

**QuecPython Cloudspeaker Software Design Guide**

**LTE Standard/LPWA Module Series**

Version: 1.0.0

Date: 2023-02-27

 Status: Preliminary

**At Quectel, our aim is to provide timely and comprehensive services to our customers. If you require any assistance, please contact our headquarters:**

**Quectel Wireless Solutions Co., Ltd.**

Building 5, Shanghai Business Park Phase III (Area B), No.1016 Tianlin Road, Minhang District, Shanghai 200233, China

Tel: +86 21 5108 6236

Email: [info@quectel.com](mailto:info@quectel.com)

**Or our local offices. For more information, please visit:**

<http://www.quectel.com/support/sales.htm>.

**For technical support, or to report documentation errors, please visit:**

<http://www.quectel.com/support/technical.htm>.

Or email us at: [support@quectel.com](mailto:support@quectel.com).

**Legal Notices**

We offer information as a service to you. The provided information is based on your requirements and we make every effort to ensure its quality. You agree that you are responsible for using independent analysis and evaluation in designing intended products, and we provide reference designs for illustrative purposes only. Before using any hardware, software or service guided by this document, please read this notice carefully. Even though we employ commercially reasonable efforts to provide the best possible experience, you hereby acknowledge and agree that this document and related services hereunder are provided to you on an “as available” basis. We may revise or restate this document from time to time at our sole discretion without any prior notice to you.

**Use and Disclosure Restrictions**

**License Agreements**

Documents and information provided by us shall be kept confidential, unless specific permission is granted. They shall not be accessed or used for any purpose except as expressly provided herein.

**Copyright**

Our and third-party products hereunder may contain copyrighted material. Such copyrighted material shall not be copied, reproduced, distributed, merged, published, translated, or modified without prior written consent. We and the third party have exclusive rights over copyrighted material. No license shall be granted or conveyed under any patents, copyrights, trademarks, or service mark rights. To avoid ambiguities, purchasing in any form cannot be deemed as granting a license other than the normal non-exclusive, royalty-free license to use the material. We reserve the right to take legal action for noncompliance with abovementioned requirements, unauthorized use, or other illegal or malicious use of the material.

**Trademarks**

Except as otherwise set forth herein, nothing in this document shall be construed as conferring any rights to use any trademark, trade name or name, abbreviation, or counterfeit product thereof owned by Quectel or any third party in advertising, publicity, or other aspects.

**Third-Party Rights**

This document may refer to hardware, software and/or documentation owned by one or more third parties (“third-party materials”). Use of such third-party materials shall be governed by all restrictions and obligations applicable thereto.

We make no warranty or representation, either express or implied, regarding the third-party materials, including but not limited to any implied or statutory, warranties of merchantability or fitness for a particular purpose, quiet enjoyment, system integration, information accuracy, and non-infringement of any third-party intellectual property rights with regard to the licensed technology or use thereof. Nothing herein constitutes a representation or warranty by us to either develop, enhance, modify, distribute, market, sell, offer for sale, or otherwise maintain production of any our products or any other hardware, software, device, tool, information, or product. We moreover disclaim any and all warranties arising from the course of dealing or usage of trade.

**Privacy Policy**

To implement module functionality, certain device data are uploaded to Quectel’s or third-party’s servers, including carriers, chipset suppliers or customer-designated servers. Quectel, strictly abiding by the relevant laws and regulations, shall retain, use, disclose or otherwise process relevant data for the purpose of performing the service only or as permitted by applicable laws. Before data interaction with third parties, please be informed of their privacy and data security policy.

**Disclaimer**

1. We acknowledge no liability for any injury or damage arising from the reliance upon the information.
2. We shall bear no liability resulting from any inaccuracies or omissions, or from the use of the information contained herein.
3. While we have made every effort to ensure that the functions and features under development are free from errors, it is possible that they could contain errors, inaccuracies, and omissions. Unless otherwise provided by valid agreement, we make no warranties of any kind, either implied or express, and exclude all liability for any loss or damage suffered in connection with the use of features and functions under development, to the maximum extent permitted by law, regardless of whether such loss or damage may have been foreseeable.
4. We are not responsible for the accessibility, safety, accuracy, availability, legality, or completeness of information, advertising, commercial offers, products, services, and materials on third-party websites and third-party resources.

