agwResult == null) {
                System.out.println("Parse json to map failed");
                return;
            }

            if ("0".equals(agwResult.get("retcode"))) {
                //Get the agent's event successfully
                event = (Map<String, String>) agwResult.get("event");
                if (event != null) {
                    System.out.println("----getAgentEvent ok:" + agwResultString);
                } else {
                    try {
                        Thread.sleep(500);
                    } catch (InterruptedException e) {
                        System.out.println("getAgentEvent InterruptedException ");
                    }
                }
            } else if ("000-003".equals(agwResult.get("retcode")) ||
"100-006".equals(agwResult.get("retcode"))) {
                //No right to visit the interface
                break;
            } else {
                System.out.println("----getAgentEvent failed");
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    System.out.println("getAgentEvent InterruptedException ");
                }
            }
        }
    }

    private static TrustManager[] trustManagers = new TrustManager[]{
            new X509TrustManager() {
                @Override
                public void checkClientTrusted(X509Certificate[] x509Certificates, String s) throws
CertificateException {
                }

                @Override
```

```
            public void checkServerTrusted(X509Certificate[] x509Certificates, String s) throws
CertificateException {
            }

            @Override
            public X509Certificate[] getAcceptedIssuers() {
                return new X509Certificate[0];
            }
        }
    };

    static {
        InputStream inStream = null;
        InputStream crlInputStream = null;
        SSLContext context = null;
        try {
            context = SSLContext.getInstance("TLSv1.2");

            if (IF_TRUST_ALL) {
                context.init(null, trustManagers, getSecurityRandomInstance());
            } else {

                CertificateFactory certificateFactory = CertificateFactory.getInstance("X.509");
                inStream = MainTest.class.getResourceAsStream("truststore.jks");

                KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
                keyStore.load(inStream, TRUSTSTORE_PASSWORD.toCharArray());

                String crlFile = "";
                if (!crlFile.isEmpty()) {
                    crlInputStream = new FileInputStream(crlFile);
                }
                loadTrustCertificate(context, certificateFactory, keyStore, crlInputStream);
            }

            sslsf = new SSLConnectionSocketFactory(context, new String[]{"TLSv1.2"}, null,
                    NoopHostnameVerifier.INSTANCE);
            httpClient = createHttpClient();
        } catch (Exception e) {
            System.out.println("init occur Exception: " + e.getMessage());
        } finally {
            IOUtils.closeQuietly(inStream, null);
            IOUtils.closeQuietly(crlInputStream, null);
        }
    }

    public static SecureRandom getSecurityRandomInstance() {
        try {
            SecureRandom secureRandom = FAST_SECURE_RANDOM.getSecureRandom();
            secureRandom.setSeed(secureRandom.generateSeed(64));
            return secureRandom;
        } catch (Exception e) {
            System.out.println("get SecureRandom instance failed" + e);
            throw e;
        }
    }

    private static void loadTrustCertificate(SSLContext context, CertificateFactory certificateFactory,
                                             KeyStore keyStore, InputStream crlInputStream) {
        try {
            Collection<CRL> crls = null;
            if (null != crlInputStream) {
                crls = new HashSet();
                crls.add(certificateFactory.generateCRL(crlInputStream));
            }
            TrustManagerFactory tmf = TrustManagerFactory.getInstance("PKIX");
            PKIXParameters pkixParams = new PKIXBuilderParameters(keyStore, new X509CertSelector());
            if (null != crls) {
                List<CertStore> certStores = new ArrayList();
```



```
                certStores.add(CertStore.getInstance("Collection", new CollectionCertStoreParameters(crls)));
                CertPathBuilder certPathBuilder = CertPathBuilder.getInstance("PKIX", new Sun());
                PKIXRevocationChecker revocateChecker = (PKIXRevocationChecker)
certPathBuilder.getRevocationChecker();
                revocateChecker.setOptions(EnumSet.of(PKIXRevocationChecker.Option.PREFER_CRLS,
                        PKIXRevocationChecker.Option.NO_FALLBACK));
                pkixParams.setRevocationEnabled(true);
                pkixParams.setCertStores(certStores);
                pkixParams.addCertPathChecker(revocateChecker);
            } else {
                pkixParams.setRevocationEnabled(false);
            }

            tmf.init(new CertPathTrustManagerParameters(pkixParams));
            context.init(null, tmf.getTrustManagers(), getSecurityRandomInstance());
        } catch (Exception e) {
            System.out.println("loadTrustCertificate occur exception: " + e.getMessage());
        }
    }

    /**
     * createHttpClient
     *
     * @return CloseableHttpClient
     * @throws Exception Exception
     */
    private static CloseableHttpClient createHttpClient() {
        Registry<ConnectionSocketFactory> registry =
RegistryBuilder.<ConnectionSocketFactory>create().register(
                HTTP, new PlainConnectionSocketFactory()).register(HTTPS, sslsf).build();

        cm = new PoolingHttpClientConnectionManager(registry);

        int maxConnect;
        int maxPerRouteConnect;
        String maxConnectNum = "10";
        String maxPreConnectNum = "100";
        try {
            maxConnect = Integer.parseInt(maxConnectNum);
        } catch (NumberFormatException e) {
            maxConnect = HTTPCLIENT_CM_MAXCONNECTION;
        }

        try {
            maxPerRouteConnect = Integer.parseInt(maxPreConnectNum);
        } catch (NumberFormatException e) {
            maxPerRouteConnect = HTTPCLIENT_CM_MAXPERROUTECONNECTION;
        }

        cm.setMaxTotal(maxConnect);
        cm.setDefaultMaxPerRoute(maxPerRouteConnect);

        RequestConfig requestConfig = RequestConfig.custom()
                .setConnectTimeout(CONNECT_TIMEOUT)
                .setConnectionRequestTimeout(CONNECTION_REQUEST_TIMEOUT)
                .setSocketTimeout(SOCKET_TIMEOUT)
                .build();

        httpClient = HttpClients.custom()
                .setDefaultRequestConfig(requestConfig)
                .evictExpiredConnections()
                .evictIdleConnections(MAX_IDLE_TIME, TimeUnit.SECONDS)
                .disableAutomaticRetries()
                .setConnectionManager(cm)
                .build();
        return httpClient;
    }

    /**
```



```
 * log in
 *
 * @param workNo      the work no of the agent
 * @param password    the password of the agent
 * @param phoneNumber the phone number of the agent
 * @return
 */
public Map<String, String> login(String workNo, String password, String phoneNumber) {
    String loginUrl = AGENTGATEWAY_URL + "/resource/onlineagent/" + workNo;
    Map<String, Object> loginParam = new HashMap<String, Object>();
    loginParam.put("password", password);
    loginParam.put("phonenum", phoneNumber);
    return put(loginUrl, loginParam, null);
}

/**
 * When agent has logged in, call the interface to forcibly log in
 *
 * @param workNo      the work no of the agent
 * @param password    the password of the agent
 * @param phoneNumber the phone number of the agent
 * @return
 */
public Map<String, String> forceLogin(String workNo, String password, String phoneNumber) {
    String loginUrl = AGENTGATEWAY_URL + "/resource/onlineagent/" + workNo + "/forcelogin";
    Map<String, Object> loginParam = new HashMap<String, Object>();
    loginParam.put("password", password);
    loginParam.put("phonenum", phoneNumber);
    return put(loginUrl, loginParam, null);
}

/**
 * After log in, reset the skills
 *
 * @param workNo   the work no of the agent
 * @param autoFlag Is automatically signed into the skill queue
 * @param skillId  the id of the skill. if has more than one skill that need to be sign, it is split by ;
 * @param headers  the field is used to set the header of http request
 * @return
 */
public Map<String, String> resetSkill(String workNo, boolean autoFlag,
                                      String skillId, Map<String, String> headers) {
    String url = AGENTGATEWAY_URL + "/resource/onlineagent/" + workNo + "/resetskill/" + autoFlag;
    if (skillId != null && skillId != "") {
        url = url + "?skillid=" + skillId;
    }
    return post(url, null, headers);
}

/**
 * Get the agent's event
 *
 * @param workNo  workNo the work no of the agent
 * @param headers the field is used to set the header of http request
 * @return
 */
public Map<String, String> getAgentEvent(String workNo, Map<String, String> headers) {
    String url = AGENTGATEWAY_URL + "/resource/agentevent/" + workNo;
    return get(url, headers);
}

