How to implement a conversational chat

Understanding the Pepper Chat Flow

Within the field of AI, how to maturely apply intelligent voice chat to robot products is a topic that must be addressed. Intelligent robot chat can not only answer the user's task-based and decision-making questions to help users in real time, but also maximise the friendly interaction between robots and people in casual chat questions.

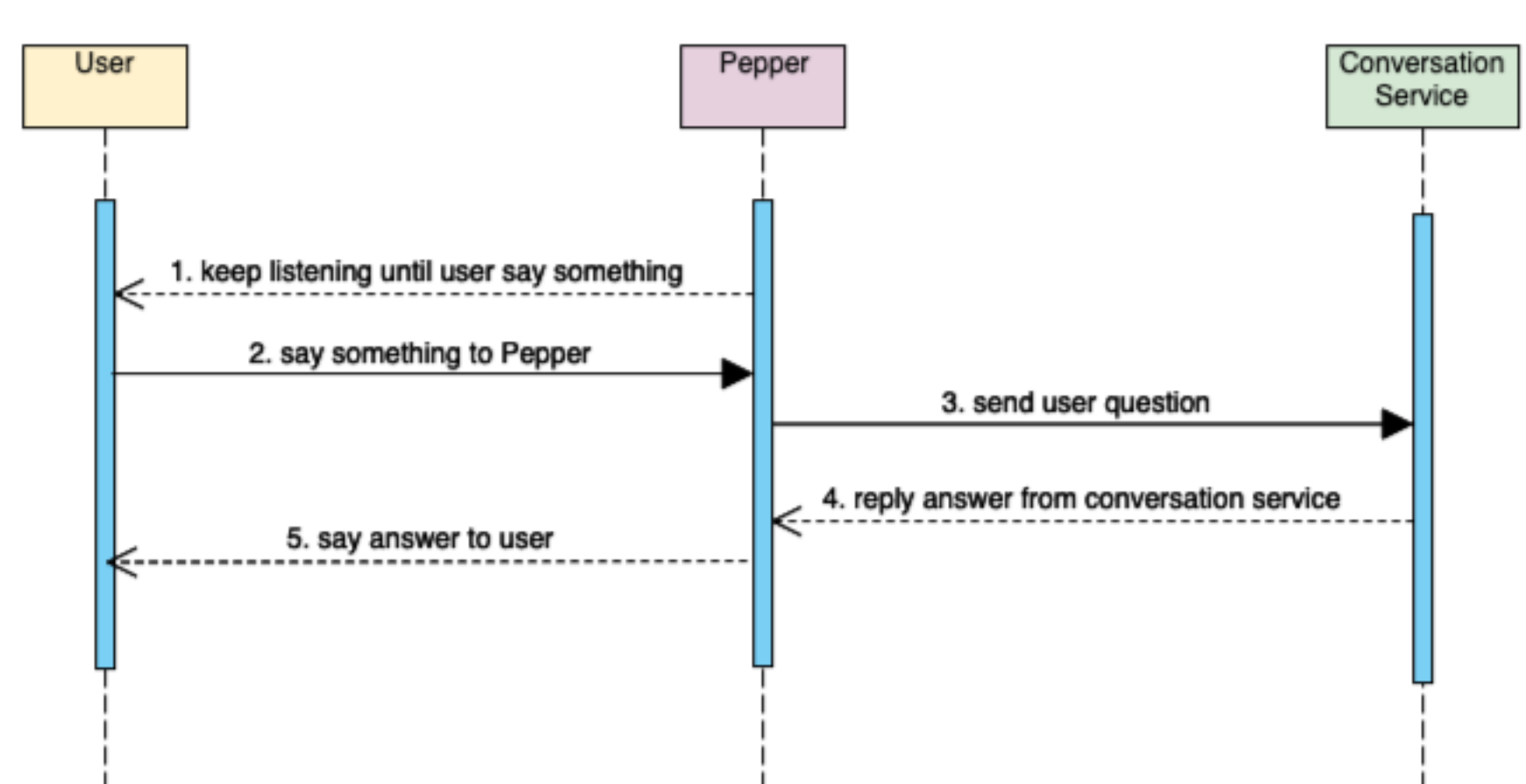
Today, we will explain and introduce Pepper's chat interaction process, and how to achieve human-robot dialogue through code.