***Copyright © Quectel Wireless Solutions Co., Ltd. 2023. All rights reserved.***

# About the Document

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Description** |
| - | 2023-02-27 | Pawn ZHOU | Creation of the document |
| 1.0.0 | 2023-02-27 | Pawn ZHOU | Preliminary |

# Contents

[About the Document 3](#_Toc128418580)

[Contents 4](#_Toc128418581)

[Table Index 5](#_Toc128418582)

[Figure Index 6](#_Toc128418583)

[1 Introduction 7](#_Toc128418584)

[1.1. Applicable Modules 7](#_Toc128418585)

[2 System Framework 8](#_Toc128418586)

[2.1. Hardware Framework 8](#_Toc128418589)

[2.2. Software Framework 8](#_Toc128418590)

[3 Key Components 10](#_Toc128418591)

[3.1. EventMesh 12](#_Toc128418592)

[3.2. AudioManager 13](#_Toc128418593)

[3.3. ConfigStoreManager 15](#_Toc128418594)

[3.4. LteNetManager 16](#_Toc128418595)

[3.5. DeviceInfoManager 19](#_Toc128418596)

[3.6. OtaManager 20](#_Toc128418597)

[3.7. AliYunManage 21](#_Toc128418598)

[3.8. GLight 22](#_Toc128418599)

[3.9. DeviceActionManager 23](#_Toc128418600)

[4 System Initialization Process 26](#_Toc128418601)

[5 Business Process 27](#_Toc128418602)

[6 Example 28](#_Toc128418603)

[7 Appendix Terms and Abbreviations 32](#_Toc128418604)

# Table Index

Table 1: Applicable Modules 7

Table 2: Terms and Abbreviations 32

# Figure Index

Figure 1: Hardware Framework 8

Figure 2: Software Framework 9

Figure 3: Software Code UML Class Diagram 11

Figure 4: EventMesh Middleware Flow Diagram 12

Figure 5: Cloudspeaker System Initialization Flowchart 26

Figure 6: Business Flow Chart 27

# 

1. Introduction

This document describes the design framework of QuecPython Cloudspeaker, including the software and hardware framework, function descriptions of key components, system initialization process, business process and function examples. It helps users quickly understand the overall architecture and functions of QuecPython Cloudspeaker.

* 1. Applicable Modules

Table 1: Applicable Modules

|  |  |
| --- | --- |
| **Module Family** | **Module** |
| LTE Standard | EC21-AUX |
| EC200U Series |
| EG915U Series |
| EC200A Series |
| NB-IoT | BC65 |
| LPWA | BG95 Series |
| BG77 |

1. System Framework

* 1. Hardware Framework

The hardware system framework is as follows:

The module supports hardware interfaces such as SIM card, GPIO and UART.



Figure 1: Hardware Framework

* 1. Software Framework

The software system framework is as follows:

1. APP layer processes the core business of Cloudspeaker and parses the uplink and downlink data.
2. EventMesh is an event processor that supports event subscription and publication to trigger and execute business functions.
3. After receiving external events or data, the module calls the event or data processor registered by upper-layer through EventMesh to process the event or data.



Figure 2: Software Framework

1. Key Components

The following is a UML class diagram describing the dependencies and inheritance relationships between components in the project software code. Cloudspeaker is taken as the top-level object, and it is associated with the dependent components. As shown below:



Figure 3: Software Code UML Class Diagram

* 1. EventMesh
* **Function description**

EventMesh is a dynamic infrastructure middleware. In event-driven architecture, an event refers to a change, action or observation in the system. Then a notification is generated and used in the various processor functions that respond to the event.

* **Principle**

EventMesh middleware flow diagram is as follows. Please contact Quectel Technical Support for details.



Figure 4: EventMesh Middleware Flow Diagram

**NOTE**

|  |
| --- |
| The upper-layer business function and the actual executive function communicate through EventMesh, and all events in the system come from the EventStore (event pool). Note: EventStore is used to store user registration events in the EventMesh framework. EventStore is not shown in the open interfaces. Therefore, the EventStore in the figure above is only used to show the basic implementation principle of EventMesh. |

* **Subscribe to an event**

|  |
| --- |
| import EventMesh  def test(event, msg):  return msg  # Function for subscribing to an event  EventMesh.subscribe("test\_event", test) |

* **Publishing an event**

|  |
| --- |
| import EventMesh  def test(event, msg):  return msg  # Function for subscribing to an event  EventMesh.subscribe("test\_event", test)  # Publish an event  EventMesh.publish("test\_event", "OK") |

* 1. AudioManager
* **Function description**

Control the audio output of a device, including TTS voice broadcast, audio file playback, audio volume settings and other functions.