/**
 * Send http's PUT request
 *
 * @param url          the address of the request
 * @param entityParams the paramters of entity
 * @param headers      the field is used to set the header of http request
 * @return
 */
```



```
    public Map<String, String> put(String url, Map<String, Object> entityParams, Map<String, String>
headers) {
        CloseableHttpClient httpClient = null;
        Map<String, String> resultMap = null;
        try {
            httpClient = createHttpClient();

            HttpPut httpPut = new HttpPut(url);
            if (entityParams != null) {
                String jsonString = JSON.toJSONString(entityParams);
                HttpEntity entity = new StringEntity(jsonString);
                httpPut.setEntity(entity);

            }

            if (headers != null) {
                Set<Entry<String, String>> headersSet = headers.entrySet();
                for (Entry<String, String> entry : headersSet) {
                    httpPut.setHeader(entry.getKey(), entry.getValue());
                }
            }
            httpPut.setHeader("Content-Type", "application/json");
            HttpResponse response = httpClient.execute(httpPut);
            int returnCode = response.getStatusLine().getStatusCode();
            if (returnCode == HttpStatus.SC_OK) {
                InputStream is = response.getEntity().getContent();
                BufferedReader in = new BufferedReader(new InputStreamReader(is));
                StringBuffer buffer = new StringBuffer();
                String line = "";
                while ((line = in.readLine()) != null) {
                    buffer.append(line);
                }

                Header[] allHeaders = response.getAllHeaders();
                String guid = "";
                if (allHeaders != null && allHeaders.length > 0) {
                    for (Header header : allHeaders) {
                        if (header.getName().equals("Set-GUID")) {
                            String setGuid = header.getValue();
                            if (setGuid != null) {
                                guid = setGuid.replace("JSESSIONID=", "");
                            }
                            break;
                        }
                    }
                }
                resultMap = new HashMap<String, String>();
                resultMap.put("guid", guid);
                resultMap.put("result", buffer.toString());
            } else {
                System.out.println(returnCode);
            }
            return resultMap;
        } catch (ClientProtocolException e) {
            System.out.println("HttpPut ClientProtocolException ");
        } catch (IOException e) {
            System.out.println("HttpPut IOException ");
        } catch (Exception e) {
            System.out.println("HttpPut Exception ");
        } finally {
            httpClient.getConnectionManager().shutdown();
        }
        return resultMap;
    }


    /**
     * Send http's POST request
     *
```



```
    * @param url          the address of the request
    * @param entityParams the paramters of entity
    * @param headers      the field is used to set the header of http request
    * @return
    */
   public Map<String, String> post(String url, Map<String, Object> entityParams, Map<String, String>
headers) {
       CloseableHttpClient httpClient = null;
       Map<String, String> resultMap = null;
       try {
           httpClient = createHttpClient();

           HttpPost httpPost = new HttpPost(url);
           if (entityParams != null) {
               String jsonString = JSON.toJSONString(entityParams);
               HttpEntity entity = new StringEntity(jsonString);
               httpPost.setEntity(entity);

           }

           if (headers != null) {
               Set<Entry<String, String>> headersSet = headers.entrySet();
               for (Entry<String, String> entry : headersSet) {
                   httpPost.setHeader(entry.getKey(), entry.getValue());
               }
           }
           httpPost.setHeader("Content-Type", "application/json");
           HttpResponse response = httpClient.execute(httpPost);
           int returnCode = response.getStatusLine().getStatusCode();
           if (returnCode == HttpStatus.SC_OK) {
               InputStream is = response.getEntity().getContent();
               BufferedReader in = new BufferedReader(new InputStreamReader(is));
               StringBuffer buffer = new StringBuffer();
               String line = "";
               while ((line = in.readLine()) != null) {
                   buffer.append(line);
               }
               resultMap = new HashMap<String, String>();
               resultMap.put("result", buffer.toString());
           } else {
               System.out.println(returnCode);
           }
           return resultMap;
       } catch (ClientProtocolException e) {
           System.out.println("HttpPost ClientProtocolException");
       } catch (IOException e) {
           System.out.println("HttpPost IOException");
       } catch (Exception e) {
           System.out.println("HttpPost Exception");
       } finally {
           httpClient.getConnectionManager().shutdown();
       }
       return resultMap;
   }


   /**
    * Send http's GET request
    *
    * @param url     the address of the request
    * @param headers the field is used to set the header of http request
    * @return
    */
   public Map<String, String> get(String url, Map<String, String> headers) {
       CloseableHttpClient httpClient = null;
       Map<String, String> resultMap = null;
       try {
           httpClient = createHttpClient();
```



```
            HttpGet httpGet = new HttpGet(url);
            if (headers != null) {
                Set<Entry<String, String>> headersSet = headers.entrySet();
                for (Entry<String, String> entry : headersSet) {
                    httpGet.setHeader(entry.getKey(), entry.getValue());
                }
            }
            httpGet.setHeader("Content-Type", "application/json");
            HttpResponse response = httpClient.execute(httpGet);
            int returnCode = response.getStatusLine().getStatusCode();
            if (returnCode == HttpStatus.SC_OK) {
                InputStream is = response.getEntity().getContent();
                BufferedReader in = new BufferedReader(new InputStreamReader(is));
                StringBuffer buffer = new StringBuffer();
                String line = "";
                while ((line = in.readLine()) != null) {
                    buffer.append(line);
                }

                Header[] allHeaders = response.getAllHeaders();
                String guid = "";
                if (allHeaders != null && allHeaders.length > 0) {
                    for (Header header : allHeaders) {
                        if (header.getName().equals("Set-GUID")) {
                            String setGuid = header.getValue();
                            if (setGuid != null) {
                                guid = setGuid.replace("JSESSIONID=", "");
                            }
                            break;
                        }
                    }
                }
                resultMap = new HashMap<String, String>();
                resultMap.put("guid", guid);
                resultMap.put("result", buffer.toString());
            } else {
                System.out.println(returnCode);
            }
            return resultMap;
        } catch (ClientProtocolException e) {
            System.out.println("HttpGet ClientProtocolException");
        } catch (IOException e) {
            System.out.println("HttpGet IOException");
        } catch (Exception e) {
            System.out.println("HttpGet Exception");
        } finally {
            httpClient.getConnectionManager().shutdown();
        }
        return resultMap;
    }
}

package com.huawei.example;

import lombok.extern.log4j.Log4j2;

import org.bouncycastle.crypto.BlockCipher;
import org.bouncycastle.crypto.engines.AESEngine;
import org.bouncycastle.crypto.prng.SP800SecureRandomBuilder;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.Security;

@Log4j2
public class FastSecureRandomUtil {
    private static final int AES_KEY_BIT_LENGTH = 256;
```

```
    private static final int ENTROPY_SOURCE_BIT_LENGTH = 384;

    private static final boolean FORCE_RESEED = false;

    private static final boolean PREDICTION_RESISTANT = false;

    static {
        if (Security.getProvider(BouncyCastleProvider.PROVIDER_NAME) == null) {
            Security.addProvider(new BouncyCastleProvider());
        }
    }

    private final SecureRandom fastRandom;

    public FastSecureRandomUtil() throws Exception {
        SecureRandom source = null;
        try {
            source = SecureRandom.getInstanceStrong();
        } catch (NoSuchAlgorithmException e) {
            log.error("FastSecureRandom NoSuchAlgorithmException: {}.", e.getMessage());
            throw new Exception("FastSecureRandom NoSuchAlgorithmException.", e);
        }

        BlockCipher cipher = new AESEngine();

        byte[] nonce = new byte[ENTROPY_SOURCE_BIT_LENGTH / Byte.SIZE];
        source.nextBytes(nonce);

        fastRandom = new SP800SecureRandomBuilder(source,
PREDICTION_RESISTANT).setEntropyBitsRequired(
                ENTROPY_SOURCE_BIT_LENGTH).buildCTR(cipher, AES_KEY_BIT_LENGTH, nonce,
FORCE_RESEED);
    }

    public byte[] getRandomBytes(int byteSize) {
        byte[] randomBytes = new byte[byteSize];
        fastRandom.nextBytes(randomBytes);
        return randomBytes;
    }


    public SecureRandom getSecureRandom() {
        return fastRandom;
    }
}
```

2.5 Response Message

The following response messages are returned if the agent has not logged in from any client:

```
----login ok
----resetSkill ok
----getAgentEvent ok:{"message":"","retcode":"0","event":
{"content":null,"eventType":"AgentOther_InService","workNo":"291"}}
----getAgentEvent ok:{"message":"","retcode":"0","event":
{"content":null,"eventType":"AgentState_Ready","workNo":"291"}}
```

The following response messages are returned if the agent has logged in from another client:

```
----has login
----forceLogin ok
----resetSkill ok
----getAgentEvent ok:{"message":"","retcode":"0","event":
```


{"content":null,"eventType":"AgentOther_InService","workNo":"291"}}
----getAgentEvent ok:{"message":"","retcode":"0","event":
{"content":null,"eventType":"AgentState_Ready","workNo":"291"}}

3 Basic App Development

3.1 Viewing Interface Invoking Logs
3.2 Basic Functions and Configurations
3.3 Basic Voice Calls
3.4 Functions Used After Call Setup
3.5 Real-time Quality Check
3.6 Voice Playback
3.7 Basic Multimedia Calls

3.1 Viewing Interface Invoking Logs

- All interface request logs are recorded in the **/home/elpis/tomcat/logs/agentgateway/agentgateway-rest.log** file.

3.2 Basic Functions and Configurations

3.2.1 Signing In

1. After the sign-in is performed by the interface in **Signing In**, the parameters related to the sign-in are stored in the request content and are identified in JSON format. The following is an example:
   ```
   {"password":"xxxxxxxx","phonenum":"6001"}
   ```
   In the preceding information, the **password** field indicates the sign-in password, and the **phonenum** field indicates the phone number of the agent.
2. After the agent successfully signs in, the following information is displayed:
   ```
   {"retcode":"0","msg":"",result:{"workno":"6001","mediatype":"TTT"}}
   ```
   In the preceding information, the **mediatype** field indicates whether the agent successfully signs in to the CTI server, WebM, and MailM. **T** indicates successful sign-in, and **F** indicates sign-in failure. Voice and video call businesses can be handled only after the agent successfully signs in to the CTI

Server. Text chat businesses can be handled only after the agent successfully signs in to the WebM. Currently, the agent cannot sign in to the MailM.

3. If the agent fails to sign in and the value of **retcode** is **100-002**, the agent has signed in and the interface in **Forcibly Signing In** is invoked to forcibly sign in.

3.2.2 (Optional) Querying Skills Configured for an Agent

After an agent signs in, the agent can invoke the interface in **Querying Configured Skill Queues** to obtain the skill queues that the agent can sign in to.

3.2.3 Signing In to a Skill Queue

An agent uses **Resetting Skill Queues** to sign in all skill queues configured for the agent or signs in to a specified skill queue.

3.2.4 Obtaining Agent Events

If an agent successfully signs in to a skill queue, the agent can start to provide services. In this case, the agent client needs to start a timer (200 milliseconds is recommended) to obtain an event of the agent in polling mode.

- Invoke the interface in **Obtaining Events of One Agent in Polling Mode** to obtain and process an event.

