

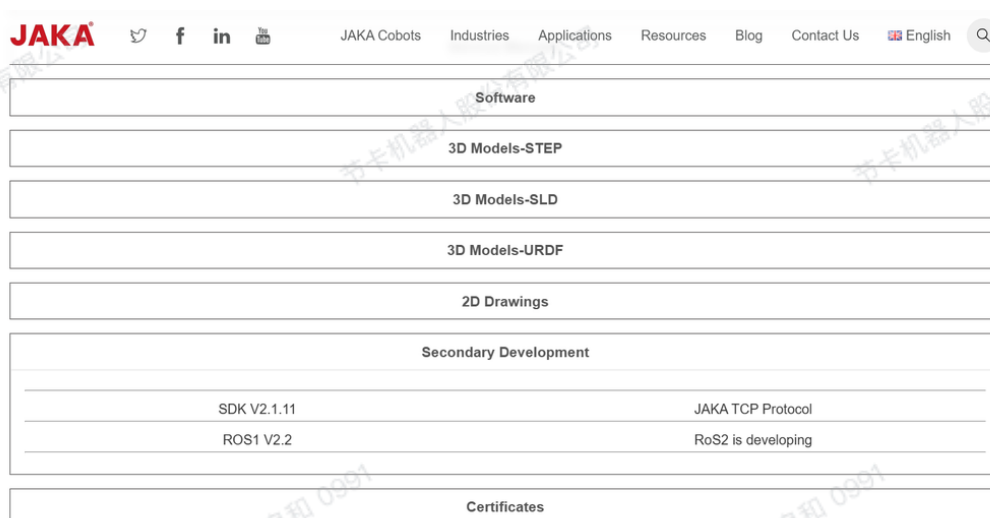
# JAKA SDK Quick Start - EN

## 1. Documentation

JAKA SDK is an efficient toolkit designed to help developers easily interact with JAKA robots. SDK provides a set of interfaces to support device connection, control, data transmission and other functions. Through this guide, you will be able to build your own application based on the SDK provided by JAKA in a short time and realize basic interaction with JAKA robots.

## 2. Get JAKA SDK

The JAKA SDK software package can be downloaded from the JAKA official website. The English version can be found on the English official website <https://www.jakarobotics.com/>, click [Resources] -> [Download] -> [Technical Information] -> [Secondary Development] to view the latest and historical versions.



## 3. Writing user programs

To help users quickly build their first application on different platforms, this article uses three examples to illustrate how to use different compilation tools to create applications on Windows and Linux platforms:

- 1) Create C# applications using Microsoft Visual Studio on Windows;
- 2) Use CMake to create C++ applications on Linux;

3) Use Qt Creator to create Qt applications on Linux.

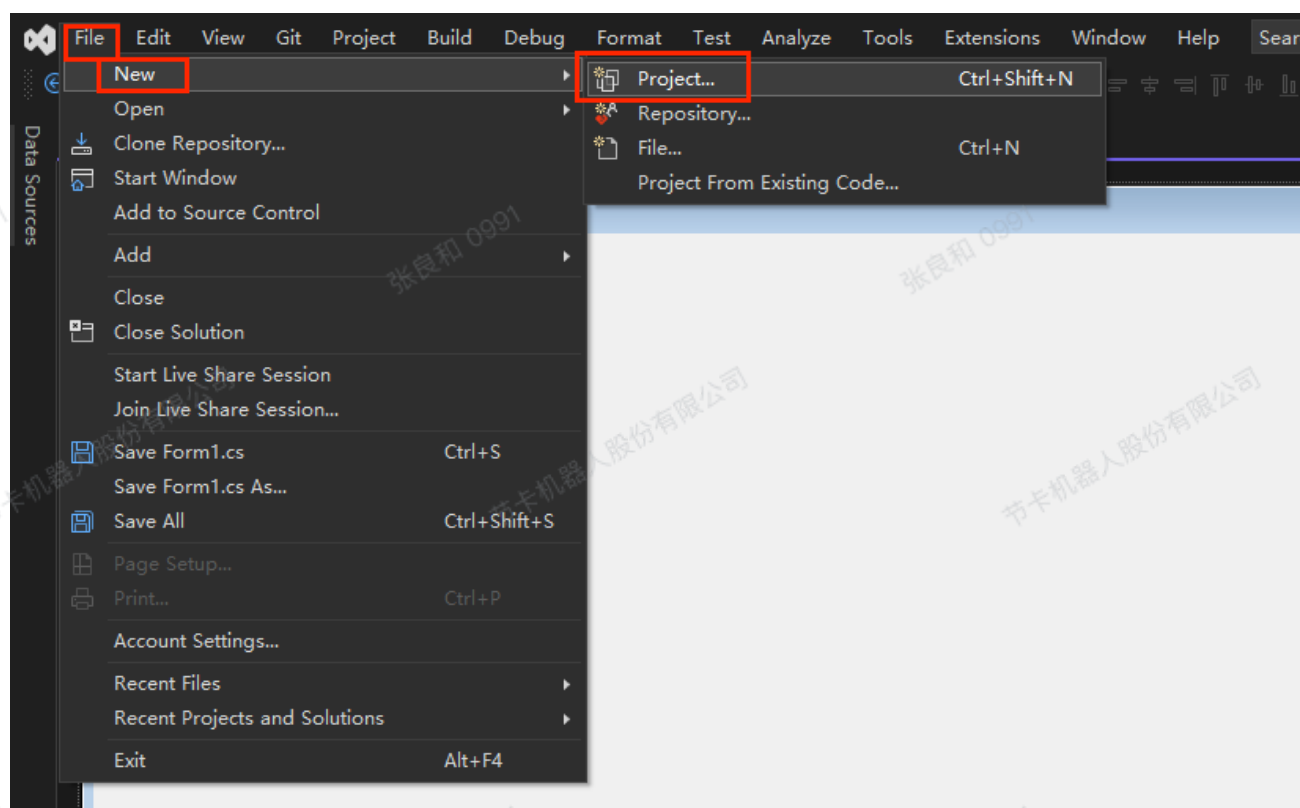
## 3.1 Creating C# Applications with Visual Studio on Windows

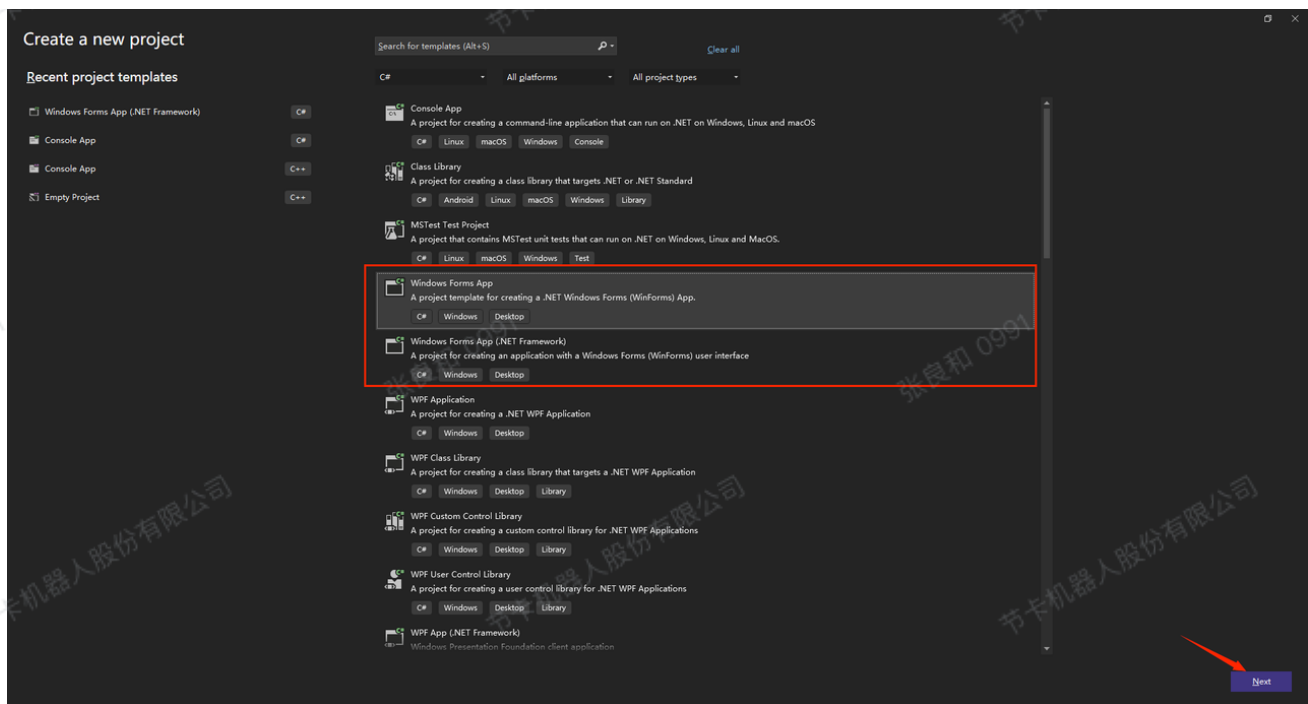
### 3.1.1 Compilation environment installation

Users need to download and install the Microsoft Visual Studio programming platform software. Please install it according to the online resources. The following description assumes that the reader has installed the software. During the deployment of the C# development environment, you may be required to update the .NET framework. Please deploy it according to the requirements (the .NET framework for this project is .NET6.0 when it runs normally).

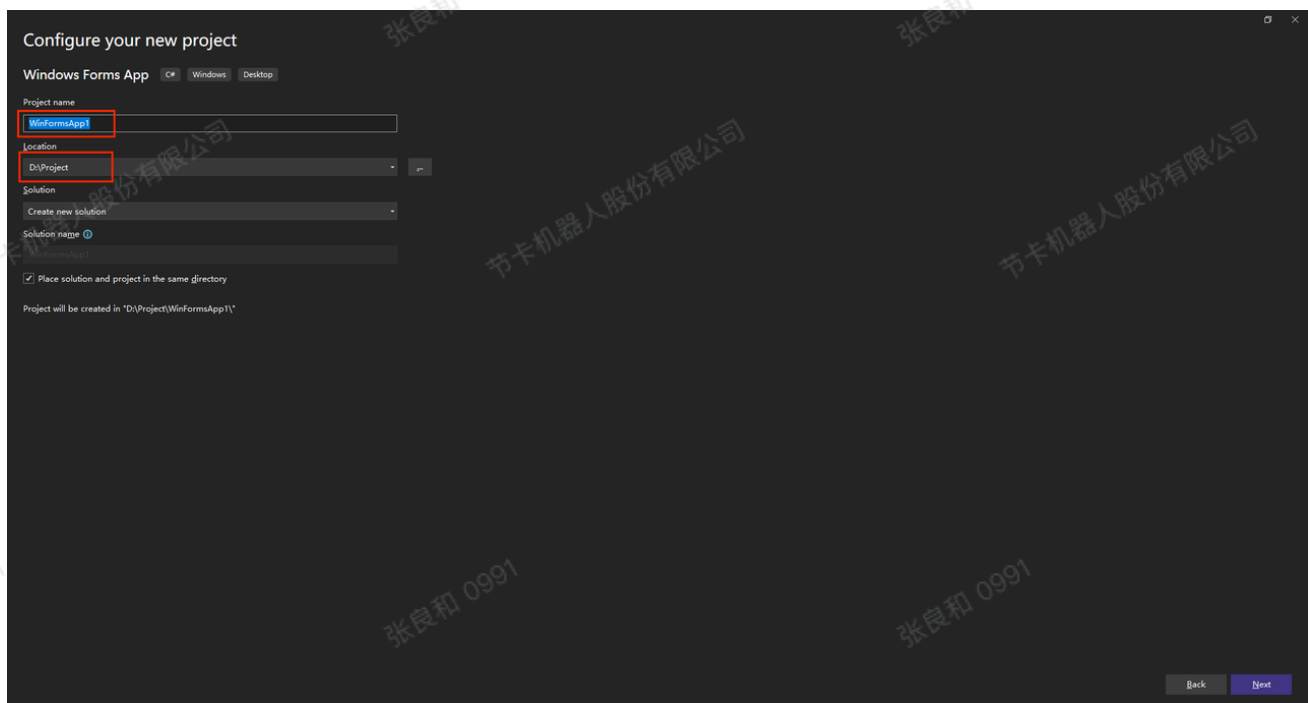
### 3.1.2 Create New Project

Open the software and click File->New->Project in the upper left corner, as shown in the figure below. In this example, we choose to create a form application.