* **Principle**

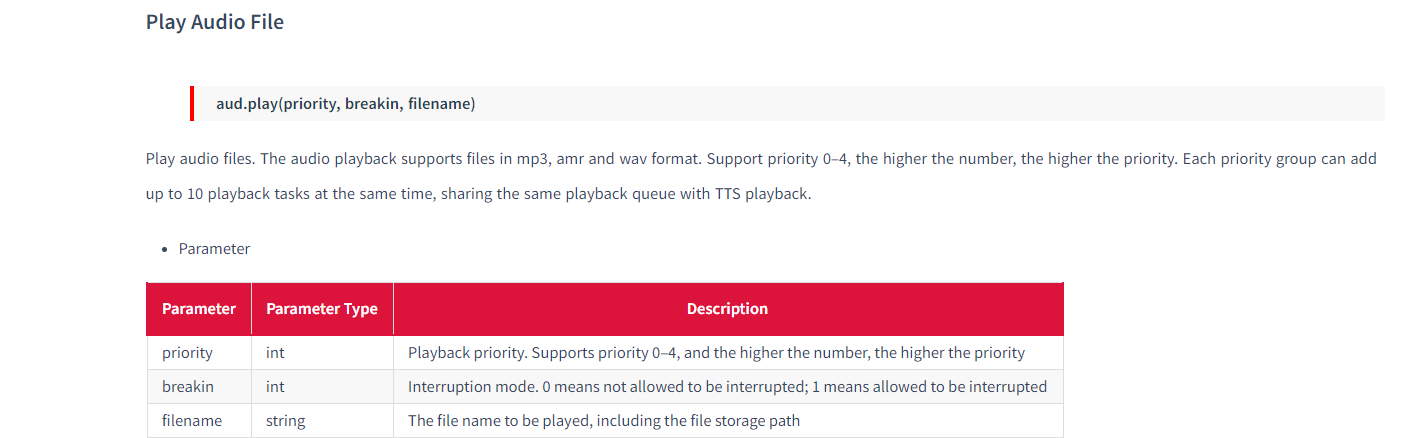
1. Create a class method to initialize the audio function through *\_\_init\_\_()*, and reduce the original volume level to 5 levels for easy use.

|  |
| --- |
| class AudioManager(Abstract):  """  audio initialization  audio file playback  TTS broadcast management  """  def \_\_init\_\_(self):  self.\_\_audio = audio.Audio(0)  self.\_\_tts = audio.TTS(0)  self.\_\_audio\_volume = 3  self.\_\_tts\_priority = 2  self.\_\_tts\_breakin = 0  self.\_\_tts\_mode = 2  self.\_\_volume\_level = {  1: 1,  2: 3,  3: 6,  4: 9,  5: 11  }  self.log = get\_logger(\_\_name\_\_ + "." + self.\_\_class\_\_.\_\_name\_\_) |

1. After the class method is initialized, the audio function initialization is registered as an event through EventMesh to complete the volume initialization setting.

|  |
| --- |
| def post\_processor\_after\_initialization(self):  self.\_\_set\_audio\_pa()  vol\_num = EventMesh.publish("persistent\_config\_get", "vol\_num")  self.\_\_audio\_volume = vol\_num  # Set TTS volume  self.\_\_audio.setVolume(self.\_\_volume\_level.get(self.\_\_audio\_volume))  EventMesh.subscribe("audio\_play", self.audio\_play)  EventMesh.subscribe("tts\_play", self.tts\_play)  EventMesh.subscribe("get\_audio\_state", self.get\_audio\_state)  EventMesh.subscribe("get\_tts\_state", self.get\_tts\_state)  EventMesh.subscribe("add\_audio\_volume", self.add\_audio\_volume)  EventMesh.subscribe("reduce\_audio\_volume", self.reduce\_audio\_volume) |

1. Play audio file through the audio API provided by QuecPython. See <https://python.quectel.com/wiki/#/> for details about the audio API.



|  |
| --- |
| def audio\_play(self, topic=None, filename=None):  """Play audio"""  if filename is None:  return  state = self.\_\_audio.play(self.\_\_tts\_priority, self.\_\_tts\_breakin, filename)  return True if state == 0 else False |

1. Play the input content through TTS API provided by QuecPython. Contact Quectel Technical Support for details about the TTS API.

|  |
| --- |
| def tts\_play(self, topic=None, content=None):  """Play tts  \_\_tts\_priority Playback priority. The priority ranges from 0 to 4. A larger value indicates a higher priority.  \_\_tts\_breakin Interrupt mode. *0* indicates that interruptions are not allowed and *1* indicates that interruptions are allowed.  \_\_tts\_mode Encoding scheme. 2 - UTF-8  content Broadcast content  """  if content is None:  return  state = self.\_\_tts.play(self.\_\_tts\_priority, self.\_\_tts\_breakin, self.\_\_tts\_mode, content)  return True if state == 0 else False |

* 1. ConfigStoreManager
* **Function description**

Store device parameters persistently and provide methods for reading and writing file data.