3.2.5 Sign Out an Agent

After an agent signs in to the system, the agent can invoke the interface in **Signing Out** to sign out. The following information indicates that the interface is invoked successfully:

{"retcode":"0","msg":"", result:null}

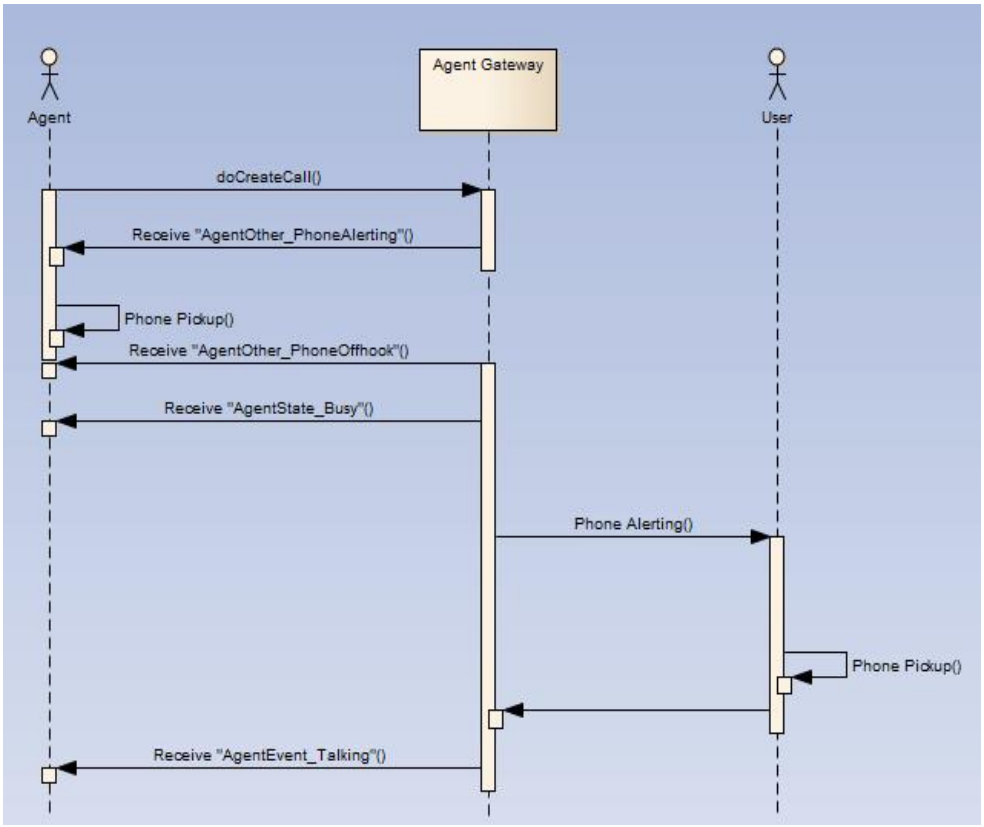
3.3 Basic Voice Calls

3.3.1 Making an Outgoing Call

- Application scenario

  An agent makes an outgoing call.

Figure 3-1 Outgoing call process



- Prerequisites
  - The agent has signed in.
  - The agent is not answering any call (except a held call).
- Implementation process

  For details, see **Making a Common Outbound Call**.
- Triggered event
  - When the agent who must pick up the phone to answer a call receives a call

    AgentOther_PhoneAlerting
  - When the agent picks up the phone

    AgentOther_PhoneOffhook

    AgentState_Busy

    AgentEvent_Customer_Alerting
  - When the agent hangs up the phone or does not pick up the phone

    AgentEvent_Call_Out_Fail

    AgentEvent_Connect_Fail

    AgentOther_PhoneRelease
  - When the called party picks up the phone

    AgentEvent_Talking, indicating that the call is successful.
  - When the called party hangs up the phone or does not pick up the phone

    AgentOther_PhoneRelease

AgentEvent_Call_Out_Fail

AgentEvent_Connect_Fail

AgentState_Ready

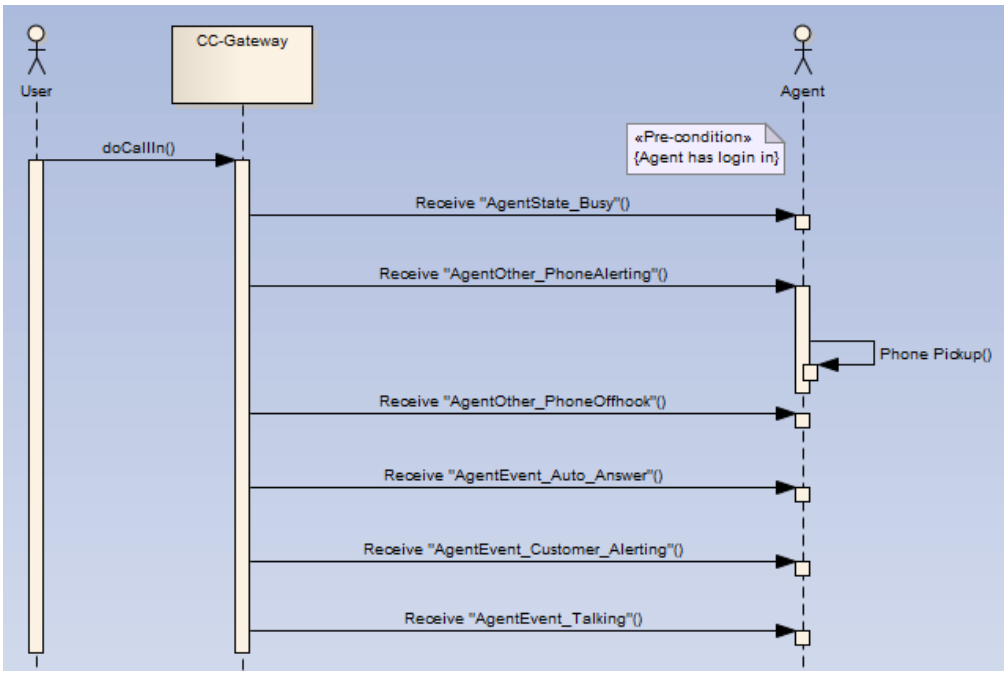
3.3.2 Answering a Call

3.3.2.1 Specifying Whether to Enable Automatic Answering

- Application scenario

  This function is used to specify whether an agent must click the answer button of the agent service software to answer an incoming call after picking up the phone. If this function is enabled, the agent does not need to click the answer button.

  Figure 3-2 Incoming call flow - automatic answer



- Prerequisites
  - The agent has signed in.
- Implementation process

  Set whether to enable automatic answering to true for sign-in based on **Signing In**, or invoke the interface in **Setting Automatic Answering** to enable automatic answering.
- Triggered event

  AgentState_Busy, indicating that the agent is presold.
  - When the agent who must pick up the phone to answer a call receives a call

    AgentOther_PhoneAlerting
  - When the agent picks up the phone

    AgentOther_PhoneOffhook

AgentEvent_Auto_Answer

AgentEvent_Customer_Alerting

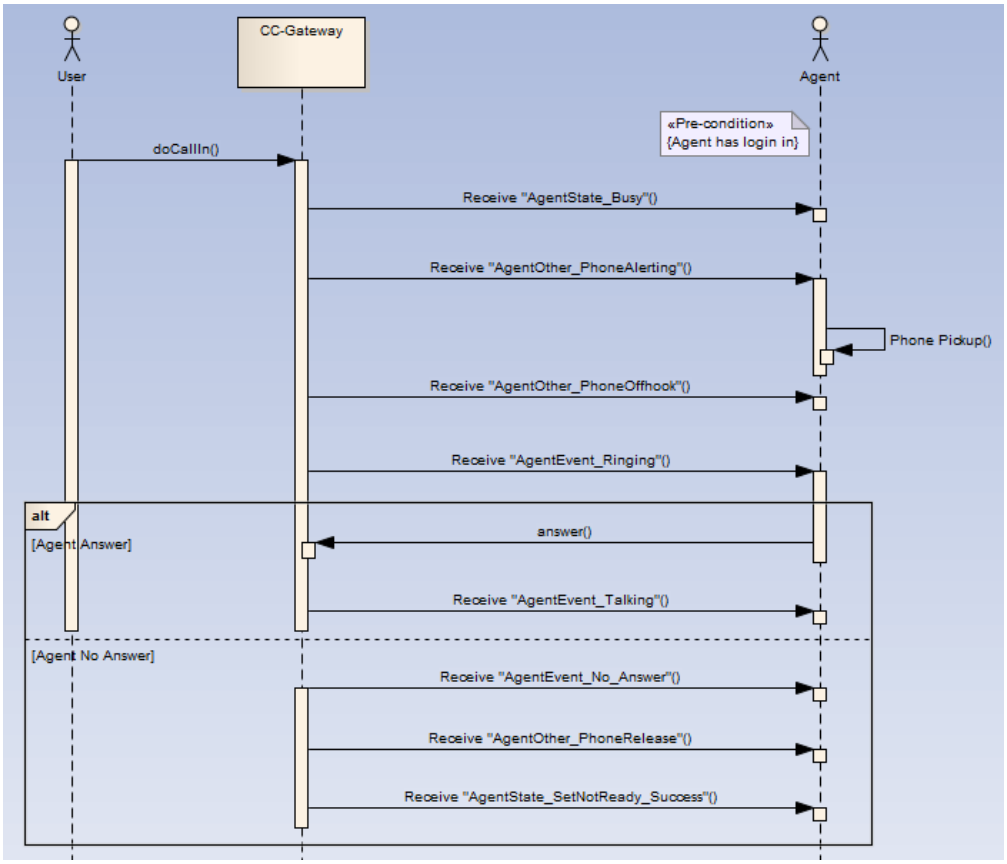
AgentEvent_Talking, indicating that the call is successfully connected.

3.3.2.2 Specifying Whether to Enable Manual Answering

- Application scenario

  This function is used to specify whether an agent must click the answer button of the agent service software to answer an incoming call after picking up the phone. If this function is enabled, the agent needs to click the answer button.

  Figure 3-3 Incoming call process - manual answer



- Prerequisites
  - The agent has signed in.
- Implementation process

  Set whether to enable automatic answering to false for sign-in based on **Signing In**, or invoke the interface in **Setting Automatic Answering** to enable manual answering. After an incoming call request is received, the interface in **Answering a Call** is invoked to answer the call.