Click Next. If you need a cross-platform application, it is recommended to select Windows Form App; if it only runs on Windows, you can select Windows Forms App (.Net Framework). After further setting the project name and storage location, click Next until the project is created.



### 3.1.3 Compile and link environment configuration

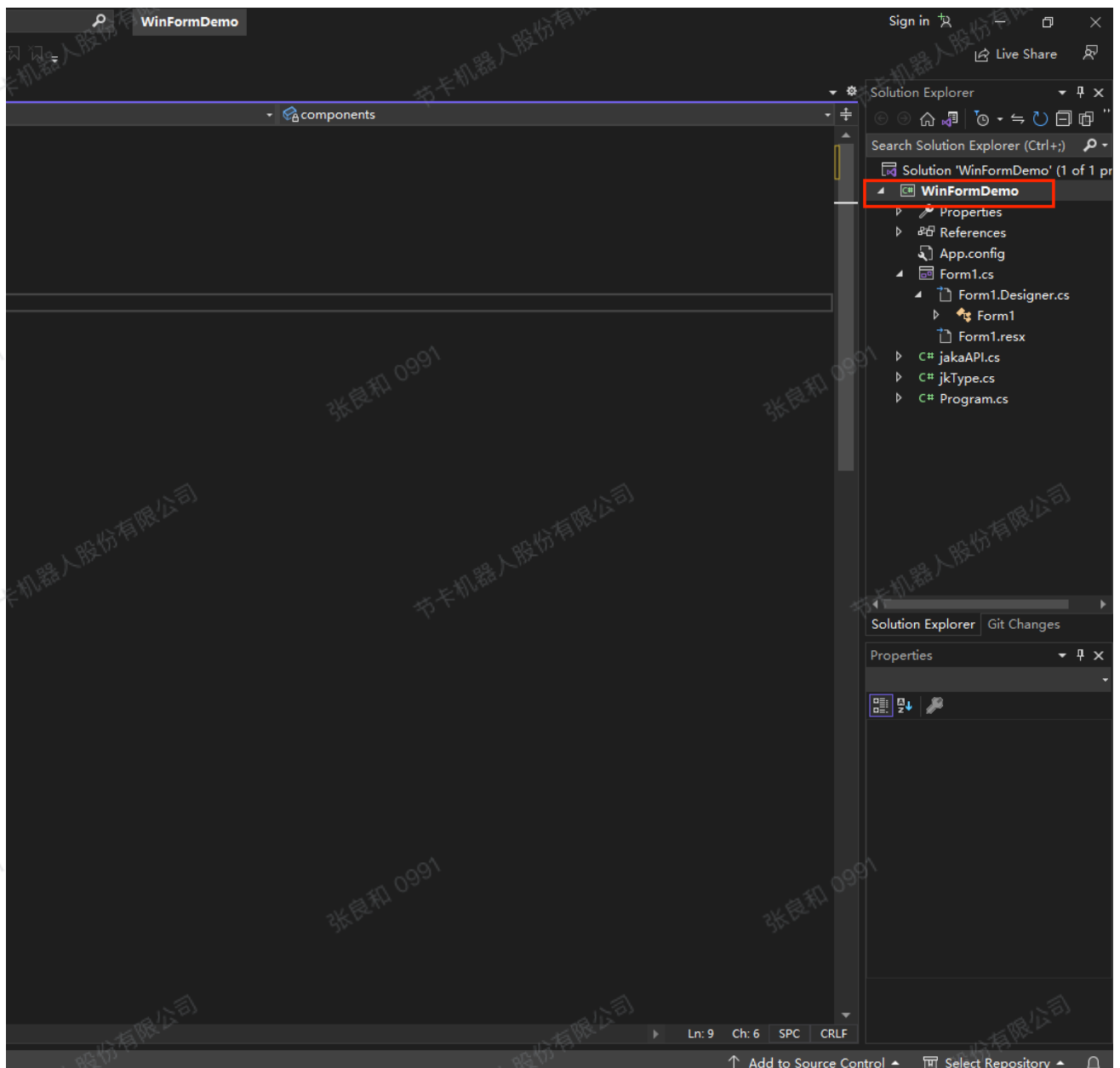
After decompressing the SDK software package downloaded from the JAKA official website, you can see the file directory as follows.

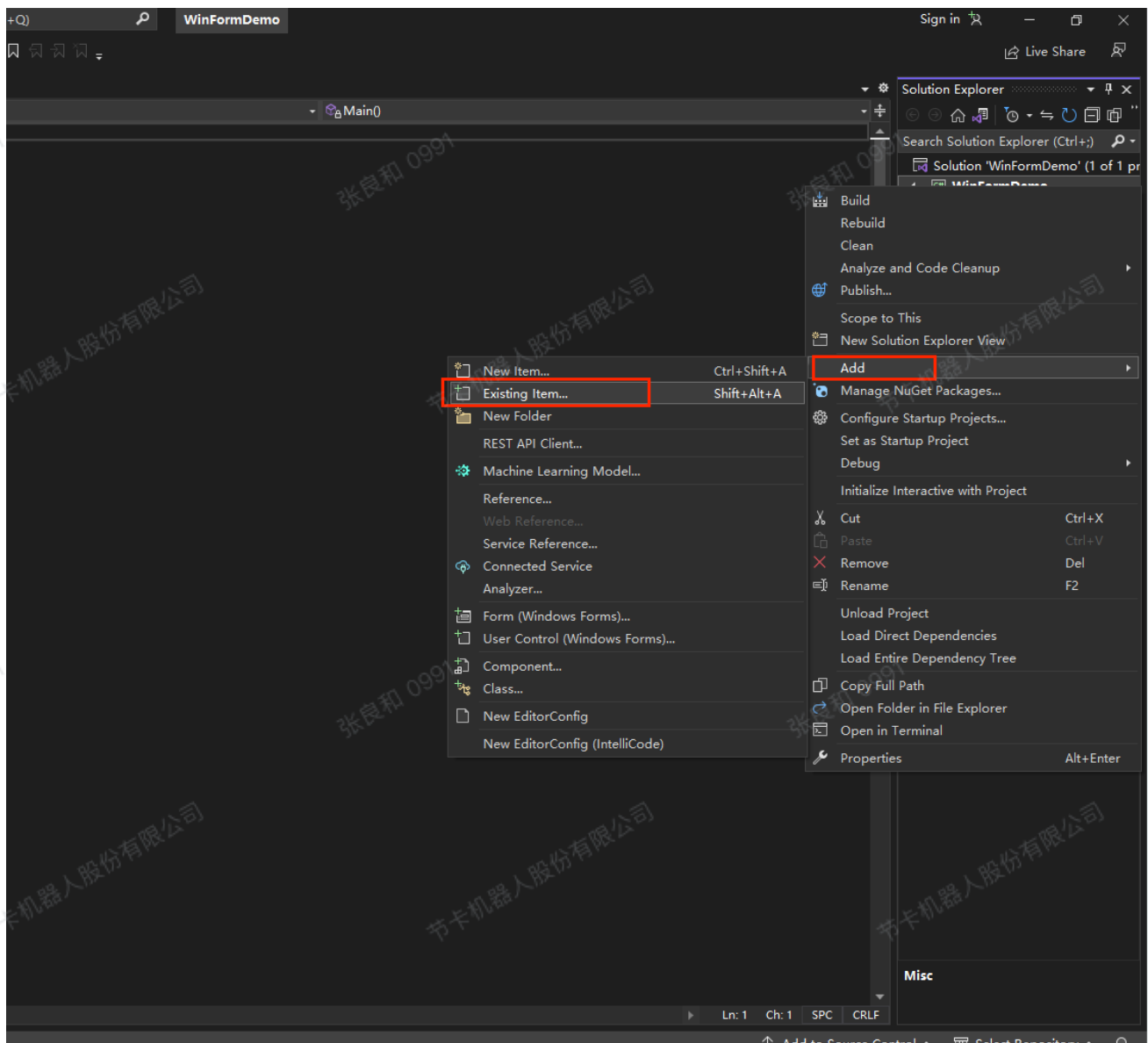
sdk-v2-1-11 >				在 sdk-v2-1-11 中搜索
名称	修改日期	类型	大小	
doc	2024/05/06 17:58	文件夹		
SDK2.1.11	2024/05/06 10:32	文件夹		
changelog	2024/05/06 10:13	文件	4 KB	
commonly used Microsoft runtime li...	2024/06/20 17:24	应用程序	79,250 KB	
readme_EN.txt	2024/05/06 18:08	文本文档	3 KB	

For Windows C# application development, you can find the include (containing header files) and x64 (containing library files under Windows 64-bit system) directories in the Windows\csharp directory.

sdk-v2-1-11 > SDK2.1.11 > Windows > csharp >				在 csharp 中搜索
名称	修改日期	类型	大小	
include	2024/05/06 10:31	文件夹		
x64	2024/05/06 10:31	文件夹		

Enter the newly created project, click Add--->Existing Item in the right sidebar, and add the two header files in the include folder: jakaAPI.cs and jkType.cs to the project.





