Instructions for Using STONE UART Instruction Parsing Library

1. Migration Instructions for STONE UART Instruction Parsing Library:

For Arduino Platform

When using the STONE UART Library on the Arduino platform, first open the example folder, then open the Arduino folder, and then copy the following files to the same directory as the Arduino project to be used (files with .ino suffix)

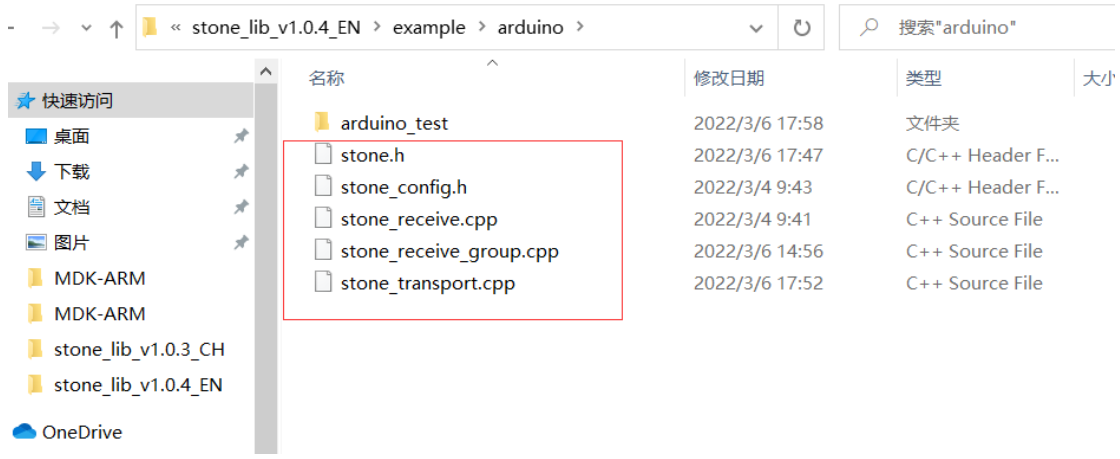

Figure 1.1 STONE library file

Figure 1.2 Copy the library files to the project directory

After the copy is successful, open the Arduino project to see all STONE UART library files in the IDE, as shown in the following figure

Figure 1.3 Library file displayed after opening the Arduino IED

Then what you need to do in your project is

1) Open “stone_config.h”, set the number after #define MCU_ARDUINO to 1, and ensure that the other three adjacent properties are set to 0
2) Import the header file #include stone.h
3) Serial port initialization “Serial.begin(115200)”;
4) Start the STONE serial port to receive “serial_receive()” in the loop function;

Figure 1.4 Initial configuration

At this point, you can compile and download it to your Arduino development board. please refer to readme.md file for specific function calls.

For STM32 Platform

When using in STM32, copy the files in Inc (include) and Src (source) to the same name directory of the MDK, STM32 IDE and other compiler projects, then add the c file and h file to the project, and select the file corresponding to the STONE UART Library.