- Triggered event

  AgentState_Busy, indicating that the agent is presold.
  - When the agent who must pick up the phone to answer a call receives a call

AgentOther_PhoneAlerting

- When the agent picks up the phone

  AgentOther_PhoneOffhook

  AgentEvent_Ringing, indicating that an incoming call request is received.

  AgentEvent_Customer_Alerting
- After the agent invokes the call answering interface

  AgentEvent_Talking, indicating that the call is successfully connected.
- When the interface invocation for answering a call times out

  AgentEvent_No_Answer

3.3.2.3 Querying Call Information Before Answering a Call

- Application scenario

  When a non-automatic answering is set for an agent, the agent needs to know the skill queue from which a call comes and the number of the subscriber before the agent answers the call.
- Prerequisites
  - The agent has signed in.
  - The CTI platform has allocated a call to the agent. (The agent phone is ringing or the attendant picks up the phone.)
- Implementation process

  For details, see **Querying Call Information Before Answering a Call**.

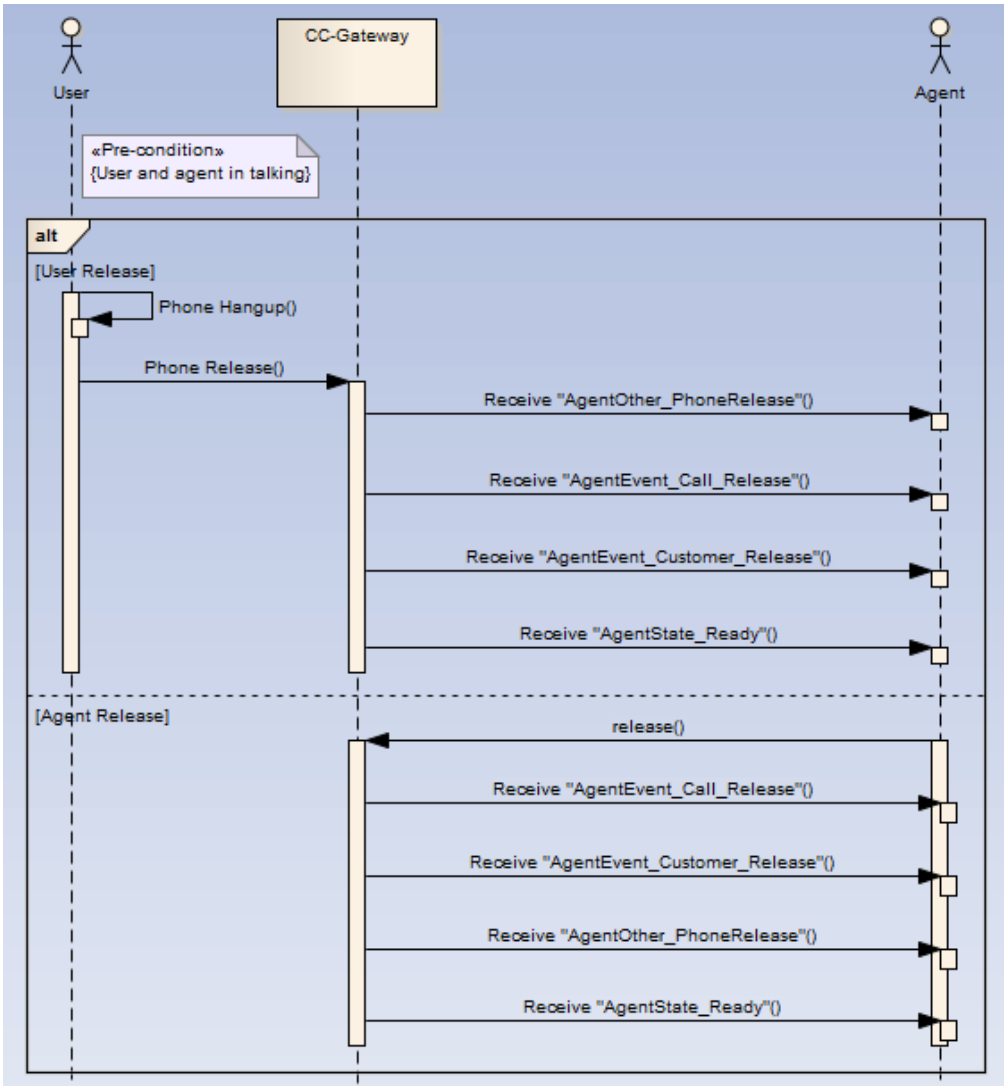
3.4 Functions Used After Call Setup

3.4.1 Releasing a Call

- Application scenario

  An agent must release a call.

Figure 3-4 Releasing a call



- Prerequisites
  The agent has an ongoing call.
- Implementation process
  For details, see **Dropping a Call**.
- Triggered event
  - AgentEvent_Call_Release
  - AgentEvent_Customer_Release
  - AgentOther_PhoneRelease
  - AgentState_Ready

3.4.2 Holding or Unholding a Call

3.4.2.1 Holding a Call

- Application scenario
  An agent must hold an outgoing or incoming call.

- Prerequisites

  The agent has an ongoing call.
- Implementation process

  Holding a Call
- Triggered event

  AgentEvent_Hold

3.4.2.2 Unholding a Call

- Application scenario

  An agent must unhold a call.
- Prerequisites

  The agent is holding a call.
- Implementation process

  For details, see **Unholding a Call**.
- Triggered event

  AgentEvent_Talking

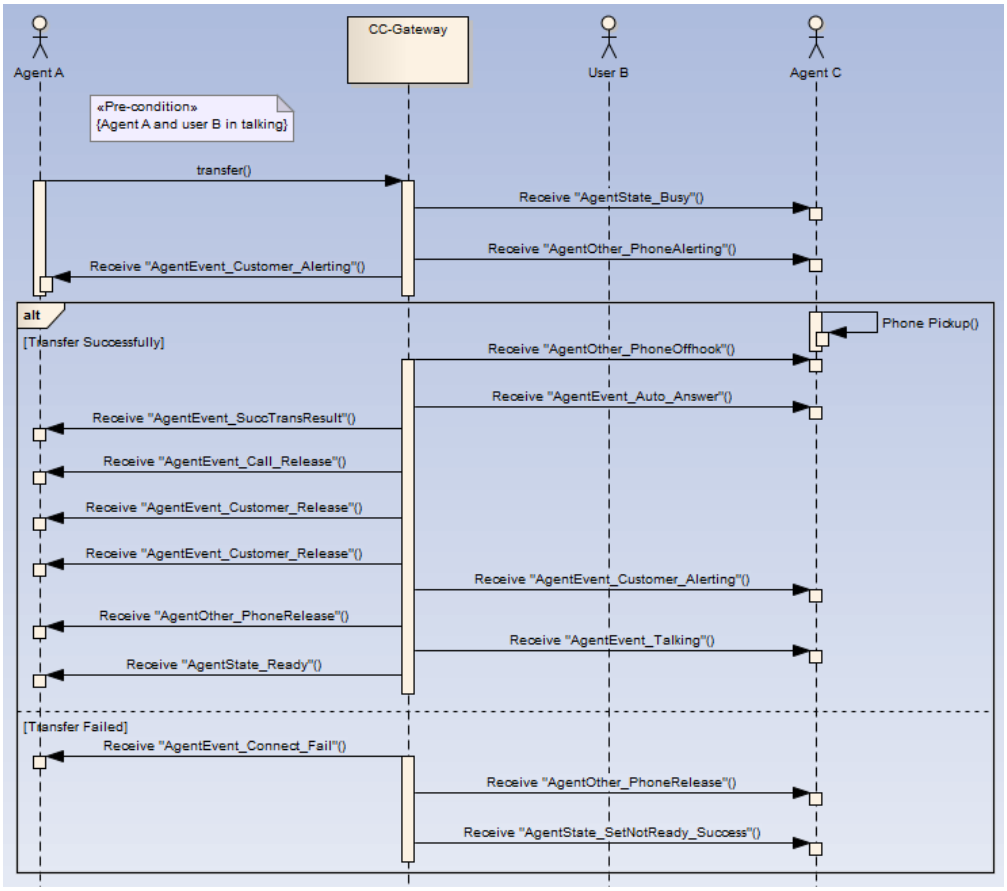
3.4.3 Transferring a Call

- Application scenario

  After a call between agent A and user B is set up, the agent must transfer the call to another agent or an external line. Call transfer is classified into internal transfer and external transfer.
  - Internal transfer indicates that a call is transferred to an internal device such as the agent, IVR, skill queue, and access code.
    - Success transfer and blind transfer for transferring a call to another agent
    - Success transfer for transferring a call to another skill queue
    - Blind transfer and hang-up transfer for transferring a call to another IVR
    - Success transfer and blind transfer for transferring a call to another access code
  - External transfer indicates that a call is transferred to an external user, that is, a phone number. External transfer supports talk transfer and three-party transfer, that is, agent A, user B, and user C form a third-party call (in the three-party call formed by talk transfer, user B is actually in the holding state). After the talk transfer or three-party transfer is successful:
    - If agent A releases the call first, a call is set up between user B and user C.
    - If user B releases the call first, a call is set up between user C and agent A.

- If user C releases the call first, a call is set up between user B and agent A.
- If agent A must retrieve the call with user B, agent A invokes the interface described in **Canceling Transfer** to retrieve the call. The call with user C is ended, and agent A talks with user B.

Figure 3-5 Transferring a call to another agent – success transfer process



- Prerequisites

  The agent is making a call.
- Implementation process

  For details, see **Transferring a Call**.
- Triggered events
  - AgentEvent_Customer_Alerting
  - AgentEvent_Call_Release
  - AgentEvent_Customer_Release
  - AgentOther_PhoneRelease
  - AgentState_Ready
  - Success transfer: AgentEvent_SuccTransResult
  - Talk transfer or three-party transfer: AgentEvent_Conference

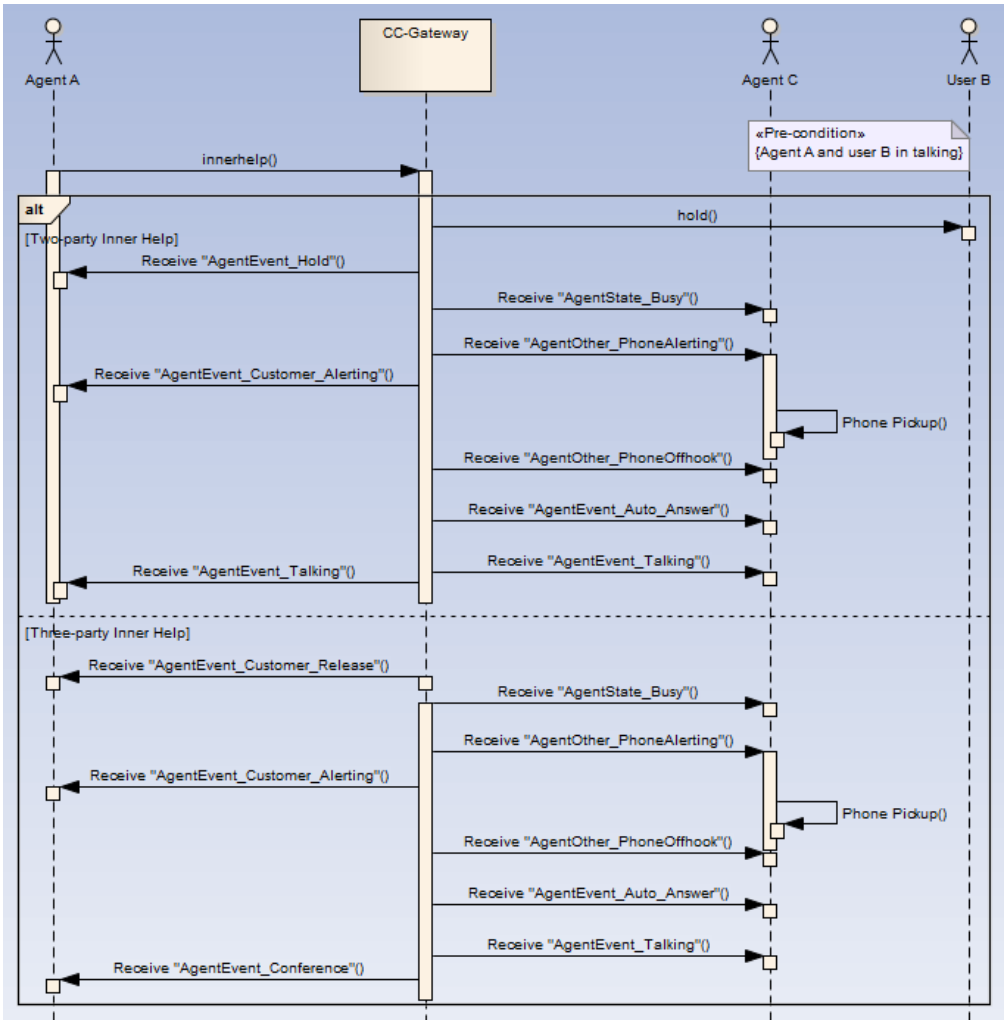
3.4.4 Asking for Internal Help

- Application scenario

  Agent A and user B have set up a call, and agent A must ask for internal help from another agent (such as agent C). Internal help is classified into two-party help and three-party help.

  - Two-party help: After agent C answers the internal help call from agent A, the call of user B is held, and agent A and agent C are in a call. After the two-party help request is successfully processed:
    - If agent C releases the call first, the call between agent A and user B is resumed.
    - If agent A releases the call first, the call of user B is transferred to agent C.
    - To retrieve the call with user B, agent A invokes the interface in **Disconnecting a Call of a Specified ID** based on **callId** of the call between agent A and agent C.
  - Three-party help: After agent C answers the internal help call from agent A, the agent A, user B, and agent C are in a call. After the three-party help request is successfully processed:
    - If agent A releases the call first, user B and agent C form a two-party call.
    - If agent C releases the call first, user B and agent A form a two-party call.
    - If user B releases the call first, agent A and agent C form an internal call.
    - If agent A must retrieve the call with user B, agent A invokes the interface in **Releasing a Connection of a Specified Number** (the released number input at the interface is the ID of agent C).

Figure 3-6 Asking for internal help



- Prerequisites
  - The agent is making a call with a customer.
- Implementation process

  For details, see **Asking for Internal Help**.
- Triggered event
  - Asking for help during two-party conversation

    AgentEvent_Hold

    AgentEvent_Customer_Alerting

    AgentEvent_Talking
  - Asking for help during three-party conversation

    AgentEvent_Customer_Alerting

    AgentEvent_Customer_Release

    AgentEvent_Conference

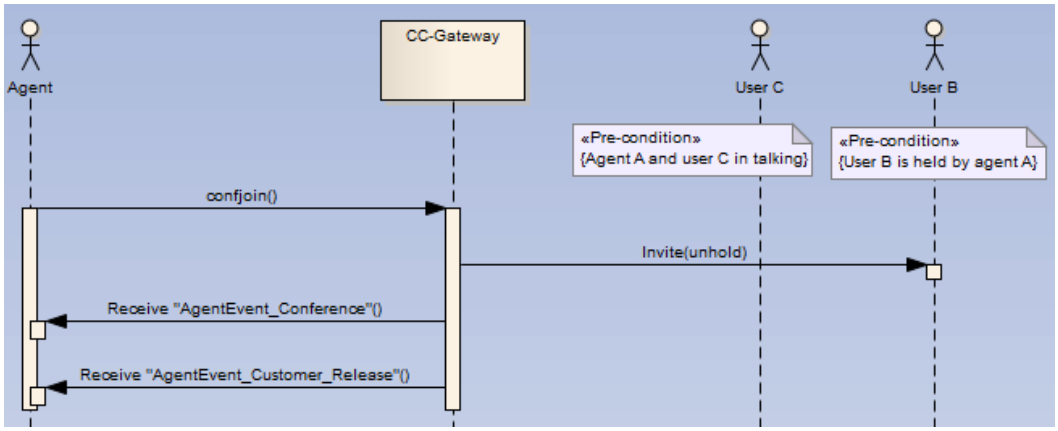
3.4.5 Three-Party Call

- Application scenario