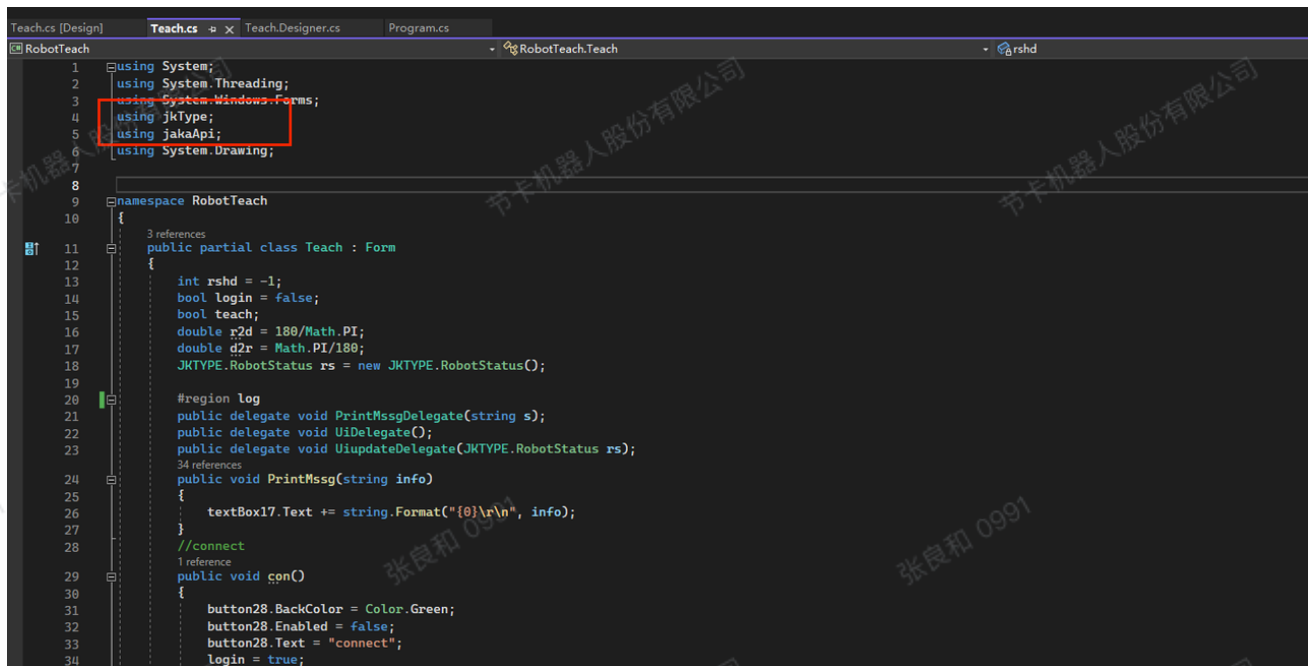
Put the C# shared library file in the SDK package into the main program directory of the project, usually in the bin directory.

名称	修改日期	类型	大小
WinFormDemo.exe	2024/11/6 16:17	应用程序	8 KB
WinFormDemo.exe.config	2024/11/6 14:23	CONFIG 文件	1 KB
WinFormDemo.pdb	2024/11/6 16:17	Program Debug...	30 KB
jakaAPI.dll	2024/10/12 10:43	应用程序扩展	1,142 KB
jakaAPI.exp	2024/10/12 10:43	Exports Library ...	61 KB
jakaAPI.lib	2024/10/12 10:43	Object File Library	101 KB

Add the following two lines of code in the user program so that the types and methods defined in the JAKA SDK can be used in subsequent codes.

```
using jkType;

using jakaApi;
```



```
1 using System;
2 using System.Threading;
3 using System.Windows.Forms;
4 using jkType;
5 using jkApi;
6 using System.Drawing;
7
8 namespace RobotTeach
9 {
10     3 references
11     public partial class Teach : Form
12     {
13         int rshd = -1;
14         bool login = false;
15         bool teach;
16         double r2d = 180/Math.PI;
17         double d2r = Math.PI/180;
18         JKTYPE.RobotStatus rs = new JKTYPE.RobotStatus();
19
20         #region log
21         public delegate void PrintMssgDelegate(string s);
22         public delegate void UiDelegate();
23         public delegate void UiupdateDelegate(JKTYPE.RobotStatus rs);
24         34 references
25         public void PrintMssg(string info)
26         {
27             textBox17.Text += string.Format("{0}\r\n", info);
28         }
29         //connect
30         1 reference
31         public void con()
32         {
33             button28.BackColor = Color.Green;
34             button28.Enabled = false;
35             button28.Text = "connect";
36             login = true;
37         }
38     }
39 }
```

### 3.1.4 Writing an application

After creating the program, you can develop the interface and corresponding functions according to your needs. After completing code editing, click [Build]->[Build Solution] to compile and generate an executable program. Finally, click the Start button on the toolbar to execute or debug the program.

The example provides a demonstration program for robot teaching, which includes the following functions: connection, power-on and enable, joint space motion control, Cartesian space posture control, etc. Customers can refer to the sample code for specific implementation.

## 3.2 Creating C++ Applications with CMake on Linux

### 3.2.1 Download and install CMake tools

It is recommended to download the installation package for the corresponding platform from the official website [Cmake official website]( <https://cmake.org/> ). This example uses Cmake version 3.7.2, and it is recommended to use this version or above. (For specific installation operations, please read the online resources, download and install them yourself) The following instructions assume that the reader has completed this operation.

After the download is complete, enter cmake --version in the terminal. If the following content is displayed, it means the installation is successful.

```
jakausen@ZuCAB2001:~$ cmake --version
cmake version 3.7.2

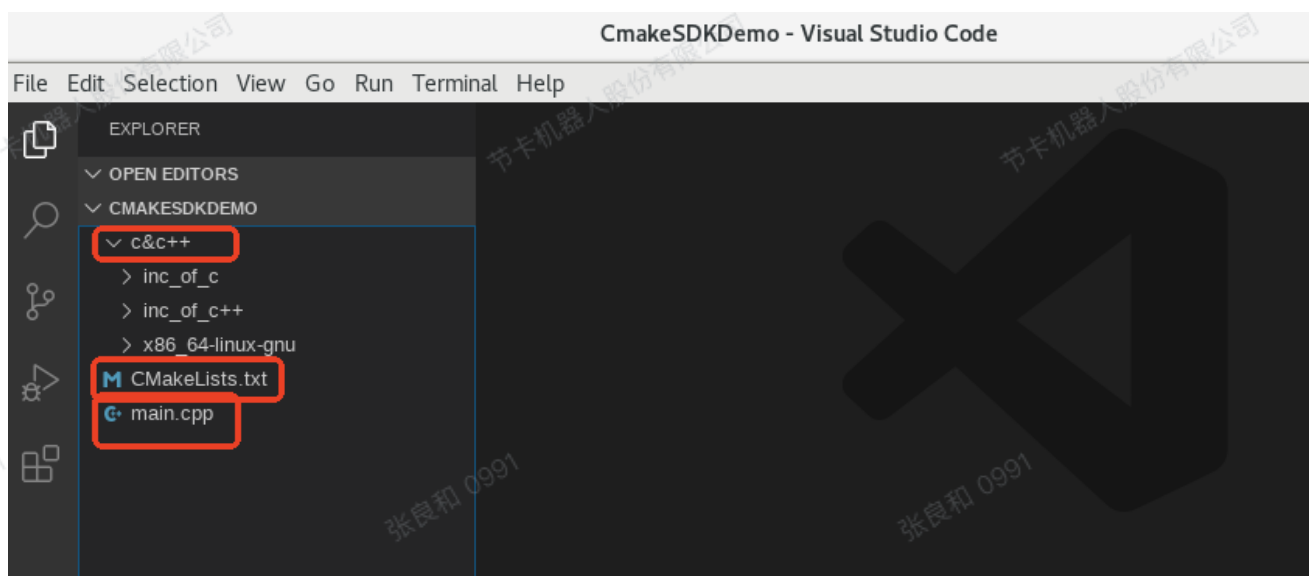
CMake suite maintained and supported by Kitware (kitware.com/cmake).
```

### 3.2.2 Download and install Visual Studio Code

Users can go to the Visual Studio Code official website ( <https://code.visualstudio.com/Download> ) to find the software installation package and download and install it. After VS Code is installed, please install the CMake extension in the Visual Studio Code extension.

### 3.2.3 Write the project framework and configure CMake

Create a new project folder CmakeSDKDemo under Linux and create the following files, where c&c++ are resource files under Linux in the SDK package provided by Jieka, including header files inc and dynamic library files. main.cpp is the main file of the project, and CMakeLists.txt is the Cmake configuration file. Please pay attention to the capitalization.



Next, use Cmake to write and run a simple hello world program. First, enter the following in CMakeLists.txt.

```
cmake_minimum_required(VERSION 3.5)
project(JAKADemo)

add_executable(hello main.cpp)
```



- `cmake_minimum_required`: Specify the minimum version requirement of CMake;
- `project`: Define the project name;
- `add_executable`: Add an executable target, the first parameter is the target name, the second parameter is the source file list.

Note: This project uses the gcc compiler by default, as shown below. Please configure the C++ compiler by yourself.

```
jakauser@ZuCAB2001:~/Desktop/sdk_test$ gcc --version
gcc (Debian 6.3.0-18+deb9u1) 6.3.0 20170516
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Write the main program `main.cpp` and enter the following code.

main.cpp

复制代码

```
#include <string>
#include <iostream>

int main(){
    std::cout << "hello world" << std::endl;
}
```

After that, we configure CMake. Enter `cmake .` in the terminal to configure the entire project. Then the configuration files required for building will be generated under the project file.

```
jakauser@ZuCAB2001:~/Desktop/CmakeSDKDemo$ cmake .
-- The C compiler identification is GNU 6.3.0
-- The CXX compiler identification is GNU 6.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/jakauser/Desktop/CmakeSDKDemo
jakauser@ZuCAB2001:~/Desktop/CmakeSDKDemo$ make
```

After the generation is complete, enter the make command in the terminal to build and compile the executable program. A hello executable program will be generated in the current directory.

```
jakauser@ZuCAB2001:~/Desktop/CmakeSDKDemo$ make
Scanning dependencies of target hello
[ 50%] Building CXX object CMakeFiles/hello.dir/main.cpp.o
[100%] Linking CXX executable hello
[100%] Built target hello
```

Enter the ./hello command in the terminal. Execute the program and the result is as follows.

```
jakauser@ZuCAB2001:~/Desktop/CmakeSDKDemo$ ./hello
hello world
```

At this point, running a simple helloworld program with CMake is complete.

### 3.2.4 Link JAKA SDK library

Enter the following content in the CMakeLists.txt file. Please refer to the comments for the function of each line of configuration.

▼ CMakeLists.txt

复制代码

```
cmake_minimum_required(VERSION 3.7.2)    # Minimum CMake Version
project(sdk_test VERSION 1.0)           # Project Definition

# C++ Standard
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED true)

# Source Directory Information
message(${CMAKE_SOURCE_DIR})

# Executable Creation
add_executable(demo main.cpp)

# Set the header file path of the SDK included in the project
target_include_directories(demo PUBLIC
${CMAKE_SOURCE_DIR}/c&c++/inc_of_c++)

# Set the SDK dynamic library link path
target_link_libraries(demo ${CMAKE_SOURCE_DIR}/c&c++/x86_64-linux-
gnu/shared/libjakaAPI.so pthread)
```

Write the main.cpp file and implement the control sample code through the SDK as follows: The functions implemented in this example are to print the SDK version information, obtain the current TCP position, and move the robot with a certain distance along the y-axis.

main.cpp

复制代码

```
#include <string>
#include <vector>
#include <iostream>
#include <chrono>
#include "JAKAZuRobot.h"
#include <thread>
int main(int argc, char** argv)
{
    JAKAZuRobot demo;

    demo.login_in("192.168.164.222");
    //sleep(2);
    demo.power_on();
    //sleep(2)
    demo.enable_robot();
    //sleep(2);

    CartesianPose tcp_pos;
    int ret;
    char ver[100];
    demo.get_tcp_position(&tcp_pos);
    demo.get_sdk_version(ver);