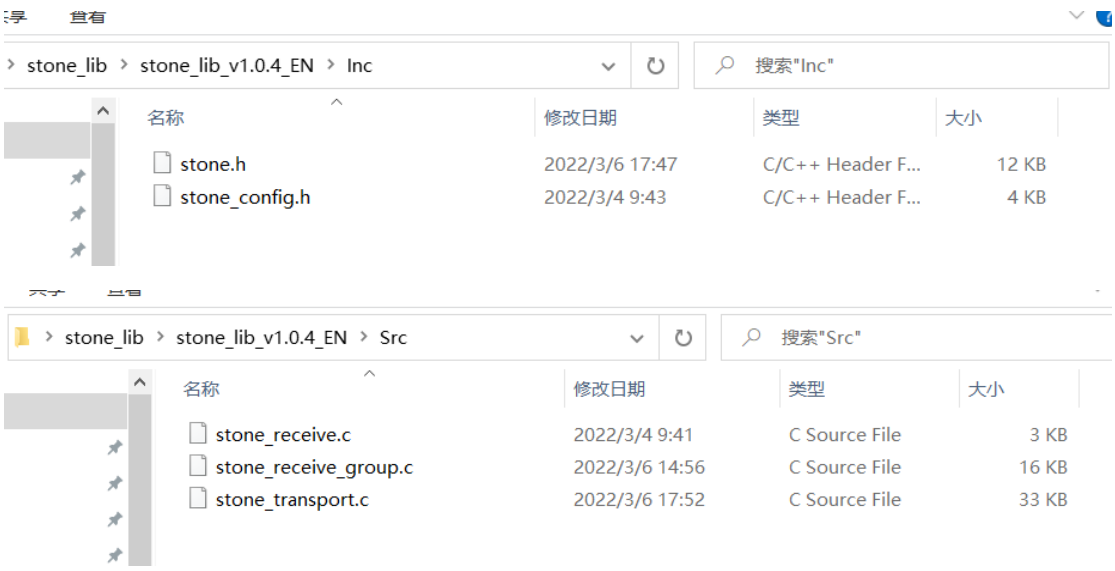

Figure 1.5 Files in Inc and Src in STONE UART Library

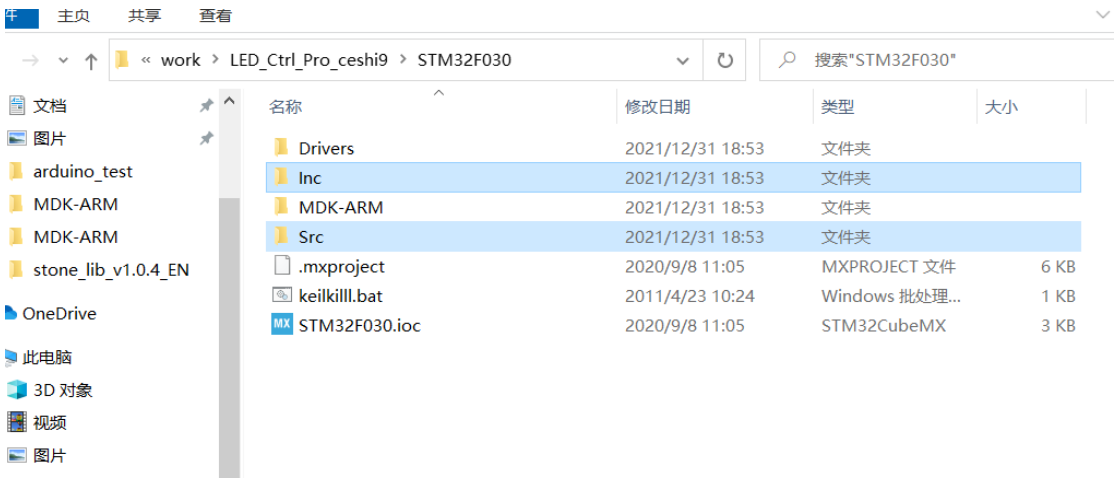

Figure 1.6 A folder with the same name in the keil MDK project

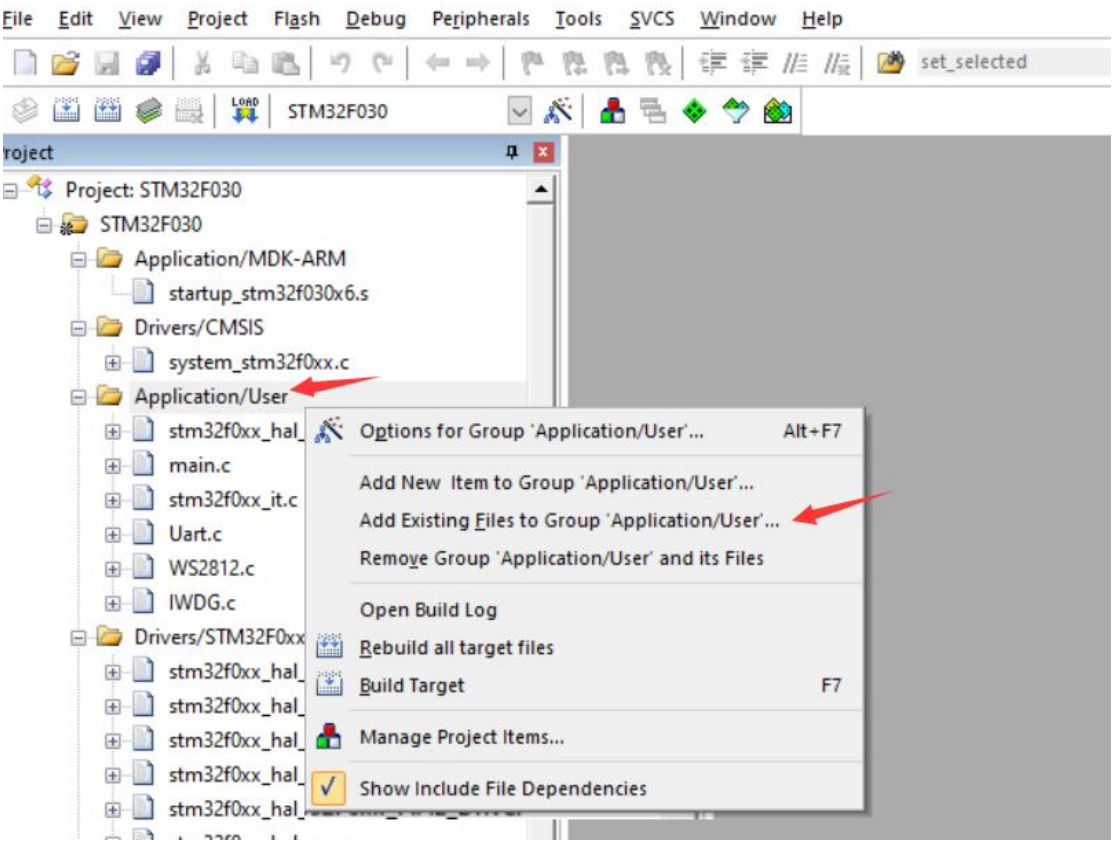

Figure 1.7 Right-click the project folder in keil and select Add Existing Files

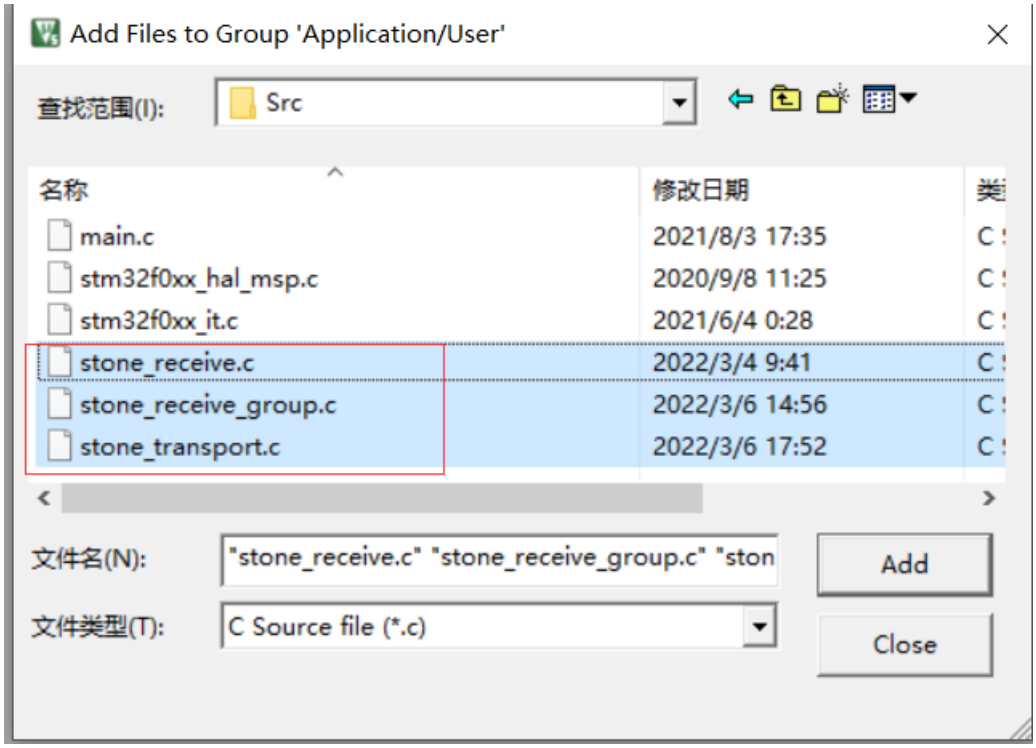

Figure 1.8 Click Add after selecting three .c files

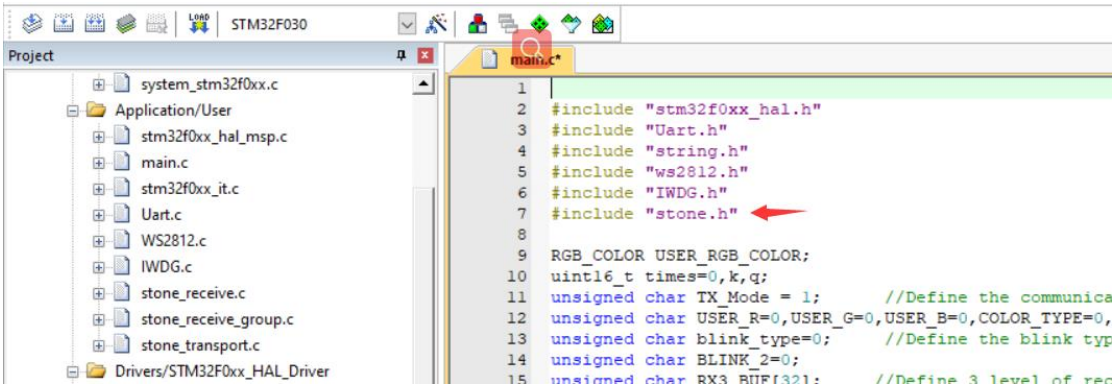

Figure 1.9 Import “stone.h” in the main.c file to use

The migration process for other platforms is similar. The main process is as follows:

1) Copy the source files and header files to the same directory as the project or specify the reference directory
2) Import "stone.h" in the main file
3) Start the STONE serial port to receive “serial_receive()” in the loop;

2. Instructions for Using STONE UART Instruction Parsing Library v1.0.4