* **Principle**

1. When a class method is initialized, it automatically checks whether the profile exists. If the profile does not exist, a JSON file is automatically created. After the file is created, default parameters are automatically written into the file. If the profile already exists, it automatically checks whether there are parameters added to the default parameters and updates the profile if there is any new parameter.
2. This method uses EventMesh to register two events: reading file and updating file.

|  |
| --- |
| def post\_processor\_after\_initialization(self):  if ql\_fs.path\_exists(self.file\_name):  file\_map = ql\_fs.read\_json(self.file\_name)  for k in self.map.keys():  if k not in file\_map:  file\_map.update({k: self.map.get(k)})  self.\_\_store(msg=file\_map)  self.map = file\_map  else:  self.\_\_store()  EventMesh.subscribe("persistent\_config\_get", self.\_\_read)  EventMesh.subscribe("persistent\_config\_store", self.\_\_store) |

* 1. LteNetManager
* **Function description**

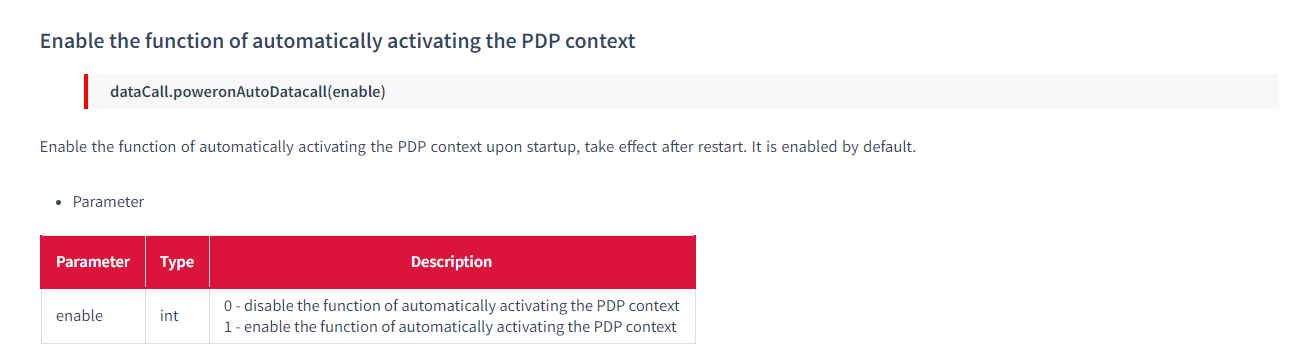
Initialize the module network and manage the network state. Automatic dial-up is used for the device by default. If you want to manually dial up or use the configured APN for dial-up, please refer to <https://python.quectel.com/wiki>.

* **Principle**

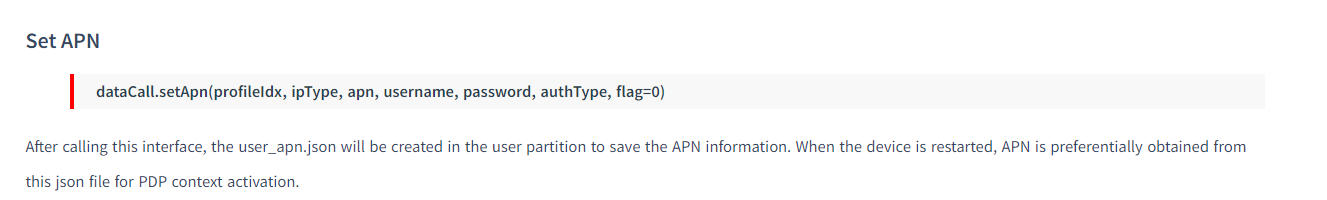
1. When the class method is initialized, it waits for the device to complete the network registration, registers the network change callback function, and determines the device's network registration status based on the return value of checkNet API. After a successful network registration, the TCP connection is started by means of event publication. If the network registration fails, it will try to register the network again. See <https://python.quectel.com/wiki> for details about the CheckNet API.