The intelligent chatting flow between Pepper and the user is briefly summarised as follows:

Pepper listens to the user's voice input through the microphone on top of its head, and then converts the input voice recognition into text.
Pepper will text with the local or remote dialogue service for query, and then get the text of the reply.
Pepper converts the reply text into reply voice and then broadcasts it to the user through the ear speaker, thus realising a round of chatting interaction.

In summary, we can find that the Pepper chat function is actually composed of three parts: Listen, Think and Say.

The interaction flow can be referred to the following figure.



Description:

In the Pepper Listen session above, Automatic Speech Recognition (ASR) is used to receive the user's voice and convert it into text.

In Pepper Think, Pepper receives the user's input text and returns the user's desired text through an internal algorithm that understands natural language, a process known as semantic understanding.

Pepper Say session is Pepper received the result of the text, the text will be converted to speech, here the use of text-to-speech technology (Text To Speech, referred to as TTS), and then finally the voice results broadcast to the user.

Among them, the current Pepper built-in Nuance's ASR (support access to third-party service providers such as Xunfei), TTS.

Use QiSDK to implement chat function

Currently there are two ways to realise the chat requirements of the following corresponding scenarios.

Scenario one:

Users interact with Pepper in a simple fixed dialogue in a network-less or weak network environment.

Example scenario:

The user can have a simple chat with Pepper, which mainly includes greeting, asking Pepper the current time, and so on.

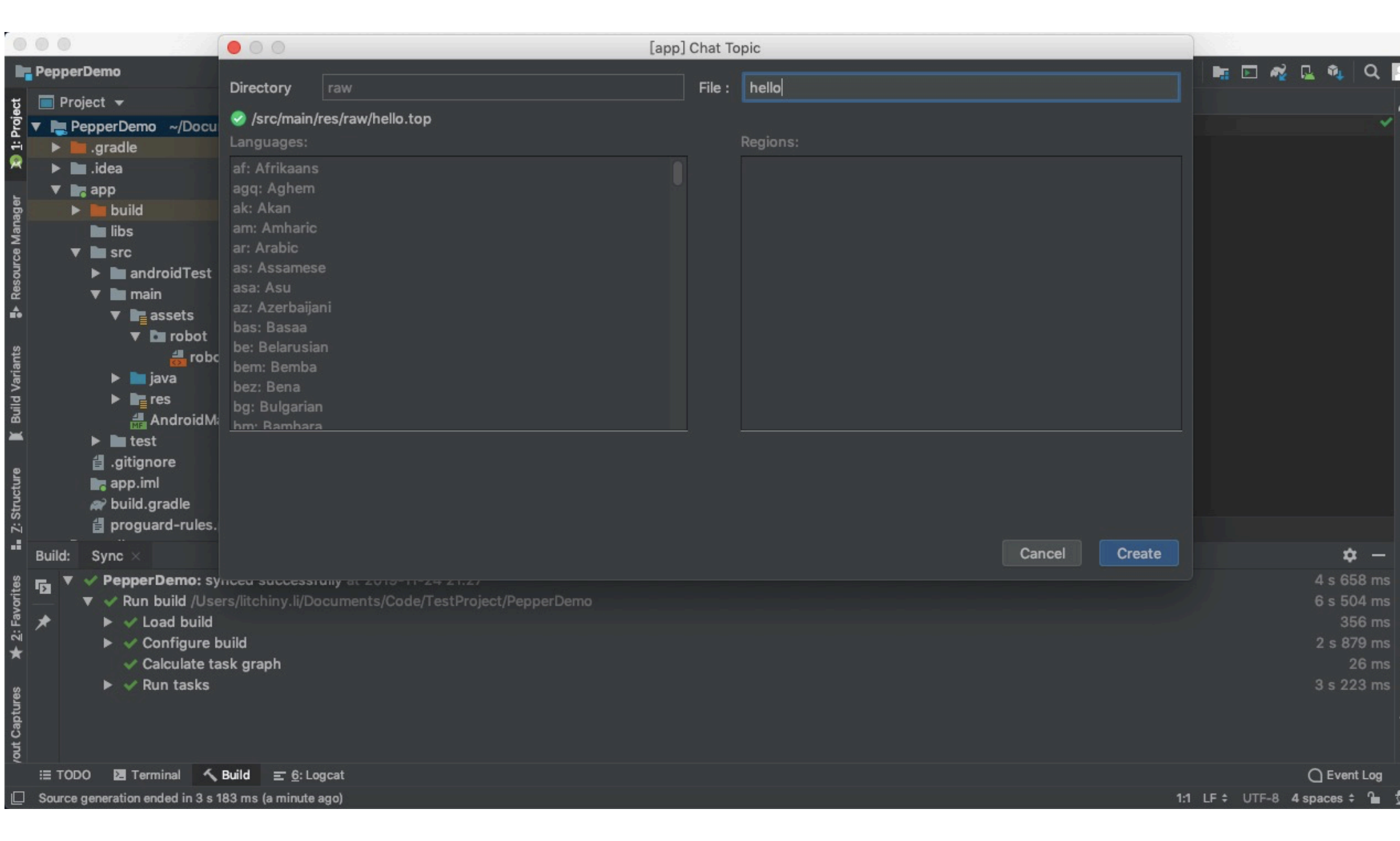
Implementation process:

Step 1: Create Robot application.

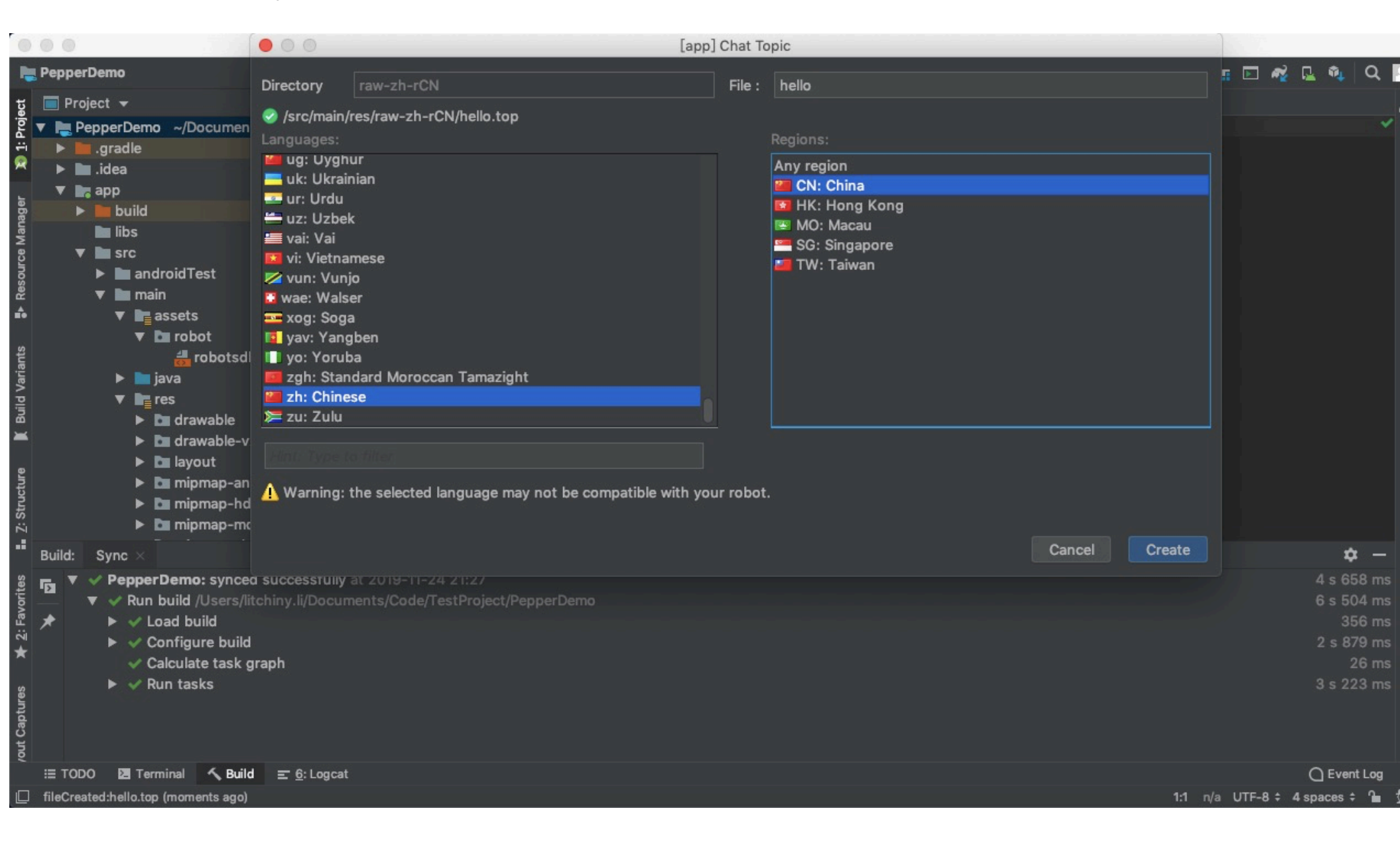
The project name is Pepper Demo.