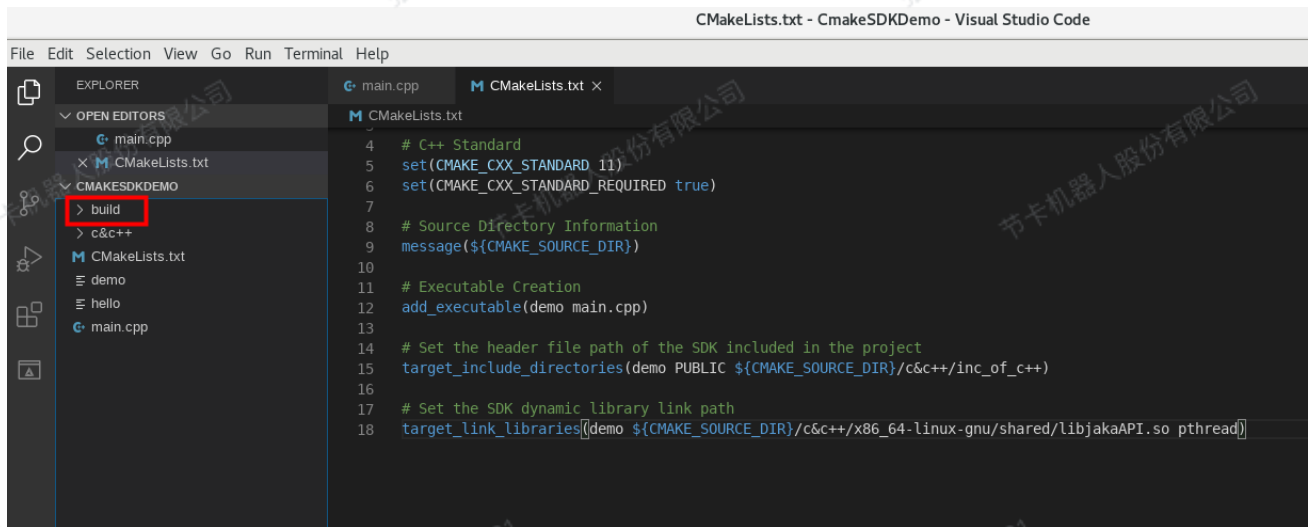
    std::cout << "SDK version is :" << ver << std::endl;
    std::cout << "tcp_pos is :\n x: " << tcp_pos.tran.x << "y: " <<
tcp_pos.tran.y << "z: " << tcp_pos.tran.z << std::endl;

    std::cout << "tcp_pos is :\n rx: " << tcp_pos.rpy.rx << "ry: " <<
tcp_pos.rpy.ry << "rz: " << tcp_pos.rpy.rz << std::endl;
    auto now = std::chrono::system_clock::now().time_since_epoch();
    auto ms = std::chrono::duration_cast<std::chrono::milliseconds>
(now).count();
    std::cout << "Current s: " << std::fixed << ((double)ms) / 1000.0f
<< std::endl;
    tcp_pos.tran.y = tcp_pos.tran.y + 60.0;
    ret = demo.linear_move(&tcp_pos, ABS, TRUE, 10, 10, 1, NULL);
    std::cout << "ret==" << ret << std::endl;
```

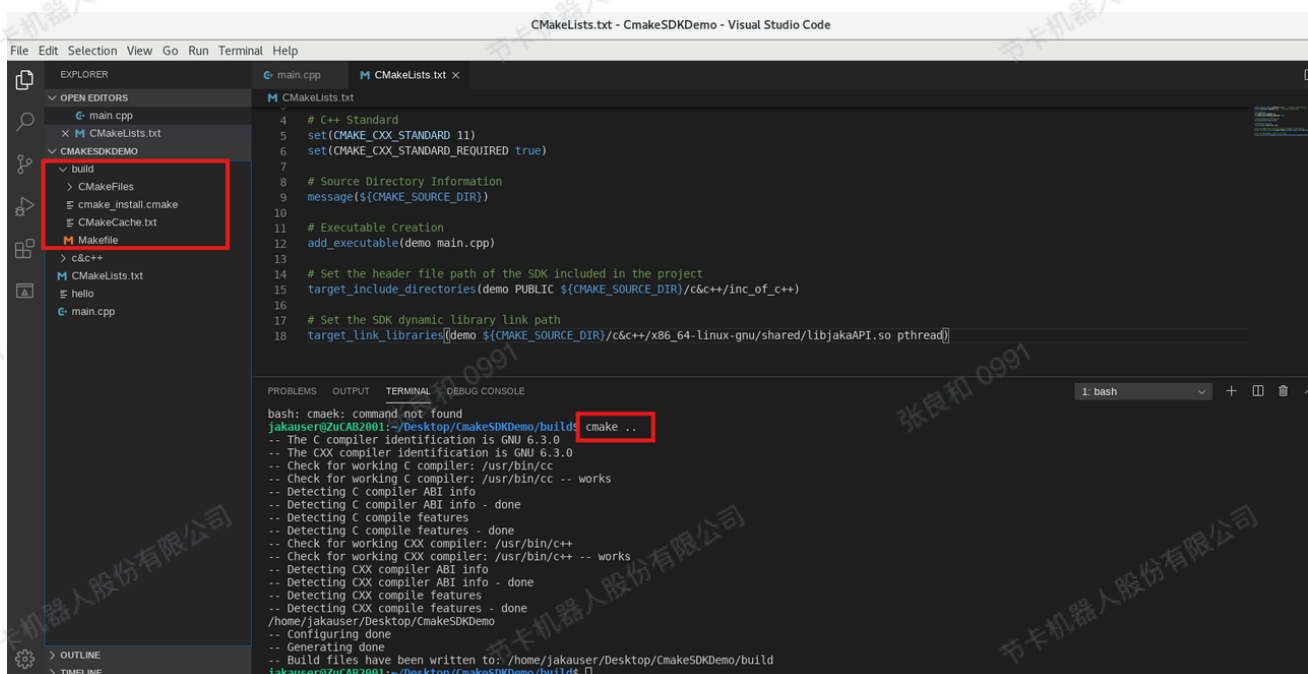
```
std::cout << "linear_move finish!" << std::endl;
now = std::chrono::system_clock::now().time_since_epoch();

return 0;
}
```

After all the above are prepared, configure and build the entire project through cmake. It is recommended to create a new build directory under the project root directory to store the intermediate files generated by the build to avoid complex file contents.



Then, enter the build directory with `cd build` and run the command `cmake ..` to build the entire project. The system will automatically put the generated intermediate files into the build directory. (Note that it is `cmake ..`, which represents the parent directory of the directory where CMakeLists.txt is located).



Then enter make in the terminal to build and compile the executable program.