|  |
| --- |
| def wait\_connect(self, timeout):  """Wait for network registration"""  self.log.info("wait net -----------")  stagecode, subcode = self.check\_net.wait\_network\_connected(timeout)  if stagecode == 3 and subcode == 1:  # Network registration succeeded  EventMesh.publish("tts\_play", TTS\_CONTENT.STR\_CONNECT\_NET\_OK)  self.log.info("module net success, run mqtt connect")  EventMesh.publish('ota\_check')  EventMesh.publish('mqtt\_connect')  self.net\_error\_audio\_stop()  else:  # Network registration failed  self.\_\_net\_error\_mode = 1  self.log.error("module net fail, wait try again")  EventMesh.publish("tts\_play", TTS\_CONTENT.SSTR\_CONNECT\_NET\_FAILED)  self.net\_fail\_process()  self.\_\_data\_call.setCallback(self.net\_state\_cb) |

1. Switch to automatic dial-up.



1. Set APN.



1. Handle the abnormal network status.

When the device network status changes, you can register the callback function to notify the APP layer, as shown below:

|  |
| --- |
| def wait\_connect(self, timeout):  """ Wait for network registration """  self.log.info("wait net -----------")  stagecode, subcode = self.check\_net.wait\_network\_connected(timeout)  if stagecode == 3 and subcode == 1:  # Network registration succeeded  EventMesh.publish("tts\_play", TTS\_CONTENT.STR\_CONNECT\_NET\_OK)  self.log.info("module net success, run mqtt connect")  EventMesh.publish('ota\_check')  EventMesh.publish('mqtt\_connect')  self.net\_error\_audio\_stop()  else:  # Network registration failed  self.\_\_net\_error\_mode = 1  self.log.error("module net fail, wait try again")  EventMesh.publish("tts\_play", TTS\_CONTENT.SSTR\_CONNECT\_NET\_FAILED)  self.net\_fail\_process()  self.\_\_data\_call.setCallback(self.net\_state\_cb) |

|  |
| --- |
| def net\_state\_cb(self, args):  """The callback function is triggered when the network status changes"""  nw\_sta = args[1]  if nw\_sta == 1:  EventMesh.publish("tts\_play", TTS\_CONTENT.STR\_CONNECT\_NET\_OK)  self.log.info("network connected!")  self.net\_error\_audio\_stop()  else:  self.net\_error\_audio\_start()  EventMesh.publish("tts\_play", TTS\_CONTENT.SSTR\_CONNECT\_NET\_FAILED)  self.log.info("network not connected!") |

The following is an example of reconnecting the network when the network status is abnormal:

If the network is abnormal, cfun is used to disable the network and then enable the network, and then register the network again. If the network registration through cfun fails, the module will automatically reboot. To limit the number of reconnection attempts, you can set a value in the profile.

|  |
| --- |
| def net\_fail\_process(self):  # If network registration fails, try cfun and register the network again. If network registration through cfun fails, the module reboots.  state = net.setModemFun(0)  if state == -1:  self.log.error("cfun net mode error, device will restart.")  utime.sleep(5)  # Power.powerRestart()  state = net.setModemFun(1)  if state == -1:  self.log.error("cfun net mode error, device will restart.")  utime.sleep(5)  # Power.powerRestart()  self.log.info("cfun net mode success, note the net again")  self.wait\_connect(30) |

* 1. DeviceInfoManager
* **Function description**

Obtain basic device information, such as the IMEI of the device and the ICCID of the SIM card.

* **Principle**

1. After the class method is initialized, the functions that obtain device information are registered as events through EventMesh. You can directly use these functions by publishing events.

|  |
| --- |
| def post\_processor\_after\_instantiation(self):  # Register events  EventMesh.subscribe("get\_sim\_iccid", self.get\_iccid)  EventMesh.subscribe("get\_device\_imei", self.get\_imei)  EventMesh.subscribe("get\_fw\_version", self.get\_device\_fw\_version)  EventMesh.subscribe("get\_csq", self.get\_csq) |

1. Use the SIM API to obtain the ICCID of the current SIM card.

|  |
| --- |
| def get\_iccid(self, event=None, msg=None):  """Query ICCID"""  if self.\_\_iccid == "":  msg = sim.getIccid()  if msg != -1:  self.\_\_iccid = msg  else:  self.log.warn("get sim iccid fail, please check sim")  return self.\_\_iccid |

1. See <https://python.quectel.com/wiki> for details about the device management API.
   1. OtaManager

* **Function description**

Update the application code remotely.