Step 2: Create Topic file

Right click and select the project name PepperDemo, find New -> Chat Topic, create hello.top (default English version for the first time), the file will be added to your /res/raw/ folder.



Since the current case uses Chinese for chatting, repeat step 1 and select Chinese in the Language field to create hello.top, which will be added to your /res/raw-zh-rCN/ folder.



Find /res/raw-zh-rCN/hello.top and fill in the following.

`` topic: ~hello() concept:(hello) [hello Hi] concept:(age_question) [How old are you? How old are you?] concept:(age_answer) [I'm 5 years old. Don't look at me, but I know a lot!] u:(~hello) ~hello u:(~age_question) ~age_answer concept:(time_question) [What time is it? What is the time now?] u:(~time_question) The current time is: $time_answer

```
**Syntax explanation:**

The u symbol represents a user rule in the following format:

u:(human input) robot answer

When Pepper hears (human input), he says robot answer.

The concept represents a set of phrases assigned to a variable, invoked in a user rule using the ~
```

Step 3: Create a Topic instance

First use the TopicBuilder class to build a Topic and then pass in the top resource created in step two.

Place the following code into the onRobotFocusGained method:

```
Topic topic = TopicBuilder.with(qiContext)//use QiContext to create Builder
.withResource(R.raw.hello) // set topic resource
.build();// build topic
```

Step 4: Create a QiChatbot instance

First build a QiChatbot using the QiChatbotBuilder class and then pass in the topic created in step three.

Put the following code into the onRobotFocusGained method:

```
java
QiChatbot qiChatbot = QiChatbotBuilder.with(qiContext)
.withTopic(topic)
.build();
```

Step 5: Implement a simple time lookup service

The return value of the current time is obtained by calling the relevant API query of Calendar, the following is the sample code.

```
private String returnTimeAnswer(){
Calendar calendar = Calendar.getInstance();
int hour = calendar.get(Calendar.HOUR_OF_DAY);
int minute = calendar.get(Calendar.MINUTE); return hour + ‘point’; int minute = calendar.get(Calend
return hour + ‘point’ + minute + ‘minute’;
}
```

In the onRobotFocusGained method, qiChatbot sets the obtained time return value.

```
// Set the current time quiz reply within the top file
qiChatbot.variable(‘time_answer’).setValue(returnTimeAnswer());
```

Note: The setting of the Q&A text value in the hello.top file must take precedence over the creation of the Chat (please refer to the time setting in this case), otherwise it will not be possible to get a reply in the dialogue.

Step 6: Create Chat Instance

Now add Chat and Future global variables in your MainActivity.

```
private Chat chat;
private Future<Void> chatFuture.
```

First build Chat using the ChatBuilder class and then pass in the qiChatbot created in step four.

Put the following code into the onRobotFocusGained method:

```
// Create a Chat
chat = ChatBuilder.with(qiContext)
.withChatbot(qiChatbot)
.build();
```

Step 7: Run Chat and add listener

The class Chat has the addOnStartedListener method, which allows us to be notified when Chat starts, using log to print out the notification on the console.

```
// Set up the listener to start Chat
chat.addOnStartedListener(() -> Log.i(TAG, ‘Discussion started.’));
```

We can now run Chat in the onRobotFocusGained method and add error log tracking.

```
// Run Chat asynchronously
chatFuture = chat.async().run();
// If an error is encountered during the run of Chat an error message needs to be logged.
chatFuture.thenConsume(future -> {
if (future.hasError()) {
Log.e(TAG, ‘Discussion finished with error.’, future.getError());
}
});
```

Note:

You need to remove the Chat startup listener in the onRobotFocusLost method and request that Chat be cancelled.

```
if (chat != null) {
chat.removeAllOnStartedListeners();
}
if (chatFuture != null) {
chatFuture.requestCancellation();
}
```

Scenario two:

The user and Pepper complete a rich corpus chat interaction in a better network environment.