```
jakauser@ZuCAB2001:~/Desktop/sdk_test$ make
Scanning dependencies of target demo
[ 50%] Building CXX object CMakeFiles/demo.dir/main.cpp.o
[100%] Linking CXX executable demo
[100%] Built target demo
```

After the above is completed, a demo executable program will be generated in the build directory. Readers can run the C++ program by running `./demo` in the terminal.

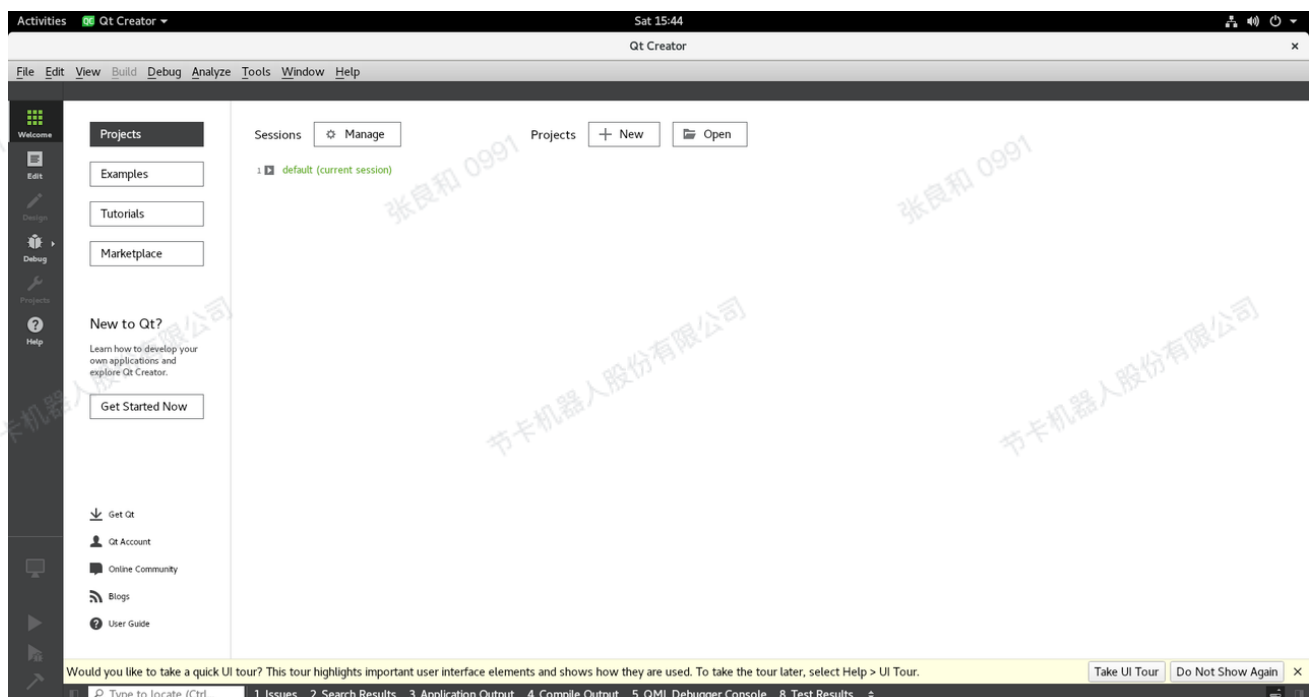
```
jakauser@ZuCAB2001:~/Desktop/sdk_test$ ./demo
try to connect: 172.30.3.172
connect success
SDK version is :libadd jakaAPI_version: V2.1.13stable_linux
tcp_pos is :
x: 697.529y: -294.055z: 1609.65
tcp_pos is :
rx: 3.07921ry: 0.143896rz: 2.59831
Current s: 1730968142.690000
ret==0
linear_move finish!
```

Through the above steps, the reader's C++ program is written and can be run successfully.

## 3.3 Creating Qt Applications with Qt Creator on Linux

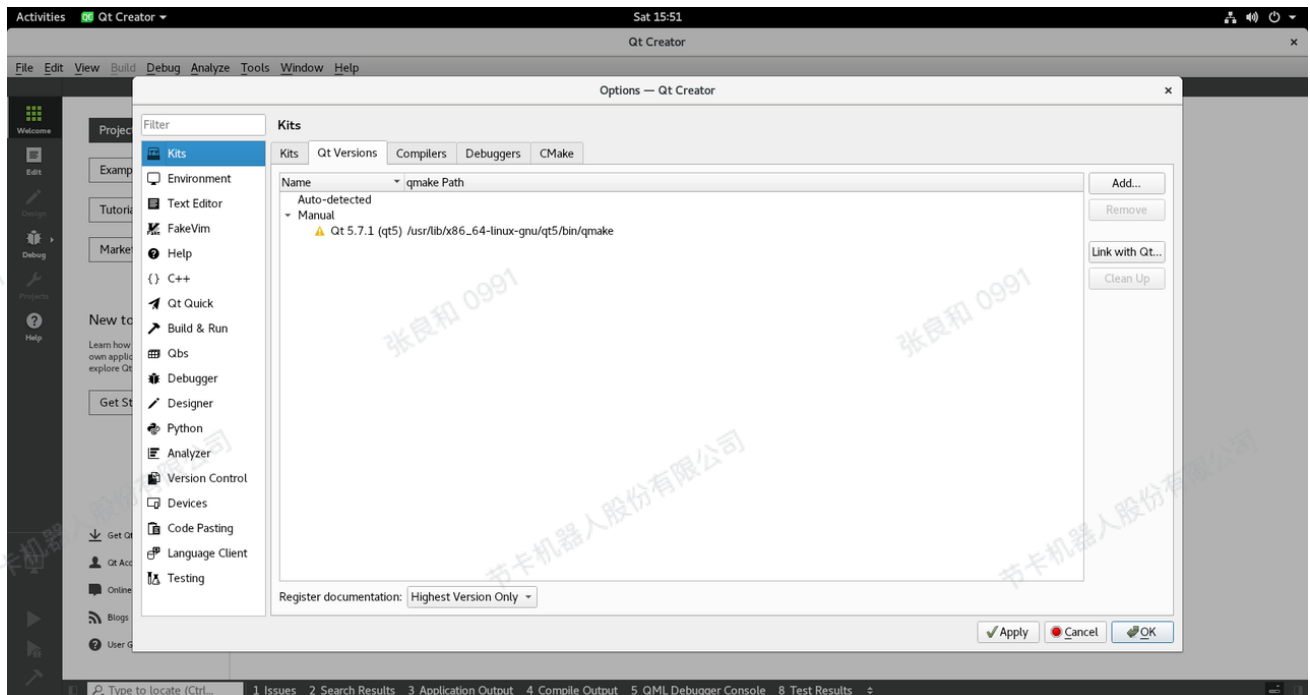
### 3.3.1 Download and install Qt Creator

Users can go to the Qt Group official website ( <https://www.qt.io/download-dev> ) to find the software installation package and download and install it. After Qt Creator is installed, start Qt Creator and the interface is as shown below.

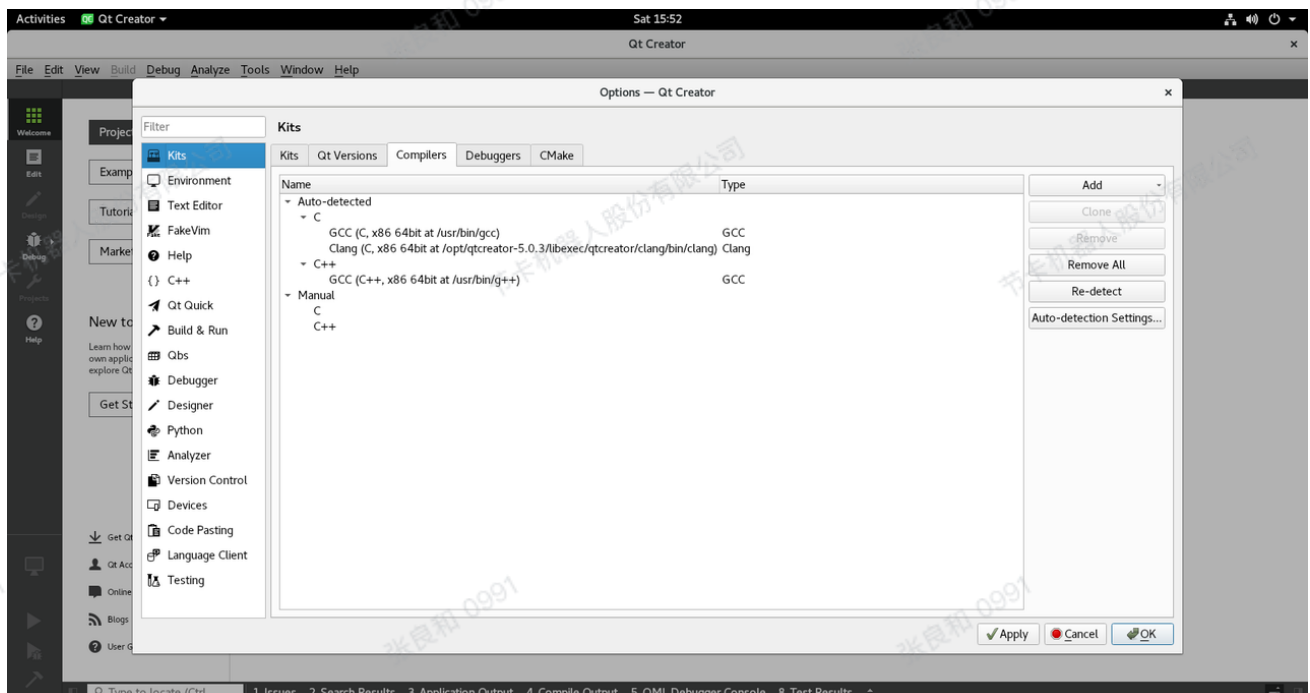


After Qt is installed, you need to further complete the Qt build configuration. Open Qt Creator, go to [Tools] -> [Options], and select the Kits tab in the pop-up dialog box.

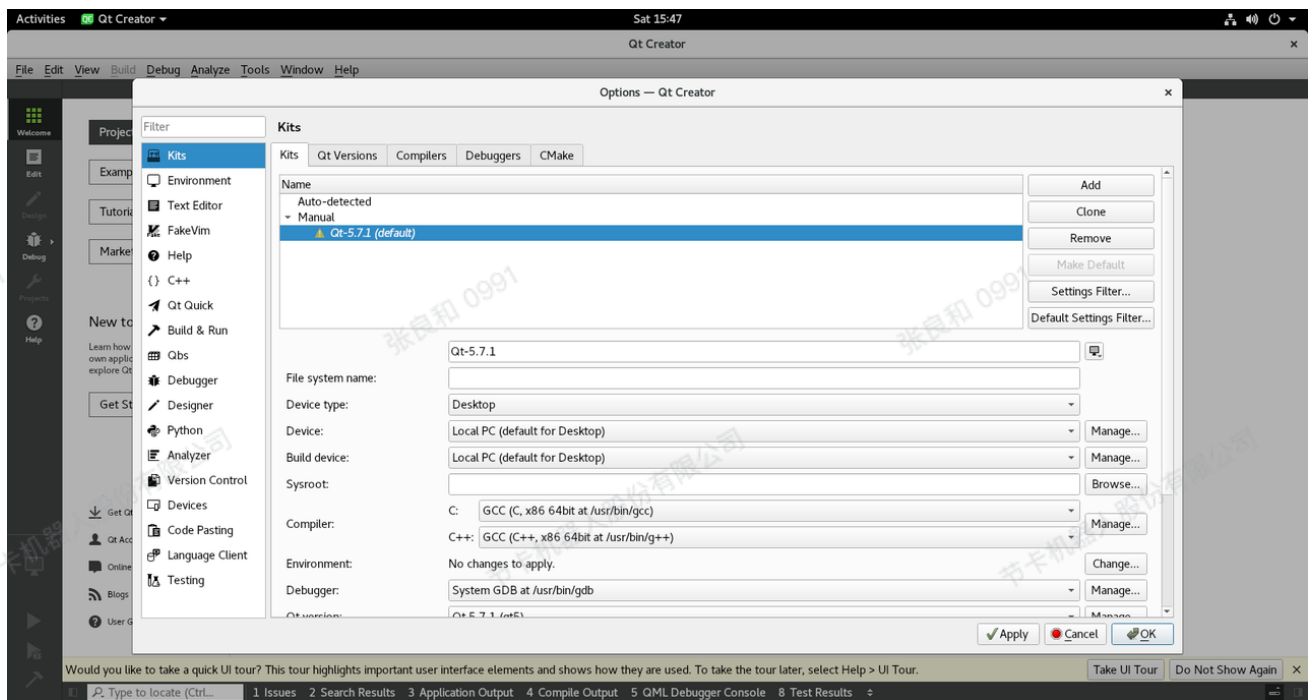
- In the Qt Versions section, make sure your installed Qt version is listed. If no version is shown, the user needs to add it manually.



- Confirm the compiler configuration in the Compilers section. Usually, Qt Creator will automatically recognize the installed compilers, but if not, the user can configure them manually.

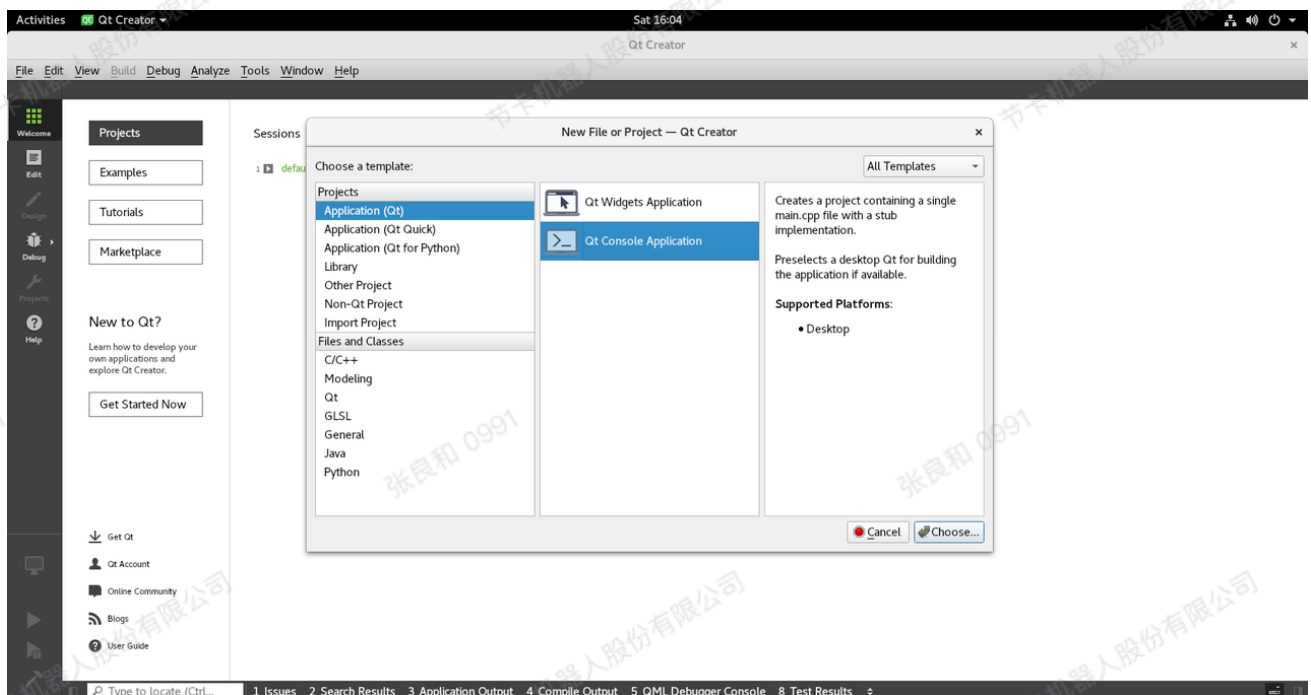


Once Qt and the compiler are configured, go to the Kits tab and click the Add button in the lower left corner to create a new Kit or select an existing Kit.



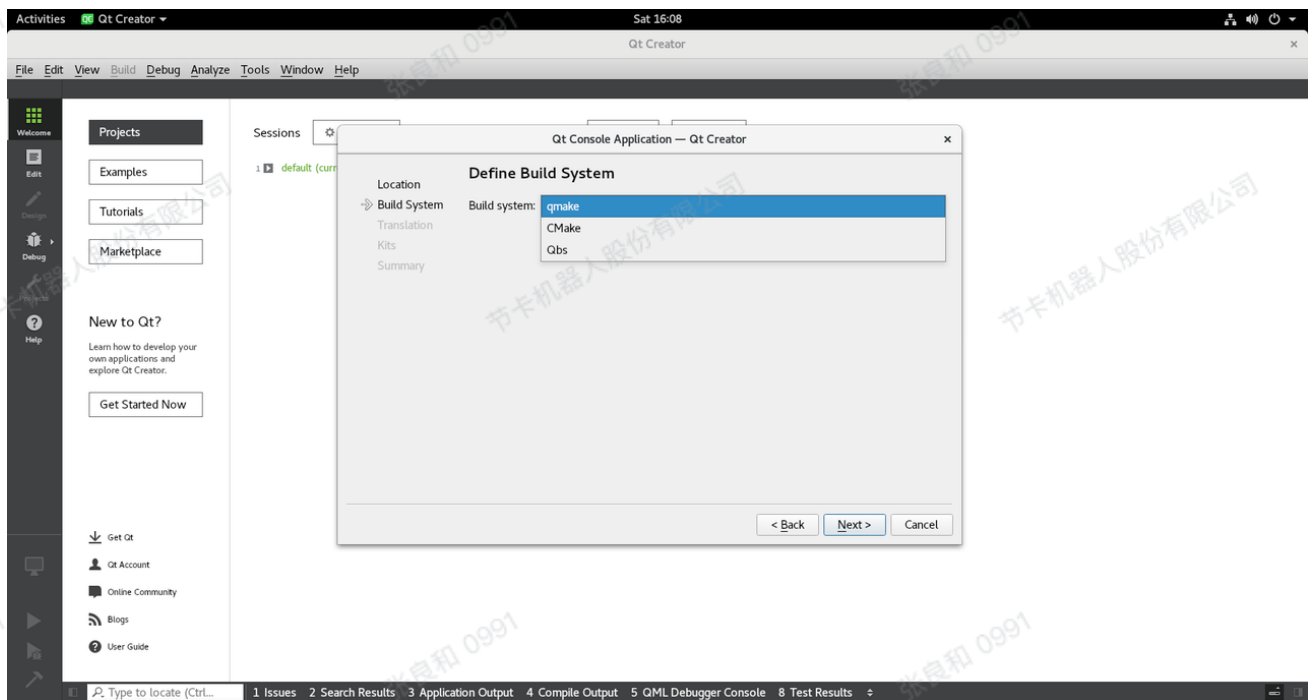