* **Principle**

1. Perform upgrade through QuecCloud IoT platform.
2. After creating an upgrade plan on QuecCloud IoT platform, power on the device. After the device successfully registers on the network, check whether an upgrade plan exists through API.
3. You can use the OTA upgrade detection function by publishing an event.

|  |
| --- |
| def post\_processor\_after\_initialization(self):  # Register an event  EventMesh.subscribe("ota\_check", self.check\_ota\_event) |

1. After the upgrade plan is detected, the download interface is called to download the upgrade file. After the upgrade file is downloaded, the module automatically reboots to complete the upgrade.

|  |
| --- |
| def upgrade\_fota\_sh(self, upgrade\_path):  try:  action = self.upgrade\_info["action"]  url = self.upgrade\_info["url"]  except Exception as e:  return self.status\_code\_dict["ERROR"][4]  try:  if action:  self.report("DOWNLOADING\_FIRMWARE")  fota = app\_fota.new()  fota.download(url, upgrade\_path)  self.report("DOWNLOADED\_NOTIFY\_UPDATE")  fota.set\_update\_flag()  self.report("UPDATE\_START")  return self.status\_code\_dict["UPGRADE"][0]  else:  return self.status\_code\_dict["UPGRADE"][1]  except Exception as e:  self.report(PROCESS\_CODE[5])  return self.status\_code\_dict["ERROR"][2] |

* 1. AliYunManage
* **Function description**

The module is connected to Alibaba Cloud IoT platform through MQTT protocol to realize the functions of connecting the platform, disconnecting from the platform, and transmitting uplink and downlink data.

* **Principle**

1. Create a class method to initialize class properties through *\_\_init\_\_()*. Class properties include ProductKey, DeviceName and DeviceSecret for connecting to Alibaba Cloud IoT platform, keep-alive time and others.

|  |
| --- |
| class AliYunManage(Abstract):  '''  MQTT interface  '''  def \_\_init\_\_(self):  self.\_\_mqtt\_client = None  self.product\_key = '' # Product jey  self.product\_secret = None # Product secret  self.device\_name = '' # Device name  self.device\_secret = '' # Device secret  self.client\_id = '' # User defined  self.clean\_session = True # Client type (False: long time, True: temporary)  self.keep\_alive = 300 # keep-alive time (s)  self.sub\_topic = '' # Subscription address  self.qos = 1 # Message service quality 0: The sender sends the message once and does not retry 1: The sender sends the message at least once, ensuring that it reaches the Broker  self.conn\_flag = False  self.start\_mqtt\_flag = False  self.log = get\_logger(\_\_name\_\_ + "." + self.\_\_class\_\_.\_\_name\_\_) |

1. After the class method is initialized, the method that can be called by other functions are registered in EventMesh through event registration.

|  |
| --- |
| def post\_processor\_after\_initialization(self):  EventMesh.subscribe("mqtt\_connect", self.start\_mqtt\_connect) |

1. Initiate a connection request.

|  |
| --- |
| def connect(self, topic=None, data=None):  if not self.conn\_flag:  self.conn\_flag = True  self.\_\_mqtt\_client = aLiYun(self.product\_key,self.product\_secret,self.device\_name,self.device\_secret)  con\_state = self.\_\_mqtt\_client.setMqtt(self.client\_id,clean\_session=self.clean\_session,reconn=True)  self.log.info("connect con\_state --{}".format(con\_state))  if con\_state != 0:  self.log.warn("mqtt connect failed!")  EventMesh.publish("tts\_play", TTS\_CONTENT.SERVER\_CONN\_FAIL)  return False  self.\_\_mqtt\_client.start()  self.\_\_mqtt\_client.setCallback(self.callback)  sub\_sta = self.\_\_mqtt\_client.subscribe(self.sub\_topic, qos=self.qos)  if sub\_sta != 0:  self.log.warn("mqtt subscribe topic failed!")  return False  EventMesh.publish("tts\_play", TTS\_CONTENT.SERVER\_CONN\_SUCCESS)  self.log.info("mqtt connect success!")  return True |

1. The downlink data callback function.

|  |
| --- |
| def callback(self, topic, msg):  '''  mqtt message callback  '''  json\_data = ujson.loads(msg)  params = json\_data.get('params', False)  self.log.info("json\_data {}".format(json\_data))  EventMesh.publish("put\_msg\_queue", params) |

* 1. GLight
* **Function description**

Manage device LED light status. Control the turn-on, turn-off and flash of LED light by enabling GPIO.