Target Scenario:

Users can chat and interact with Pepper with rich corpus, and the source of corpus is Server-side database.

Implementation Process:

Step 1: Create Robot application.

The project name is PepperBaseChat.

Step 2: Create your ChatbotReaction

The class ChatbotReaction can be interpreted as the Pepper's reaction to user voice input, e.g. it usually makes the Pepper talk (but it can also play animations, display content on a tablet, etc.).

In the app module of your project, create a new class called MyChatbotReaction inheriting from BaseChatbotReaction and implement the corresponding abstract class according to the example below:

```
public class MyChatbotReaction extends BaseChatbotReaction {
private String text;
private Future<Void> sayFuture;

protected MyChatbotReaction(QiContext qiContext, String text) {
super(qiContext);
this.text = text;
}

@Override
public void runWith(SpeechEngine speechEngine) {
Say say = SayBuilder.with(speechEngine).withText(text).build();
sayFuture = say.async().run();
}

@Override
public void stop() {
if (sayFuture != null) {
sayFuture.requestCancellation();
sayFuture = null;
}
}
}
```

Step 3: Create your Chatbot

Now that we have our custom MyChatbotReaction class, we need to create another Chatbot class that will

call the dialogue interface on the Server side.
bind the previously created MyChatbotReaction.

Create a MyChatBot class and provide the following code for it:

```
public class MyChatBot extends BaseChatbot {
private QiContext qiContext;

protected MyChatBot(QiContext qiContext) {
super(qiContext);
this.qiContext = qiContext;
}

@Override
public StandardReplyReaction replyTo(Phrase phrase, Locale locale) {
//Get the text of the user's question
String phraseText = phrase.getText();
//Get the dialogue reply content
String text = replyFromServer(phraseText);; //Create MyChatBotReaction.
//Create MyChatBotReaction
MyChatbotReaction myChatBotReaction = new MyChatbotReaction(qiContext, text); //create a new Standa
return new StandardReplyReaction(myChatBotReaction, text
TextUtils.isEmpty(phraseText) ? ReplyPriority.FALLBACK : ReplyPriority.NORMAL); }
}

private String replyFromServer(String text) {
// Call the Server side to get the reply content of the dialogue request, please implement it accor
...
Return ‘This is a greeting from Pepper’;
}
}
```

Step 4: Create Running Chat

Now add Future's global variable in your MainActivity.

```
private Future<Void> chatFuture; ``java
```

Create and run Chat in the onRobotFocusGained method in your MainActivity:

```
// Create MyChatBot
MyChatBot myChatBot = new MyChatBot(qiContext);
// Create Chat
Chat chat = ChatBuilder.with(qiContext)
.withChatbot(myChatBot)
// You need to call this method only if you are using a third-party ASR, see FAQ Q1 for a detailed
// .withAsrDriverParameters(myAsrParams)
.build();
// Run chat
chatFuture = chat.async().run();
```

Note:

Chat's permission to cancel needs to be requested in the onRobotFocusLost method.

```
if (chatFuture ! = null) {
chatFuture.requestCancellation();
}
```

Now you can run this application on Pepper and listen to Pepper broadcasting the dialogue replies obtained from the Server side while interacting with Pepper's voice.

Frequently Asked Questions

Q1：How to apply Xunfei Speech Recognition to Pepper?

A1：Please refer to: XFyun Speech Recognition.

Q2：How to modify the speed and tone of voice when Pepper speaks?

A2：Because Pepper achieves voice broadcasting through Say, you can add fixed keywords in the speaking text to achieve the purpose of modifying the speed and tone of speech.

The keywords are as follows:

tone speed content	Say effect	description

Sample code:

```
private void say(QiContext qiContext) {
if (qiContext == null) return;
String text = ‘I am now talking normally \\rspd=50\\\\my speech has slowed down.’ +
‘\\\vct=50\\\\\ My intonation has become lower.’ +
‘\\\\pau=1000\\\\\\my pauses in speech have become longer.’;
try {
Say say = SayBuilder.with(qiContext)
.withText(text)
.buildAsync()
.get();
Future sayFuture = say.async().run();
} catch (ExecutionException e) {
e.printStackTrace(); } catch (ExecutionException e) { e.printStackTrace(); }
}
}
```