After holding the call with user B, agent A makes an outbound call to user C. After setting up a call with user C, agent A adds user B to the call with user C, to form a three-party call. After the three-party call is successfully set up:

- If agent A releases the call first, a call is set up between user B and user C.
- If user B releases the call first, a call is set up between agent A and user C.
- If user C releases the call first, a call is set up between agent A and user B.
- If agent A must retrieve the call with user B, agent A invokes the interface described in **Releasing a Connection of a Specified Number** (the released number input at the interface is the number of user C).

Figure 3-7 Three-party call



- Prerequisites
  - A call has been held by the agent.
  - The agent is in the call state.
- Implementation process

  For details, see **Making a Three-Party Call**.
- Triggered event

  AgentEvent_Conference

  AgentEvent_Customer_Release

3.4.6 Muting

3.4.6.1 Starting Muting

- Application scenario

  An agent must prevent a customer from hearing the agent's voice.
- Prerequisites

  An agent has an ongoing call with a customer.

  An internal call, a three-party call, and a call that an agent makes to ask for help during three-party conversation cannot be muted.

- Implementation process
  For details, see **Muting a Call**.
- Triggered event
  None

3.4.6.2 Stopping Muting

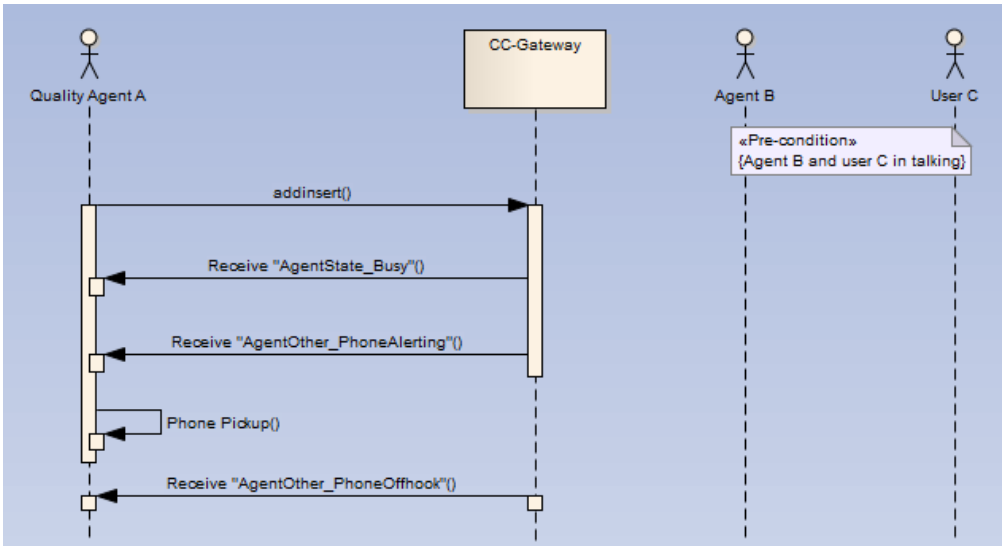
- Application Scenario
  The agent needs to enable the customer to hear the agent.
- Prerequisites
  The agent has a muted call with a customer.
- Implementation
  For details, see **Unmuting a Muted Call**.
- Triggering Event
  None

3.5 Real-time Quality Check

3.5.1 Inserting

- Application scenario
  After QC inspector A signs in, inspector A is inserted to the voice call between agent B and customer C to perform real-time quality check, forming a three-party conversation among the customer, agent, and QC inspector in the same site.

  Figure 3-8 Insert flowchart



- Prerequisites
  The QC inspector has signed in.
  A specified agent has an ongoing voice call with a customer.

- Implementation process

  For details, see **Inserting**.
- Triggered event

  When the QC inspector receives an event:

  AgentState_Busy, indicating that the agent is presold.
  - When the agent who must pick up the phone to answer a call receives a call

    AgentOther_PhoneAlerting
  - When the agent picks up the phone

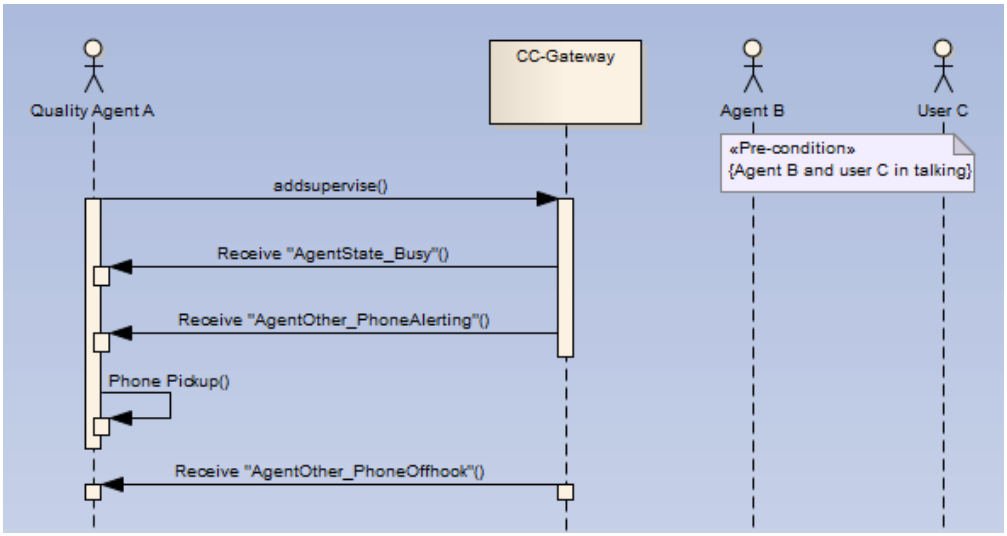
    AgentOther_PhoneOffhook

3.5.2 Listening On

- Application scenario

  After signing in, inspector A listens to the voice call between agent B and user C and performs real-time inspection. Inspector A can hear the voice of agent B and customer C. However, agent B and customer C cannot hear the voice of inspector A.

  Figure 3-9 Listening flowchart



- Prerequisites

  The inspector has signed in.

  A specified agent has an ongoing call with a customer.
- Implementation process

  For details, see **Listening**.
- Triggered event

  When the inspector receives an event:

  AgentState_Busy, indicating that the agent is presold.
  - When the agent who must pick up the phone to answer a call receives a call