### 3.3.2 Create a Qt Application

Run Qt Creator and click [File] -> [New File or Project ...] on the menu bar to create a new Qt application. In the following example, select Application (Qt) -> Qt Console Application, i.e. Qt console application, and then follow the instructions to configure it.

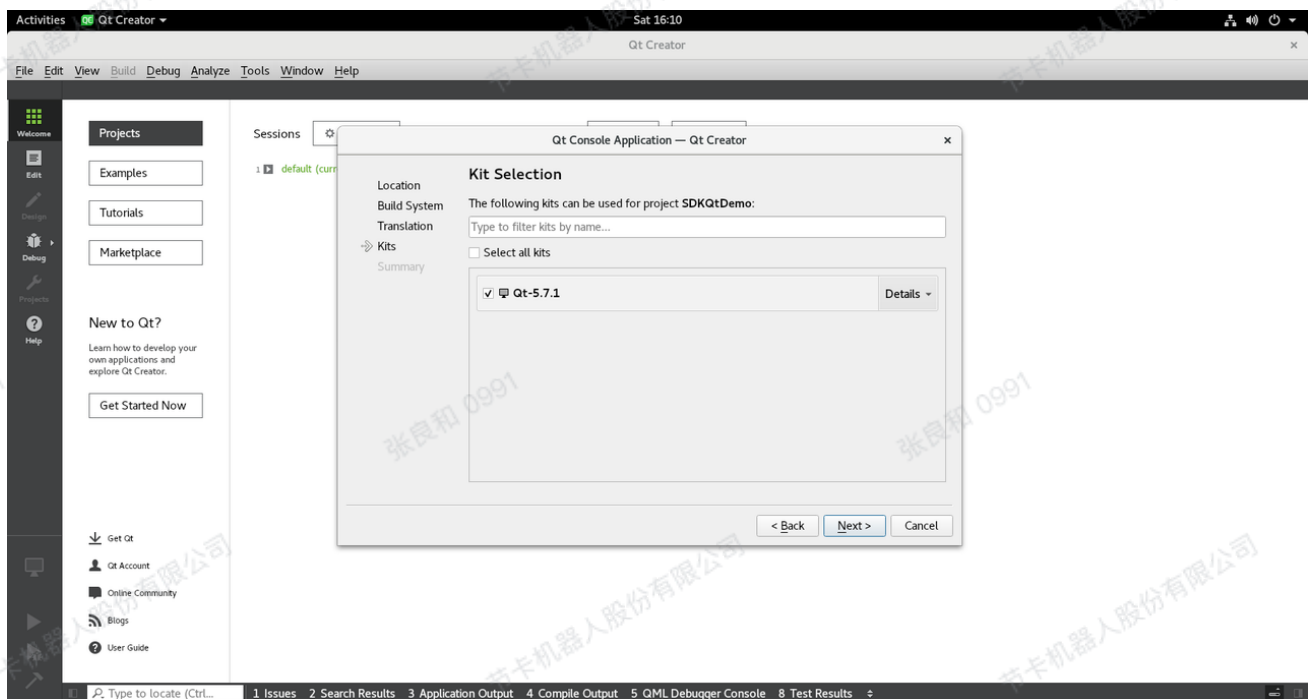


After configuring the project name and storage directory, you need to select the Build System. Qt supports multiple build systems. In this example, qmake is selected.



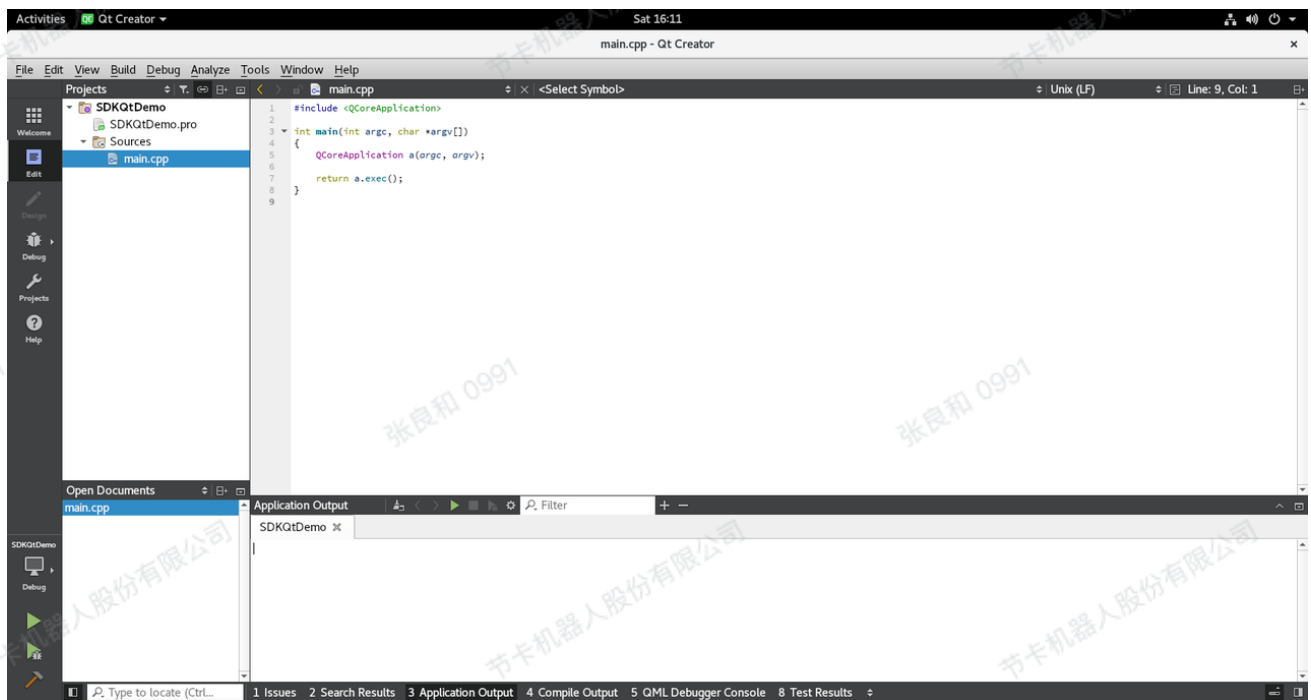


On the Kits selection interface, select the previously configured Kit, click Next and finally complete the project creation.



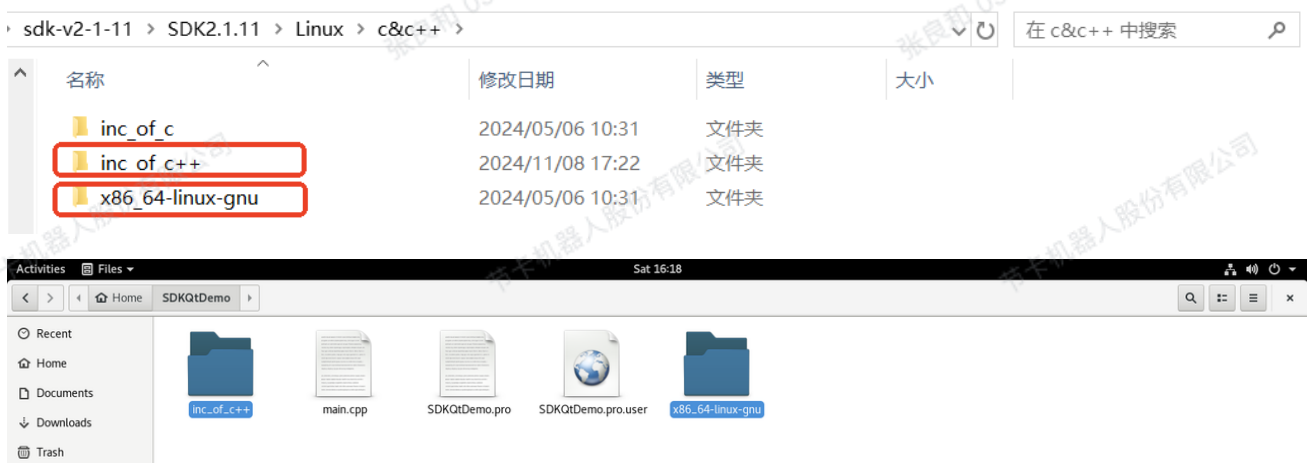
After the project is created, the initial interface is as follows.





### 3.3.3 Link Configuration for SDK

Add the header files and library files in the downloaded JAKA SDK to the project. This example is a Linux system, so copy the two folders `inc_of_c++` and `x86_64-linux-gnu` in the JAKA SDK folder to the project directory.



Double-click the project configuration file `SDKQtDemo.pro` in Qt Creator to add the header files and library files required to use the JAKA SDK to the project. The configuration added in this example is as follows:

- 1) Add the configuration line `INCLUDEPATH += inc_of_c++` to add the header file directory to the project;
- 2) Add the configuration line `LIBS += -L$(PWD)/x86_64-linux-gnu/shared -ljakaAPI` to add the `libjakaAPI.so` file to the project. `$(PWD)/x86_64-linux-gnu/shared` specifies the directory where the `libjakaAPI.so` file is located, and `jakaAPI` is the library name (i.e., `libjakaAPI.so`).



This completes the environment setup and configuration required for the Qt program to use the JAKA SDK.

### 3.3.4 Write application and use SDK

You can add `#include "JAKAZuRobot.h"` to the program and import the JAKA SDK header file to call the relevant interface. This example writes the following code for reference. Note: The IP address used in the example is the robot IP address, and the user needs to adjust it according to the situation.

▼ mian.cpp

复制代码