* **Principle**

1. Create a class method to initialize class properties and LED GPIO pin through *\_\_init\_\_()*.

|  |
| --- |
| class RLight(Light):  def \_\_init\_\_(self):  super(RLight, self).\_\_init\_\_(Pin(Pin.GPIO13, Pin.OUT, Pin.PULL\_DISABLE, 0)) |

1. After the class method is initialized, the method that can be called by other functions are registered in EventMesh through event registration.

|  |
| --- |
| def post\_processor\_after\_initialization(self):  """ Subscribe to all such events in EventMesh"""  EventMesh.subscribe("red\_on", self.on)  EventMesh.subscribe("red\_off", self.off)  EventMesh.subscribe("red\_blink", self.blink\_O)  EventMesh.subscribe("red\_read", self.read)  EventMesh.subscribe("red\_set\_timeout", self.set\_timeout) |

1. Enable the GPIO pin to control the LED light

|  |
| --- |
| def on(self, topic=None, data=None):  self.l.write(1)  def off(self, topic=None, data=None):  self.l.write(0) |

* 1. DeviceActionManager
* **Function description**

Control device status, including on/off, restart and standby state of the device, and control peripherals.

* **Principle**

1. After the class method is initialized, the method that can be called by other functions are registered in EventMesh through event registration.

|  |
| --- |
| class DeviceActionManager(Abstract):  """  Device behavior  """  def \_\_init\_\_(self):  self.\_\_led\_flag = 1  self.\_\_await\_start\_time = 0  self.\_\_lock = Lock()  self.log = get\_logger(\_\_name\_\_ + "." + self.\_\_class\_\_.\_\_name\_\_)  def post\_processor\_after\_initialization(self):  # Register events  EventMesh.subscribe("device\_start", self.device\_start)  EventMesh.subscribe("device\_shutdown", self.device\_shutdown)  EventMesh.subscribe("device\_restart", self.device\_restart)  EventMesh.subscribe("update\_wait\_time", self.update\_device\_standby\_wait\_time)  EventMesh.subscribe("update\_led\_flag", self.update\_led\_flag)  # LED light flashed  \_thread.start\_new\_thread(self.blink\_thread, ())  # Flash on standby  \_thread.start\_new\_thread(self.device\_standby, ()) |

1. The device is in standby state.

|  |
| --- |
| def device\_standby (self):  # Standby  while True:  # self.log.info("device\_standby led\_flag {}".format(self.\_\_led\_flag))  if self.\_\_led\_flag == 1:  if utime.time() - self.\_\_await\_start\_time > 30:  EventMesh.publish("blue\_blink")  utime.sleep(8)  continue  utime.sleep(10)  continue |

1. Device LED light control task.

|  |
| --- |
| def blink\_thread(self):  # Flash thread  while True:  light\_flag = EventMesh.publish("update\_led\_flag")  # self.log.info("blink\_thread -- light\_flag {}".format(light\_flag))  if light\_flag < 4 and light\_flag != 1:  if light\_flag == 3:  # Low battery voice prompt  EventMesh.publish("audio\_play", AUDIO\_FILE\_NAME.BATT\_LOW)  if light\_flag == 1:  EventMesh.publish("blue\_blink")  elif light\_flag == 2:  EventMesh.publish("green\_blink")  elif light\_flag == 3:  EventMesh.publish("red\_blink")  utime.sleep(8)  else:  utime.sleep(8) |

1. Device powers on and powers off.

|  |
| --- |
| def device\_shutdown(self, topic=None, data=None):  # Device powers off.  EventMesh.publish("audio\_play", AUDIO\_FILE\_NAME.DEVICE\_SHUTDOWN)  utime.sleep(5)  Power.powerDown()  def device\_start(self, topic=None, data=None):  # Device powers on.  EventMesh.publish("audio\_play", AUDIO\_FILE\_NAME.DEVICE\_START)  def device\_restart(self, topic=None, data=None):  # Device reboots.  Power.powerRestart() |

1. System Initialization Process



Figure 5: Cloudspeaker System Initialization Flowchart

System initialization process description:

1. Initialize all function classes of Cloudspeaker.
2. All class methods have event registration or function processing to be completed before or after the initialization and instantiation. Therefore, there must be an application class that will complete the actions before and after initialization of each class method when loading and starting all class methods.
3. Business Process



Figure 6: Business Flow Chart

1. Example

This chapter describes how to conduct secondary development based on the project code, such as adding functions and modifying the functional logic of the code. This chapter provides an example to familiarize you with the code flow.

Before you are ready to adjust your code, you need to understand how the functionality in the current code runs. First of all, the functions implemented in the entire project are event-driven through EventMesh, that is, after implementing a function, define an event name for the function, such as query the device IMEI, and map the event name with the corresponding event function in the form of key-value pairs (such as "Get\_Imei": Function). This action of creating a mapping relationship is called event subscription. EventMesh saves the event, and when you need to query the device IMEI, only need to publish Get\_Imei event to get the returned results. Therefore, each function can be developed and debugged independently to achieve the effect of decoupling.