    AgentOther_PhoneAlerting

  - When the agent picks up the phone

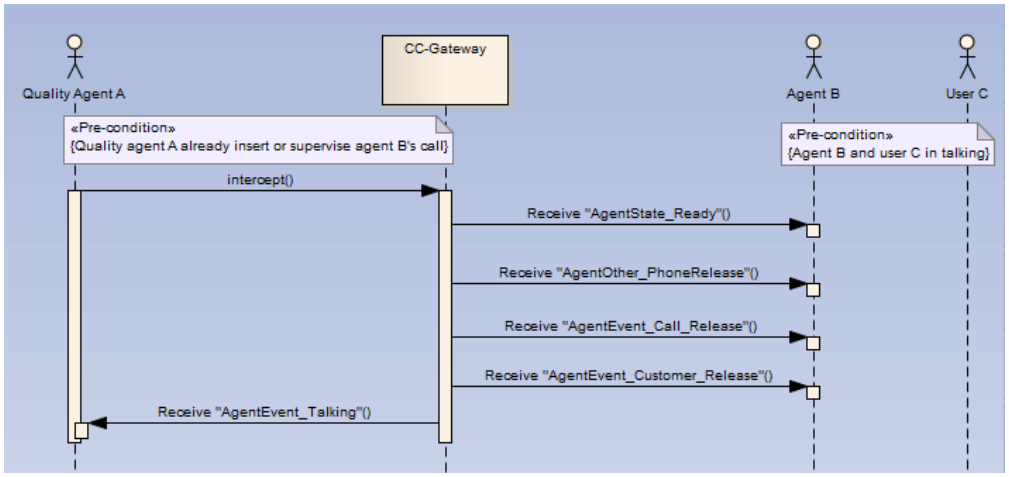
    AgentOther_PhoneOffhook

3.5.3 Intercepting

- Application scenario

  After signing in, inspector A intercepts the call between agent B and user C, talks with user C, and releases agent B.

  Figure 3-10 Interception flowchart



- Prerequisites

  The inspector has signed in.

  The inspector has performed the insertion, listening, or whisper operation on a specified agent.

  A specified agent has an ongoing call with a customer.

- Implementation process

  For details, see **Intercepting**.

- Triggered event

  When the inspected agent receives the following events:

  AgentState_Ready

  AgentOther_PhoneRelease

  AgentEvent_Call_Release

  AgentEvent_Customer_Release

  When the inspector receives an event:

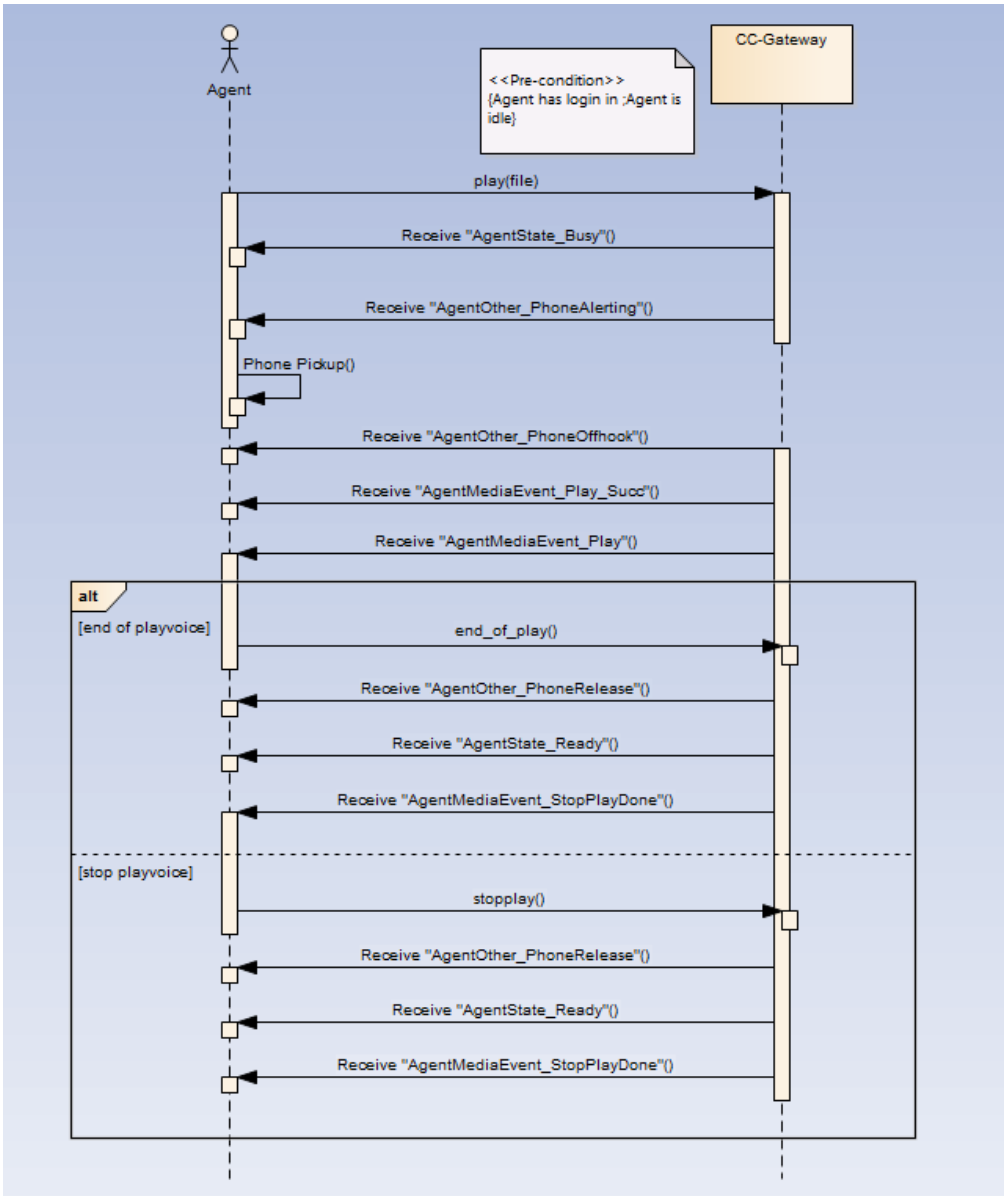
  AgentEvent_Talking

3.6 Voice Playback

3.6.1 Playing Voice in Non-calling State

- Application scenario

An agent needs to play a recording.



- Prerequisites

  The agent has signed in.

  The agent is not in a call.
- Implementation process

  For details, see **Recording Playback**.
- Triggered event

  Voice playback:
  - AgentState_Busy
  - AgentOther_PhoneAlerting
  - AgentOther_PhoneOffhook
  - AgentMediaEvent_Play_Succ
  - AgentMediaEvent_Play

Voice playback failure:

- AgentState_Busy
- AgentOther_PhoneAlerting
- AgentOther_PhoneOffhook
- AgentOther_PhoneRelease
- AgentState_Ready
- AgentMediaEvent_Play_Fail
- AgentMediaEvent_StopPlayDone

Voice playback pause

- AgentMediaEvent_PausePlayDone

Voice playback resumption:

- AgentMediaEvent_ResumePlayDone

Fast forwarding of voice playback

- AgentMediaEvent_JumpPlayDone

Fast rewinding of voice playback

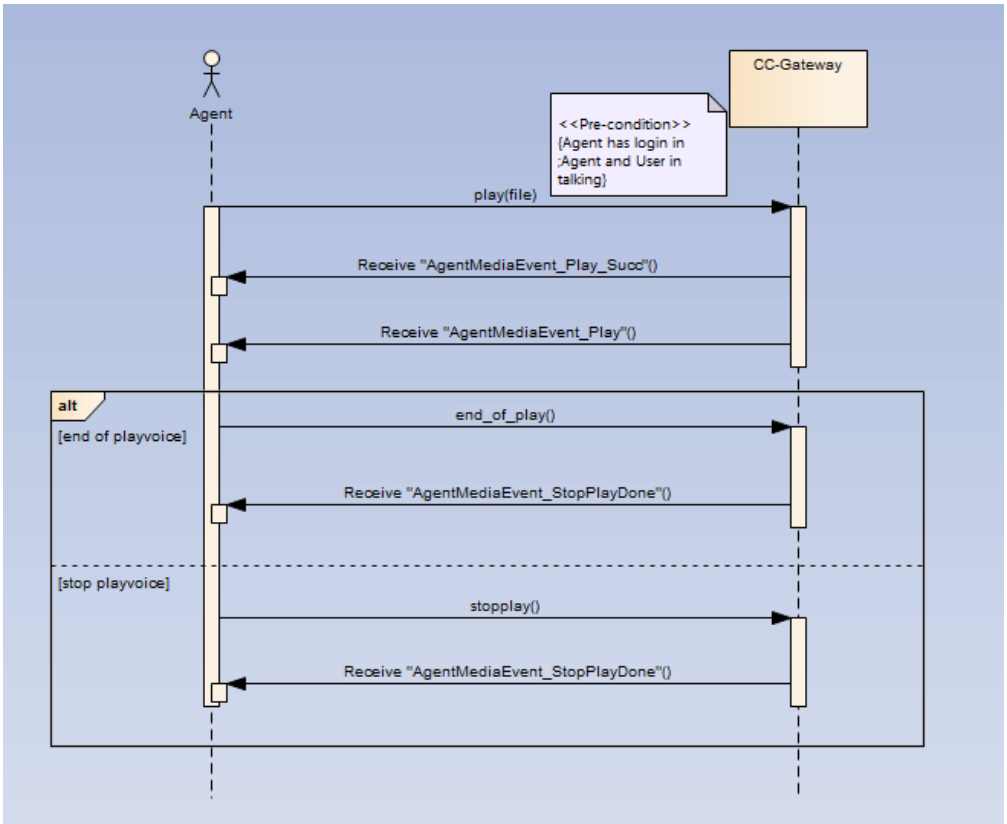
- AgentMediaEvent_JumpPlayDone

Voice playback stopping or completion

- AgentOther_PhoneRelease
- AgentState_Ready
- AgentMediaEvent_StopPlayDone

3.6.2 Playing Voice in Calling State

- Application scenario
  An agent needs to play a recording.



- Prerequisites

  The agent has signed in.

  The agent is in a call.
- Implementation process

  For details, see **Recording Playback**.
- Triggered event

  Voice playback:
  - AgentMediaEvent_Play_Succ
  - AgentMediaEvent_Play

  Voice playback failure:
  - AgentMediaEvent_Play_Fail
  - AgentMediaEvent_StopPlayDone

  Fast forwarding of voice playback
  - AgentMediaEvent_JumpPlayDone

  Fast rewinding of voice playback
  - AgentMediaEvent_JumpPlayDone

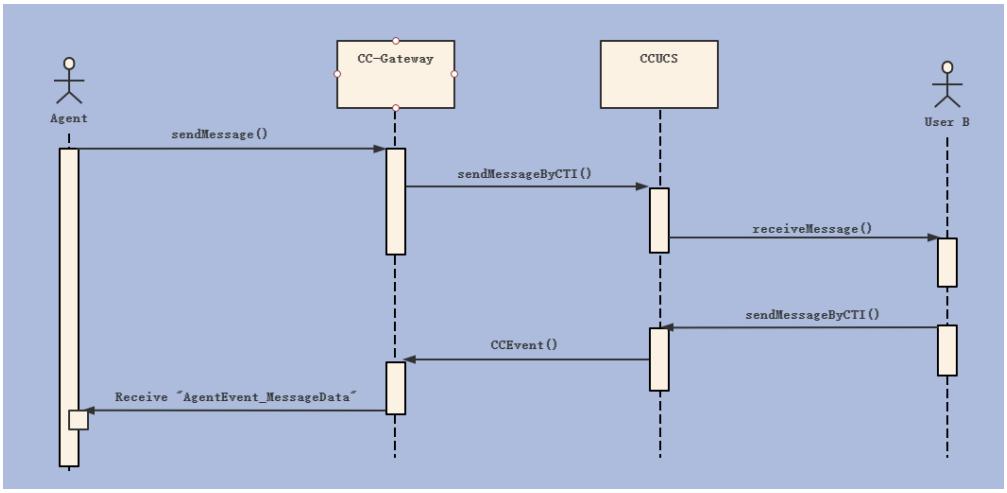
  Voice playback stopping or completion
  - AgentMediaEvent_StopPlayDone

3.7 Basic Multimedia Calls

3.7.1 Message Sending and Receiving

- Application Scenario

  An agent can have a multimedia text chat with a subscriber.



- Prerequisites
  - An agent has signed in to the system and the multimedia server of the call center platform.
  - A multimedia call is allocated to the agent.
- Implementation process

  For details about how an agent sends a message, see **Sending Messages: sendmessage**.
- Message receiving event

Multimedia message event (AgentEvent_MessageData)