```

#include <QCoreApplication>
#include "JAKAZuRobot.h"
#include <string>
#include <vector>
#include <iostream>
#include <chrono>
#include <thread>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    JAKAZuRobot demo;

    demo.login_in("192.168.1.100");
    //sleep(2);
    demo.power_on();
    //sleep(2)
    demo.enable_robot();
    //sleep(2);

    CartesianPose tcp_pos;

```

```

int ret;
char ver[100];
demo.get_tcp_position(&tcp_pos);
demo.get_sdk_version(ver);

std::cout << "SDK version is :" << ver << std::endl;
std::cout << "tcp_pos is :\n x: " << tcp_pos.tran.x << "y: " << tcp_pos.tran.y << "z: " <<
tcp_pos.tran.z << std::endl;

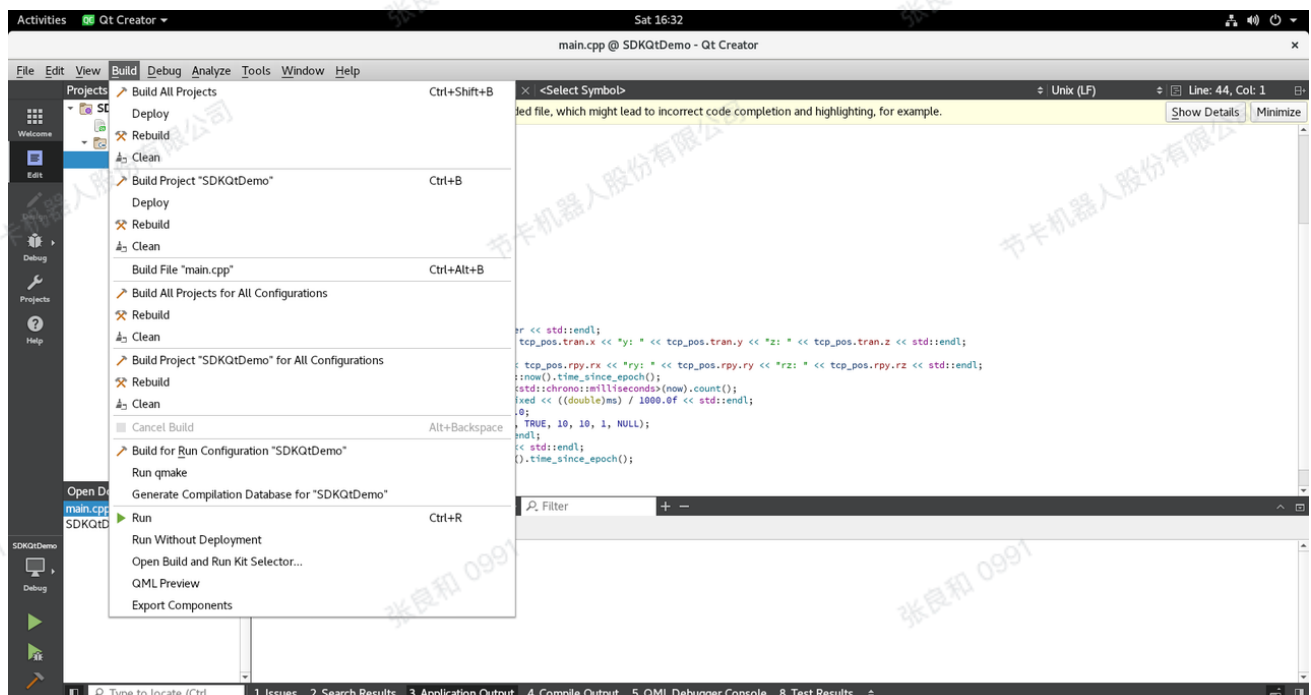
std::cout << "tcp_pos is :\n rx: " << tcp_pos.rpy.rx << "ry: " << tcp_pos.rpy.ry << "rz: "
<< tcp_pos.rpy.rz << std::endl;
auto now = std::chrono::system_clock::now().time_since_epoch();
auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(now).count();
std::cout << "Current s: " << std::fixed << ((double)ms) / 1000.0f << std::endl;
tcp_pos.tran.y = tcp_pos.tran.y + 60.0;
ret = demo.linear_move(&tcp_pos, ABS, TRUE, 10, 10, 1, NULL);
std::cout << "ret==" << ret << std::endl;
std::cout << "linear_move finish! " << std::endl;
now = std::chrono::system_clock::now().time_since_epoch();

return a.exec();
}

```

### 3.3.5 Compile and run the application

After the program is edited and saved, click the menu [Build]->[Run qmake] to generate the configuration file required for the build. After running qmake, it will prompt successful completion or exception (if any).



Then click [Build] -> [Build All Projects] to compile the source code and link it to generate an executable file. The Compile Output window at the bottom of Qt Creator will output the compilation results.



After successfully building, you can click the Run button on the left toolbar of Qt Creator to start the program. The running effect of this example is as follows.



## 4. Instructions for deploying SDK application as Addon

In order to facilitate some users to use SDK programming and deploy it in the controller to replace complex graphical programming operations, this chapter deploys and runs applications written using the Jieka SDK through Addon. Since the current Jieka Addon

function is to run the Python main file and use Python 2.7 by default, but the Jieka SDK is currently compiled with Python 3 or above, there will be problems when running the Python SDK library directly under Addon. Therefore, this document mainly explains how to call the SDK application written in the customer's C++ language through Addon, and the platform is the Linux system.

Therefore, the overall idea can be divided into two steps:

- 1) Develop user programs based on C/C++ that can run in JAKA controllers;
- 2) Develop an Addon that runs in the JAKA controller and eventually calls and executes the above SDK-based user job program.

## 4.1 Environment Setup

During the whole process, users need to prepare two development environments to complete the above two steps respectively:

- 1) SDK-based user program development environment;
- 2) JAKA Addon development environment.

### 4.1.1 Setup SDK application development environment

For this part, please refer to the process described in [3.2 Creating C++ Applications with CMake on Linux](#) for preparation.

### 4.1.2 Setup Addon development environment

First, prepare a demo developed by JAKA Addon (see the link below for download). After decompression, there are the following directories and files. The most important ones are the two files marked in red. Addon\_Demo.py is the main Python program, which is the entry program for the entire Addon runtime. Addon\_Demo\_config.ini is the Addon configuration file, which contains some configuration items about this Addon.

For detailed introduction of Addon, please refer to the relevant page of the Jaka Documentation Center: <https://www.jaka.com/docs/guide/addOn/1.2-AboutDev.html>.

名称	修改日期
client	2022/4/20 20:55
AddOn_Demo.py	2022/5/12 17:41
AddOn_Demo_config.ini	2022/5/9 14:45
readme.txt	2022/5/9 16:20
server_config.json	2022/5/9 16:47
server_config.py	2022/5/12 16:57

## 4.2 Development and Deployment

The following demonstrates how to deploy the development environment of Jieka SDK C++ under Linux. This development deployment requires the use of CMake tools. Readers are requested to download, install and configure them by themselves. The following explanation assumes that the reader has completed the configuration and has a certain development foundation.

### 4.2.1 Write user application

The writing of user operation programs can refer to the process described in [3.2 Creating C++ Applications with CMake on Linux](#) to create an executable program that can run in the robot controller environment. In this example, it is assumed that the user-developed operation program executable file is demo.

### 4.2.2 Run SDK application from Python

At this point, you need to write Python code in the Addon's Python main program to call the C++ executable program. The following are two ways for users to implement the call (Note: the default Python running version in the controller system environment is Python 2.7).

#### 1) Run via the subprocess module

▼ Addon\_Demo.py

📄 复制代码

```
import subprocess
import os
if __name__ == '__main__':
    # way 1
    args = ['./build/demo']
    result = subprocess.call(args)    # python2
```

#### 2) Run via the os module

▼ Addon\_Demo.py

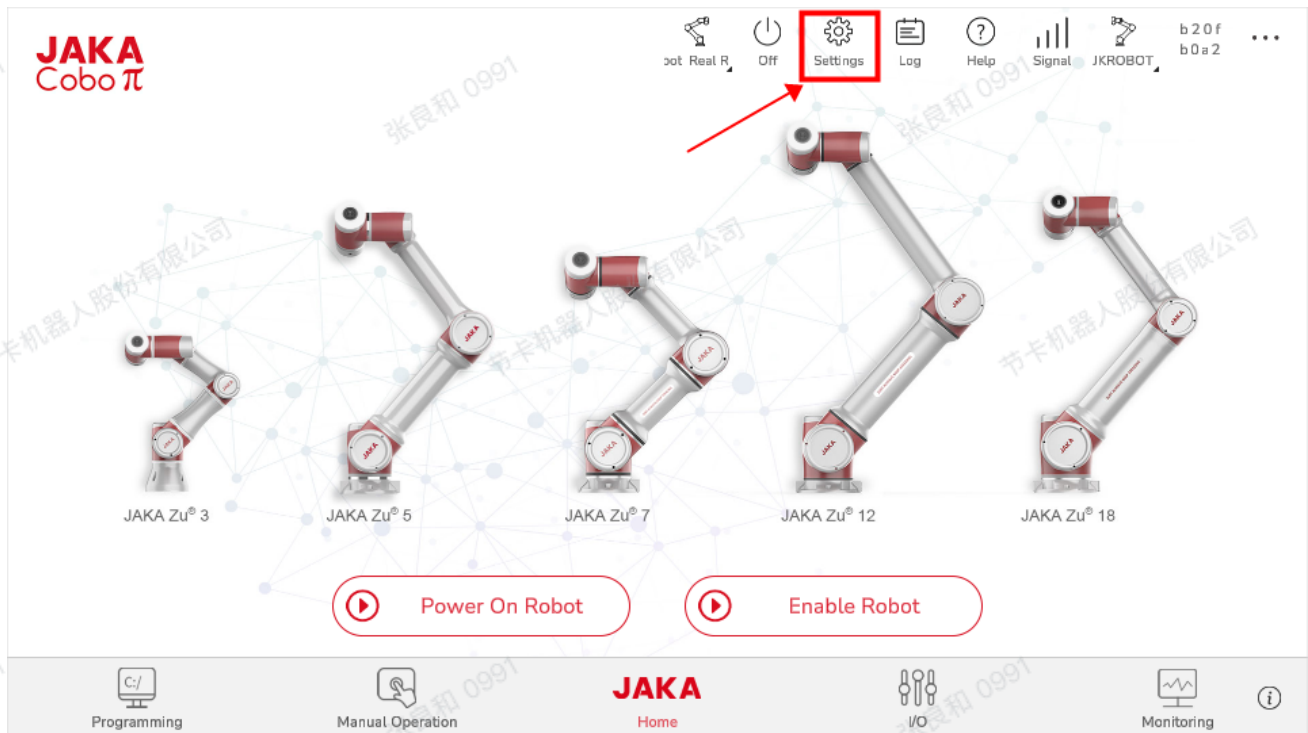
📄 复制代码