The steps of secondary development are as follows:

1. Add a function class and the function must be implemented first, as shown below:

|  |
| --- |
| class TestAddFunction(Abstract):  """  Add a function  """  def \_\_init\_\_(self):  pass  def post\_processor\_after\_initialization(self):  # Register an event  EventMesh.subscribe("test\_function", self.test\_function)  def test\_function(self, event, msg):  """Realize the function"""  print("TestAddFunction --test\_function()") |

As shown above, first, create a function class method that inherits a common base class, Abstract, which is used to restrict developers to maintain consistency in the implementation of the function. And then re-implement *post\_processor\_after\_instantiation()* in the class method. As the new function needs to be registered and started in the application class, and the aplication class calls the *post\_processor\_after\_instantiation()* in each of the function classes inherited from the Abstract class, the calling method should be subscribed to EventMesh in this class method.

Abstract base class:

|  |
| --- |
| class Abstract(object):  def post\_processor\_after\_instantiation(self, \*args, \*\*kwargs):  """Called after instantiation"""  pass  def post\_processor\_before\_initialization(self, \*args, \*\*kwargs):  """Called before initialization"""  pass  def initialization(self, \*args, \*\*kwargs):  """Initialize load"""  pass  def post\_processor\_after\_initialization(self, \*args, \*\*kwargs):  """Called after initialization """  pass |

Application class:

|  |
| --- |
| class App(object):  def \_\_init\_\_(self):  self.managers = []  self.log = get\_logger(\_\_name\_\_ + "." + self.\_\_class\_\_.\_\_name\_\_)  def append\_manager(self, manager):  if isinstance(manager, Abstract):  manager.post\_processor\_after\_instantiation()  self.managers.append(manager)  return self  def start(self):  for manager in self.managers:  manager.post\_processor\_before\_initialization()  manager.initialization()  manager.post\_processor\_after\_initialization() |

1. Add the new class method to the *main.py* entry function.
2. Import the new class method. *usr.msg* represents the *mgr.py* file under the usr (user) partition, where the new class method is written.

|  |
| --- |
| from usr.audio\_control import AudioManager  from usr.cloud import AliYunManage  from usr.common import ConfigStoreManager, Abstract  from usr.led\_control import RLight, GLight, BLight  from usr.ota\_control import OTAManager  from usr.mgr import FactoryManager, DeviceInfoManager, UartManager, KeypadManager,\  DeviceActionManager,OrderManager,ChargeManager,LteNetManager,CloudHornManager  from usr.app\_ota import TestAddFunction |

1. Add the class method to the application class.

|  |
| --- |
| app = App()  # application registration  app.append\_manager(ConfigStoreManager())  app.append\_manager(AudioManager())  app.append\_manager(RLight())  app.append\_manager(GLight())  app.append\_manager(BLight())  app.append\_manager(OTAManager())  app.append\_manager(KeypadManager())  app.append\_manager(DeviceInfoManager())  app.append\_manager(UartManager())  app.append\_manager(DeviceActionManager())  app.append\_manager(OrderManager())  app.append\_manager(ChargeManager())  app.append\_manager(AliYunManage())  app.append\_manager(LteNetManager())  app.append\_manager(CloudHornManager())  app.append\_manager(FactoryManager())  # Start  app.start() |

1. Use the new class method.

|  |
| --- |
| class PlayTestFuntion():  """  Use the new class method  """  def execute\_test\_function(self):  # Find the subscribed events by event publication and execute the corresponding event functions.  EventMesh.publish("test\_function") |

Find the subscribed events by event publication and execute the corresponding event functions.

1. Appendix Terms and Abbreviations

Table 2: Terms and Abbreviations

|  |  |
| --- | --- |
| **Abbreviation** | **Description** |
| API | Application Programming Interface |
| APN | Access Point Name |
| APP | Application |
| GPIO | General Purpose Input/Output Port |
| ICCID | Integrate Circuit Card Identity |
| IMEI | International Mobile Equipment Identity |
| IoT | Internet of Things |
| LED | Light Emitting Diode |
| MQTT | Message Queuing Telemetry Transport |
| OTA | Over-the-air programming |
| PA | Power Amplifier |
| SIM | Subscriber Identity Module |
| TCP | Transmission Control Protocol |
| TTS | Text To Speech |
| UART | Universal Asynchronous Receiver/Transmitter |
| UML | Unified Modeling Language |
| USB | Universal Serial Bus |