```
import subprocess
import os
if __name__ == '__main__':
    # way 2
    args = ['./build/demo']
    os.system('./build/demo')
```

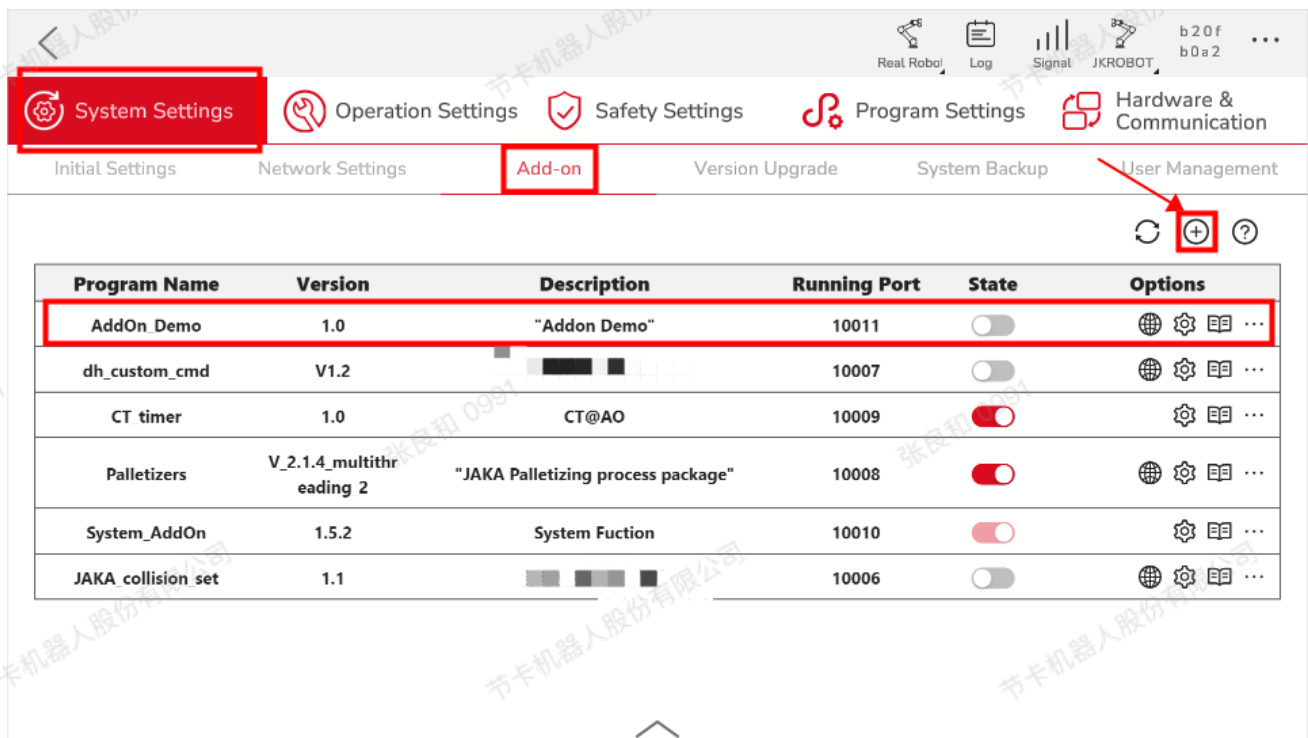
The above is an example of running the SDK program through the JAKA Addon framework. ./build/demo is the path of the user-developed program executable file under the developed Addon.

### 4.3 Run SDK application as Addon

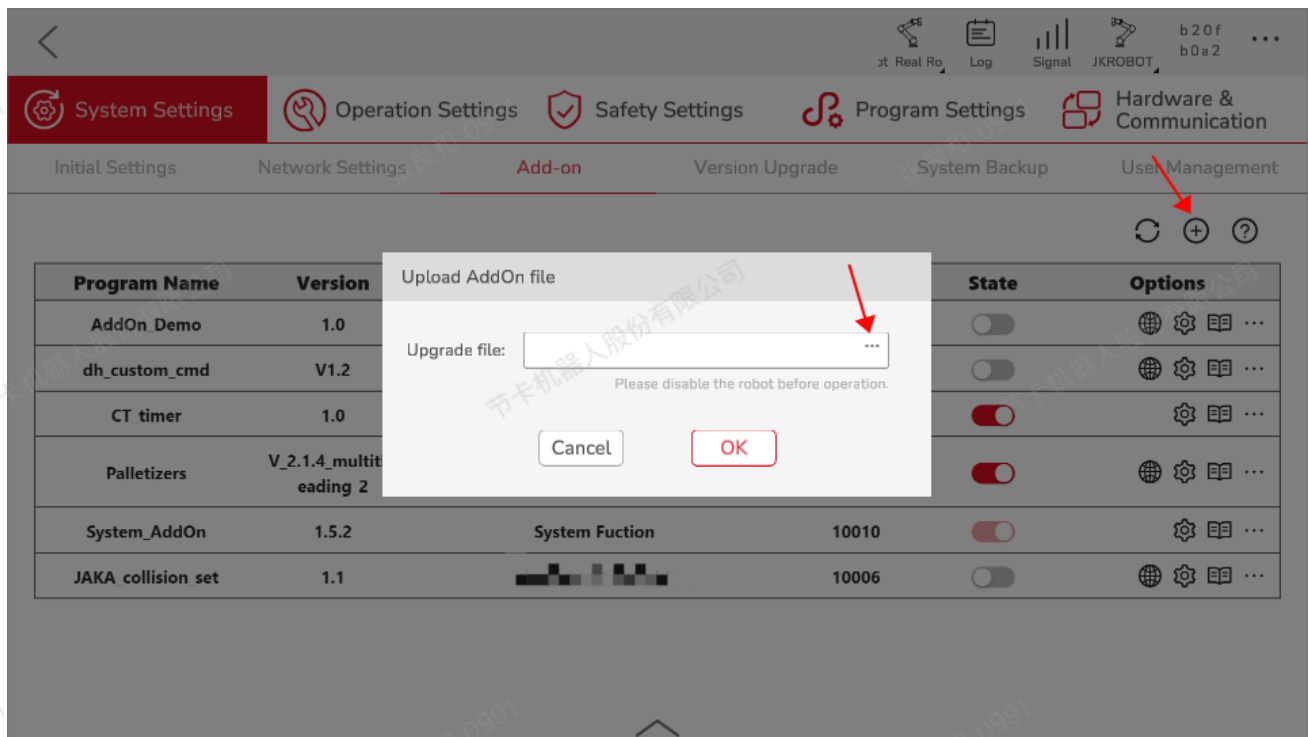
After writing the Addon, the user needs to package the project into tar.gz format to form an Addon software package. Then open the JAKA App software, You can go to [Settings]-->[System Settings]-->[Add-on], enter the add-on management interface, upload the Addon package to complete the installation. After the installation is complete, the user can control the start or end of the Addon on this page.





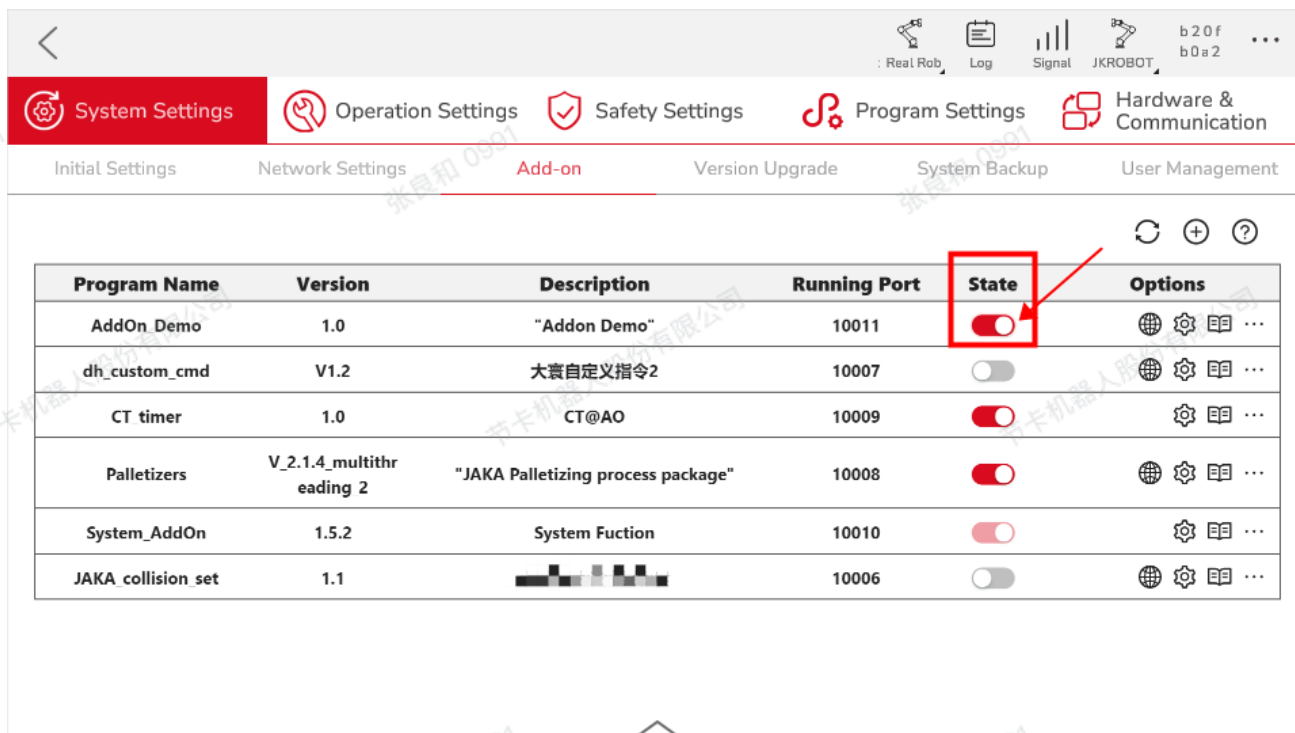


Select the Addon package in the local directory and wait for the file to be uploaded successfully.



Use the toggle button under the Stat column to turn this Addon on or off.





## 4.4 Precautions

- 1) When developing user applications based on SDK, you need to pay attention to the system architecture of the current control cabinet. JAKA controllers based on Debian Linux may have two configurations, x86 and x86\_64. You can connect to the controller through the JAKA App and check the controller version suffix to obtain the specific information;
- 2) If the SDK application was deployed on the controller, it will share the resources with the existing JAKA controller and other system services. Users need to plan resource usage reasonably, including the use of CPU, memory, and disk space, to avoid affecting the robot control system.