This STONE UART Library is created to adapt the Instructions of the STONE UART Terminal. In order to increase the portability, the code is all written in C language, and has been tested and verified for several current mainstream MCUs, Please read this document before using it for the first time.

In the "stone_config.h" file, four initialization interfaces of the development board are currently defined:

```
#define MCU_STM32                0
#define MCU_ARDUINO              1
#define MCU_ESP                  0
#define MCU_Raspberry_Pi         0
```

If users use these four types of development boards, they only need to change the following definitions from 0 to 1. If they use other series of development boards, they can refer to the initialization templates of the four development boards defined later to adjust them,

The STONE UART Library of the development board itself and several standard libraries of C language are called in the initialization, and the delay functions in the library are also macro-defined as STONE_Delay, so it is easier to change in the header file.

After configuring the "stone_config.h" file, you can start to use all the functions in the library for data processing. The following describes the specific usage:

Receiving data is currently received byte by byte and then judged, if it meets the criteria then continue to receive, if not then discard.

The CRC check code of the two bytes after the end of the Instruction frame will also be verified to ensure the integrity of the received data,

When using the Arduino IDE, the received data can be viewed for legal or illegal prompts in the serial monitor.

For example, the return data of the button widget: 53 54 3C 10 01 00 09 62 75 74 74 6F 6E 35 32 1 3E 45 54 63 91 legitimate (this word is used for prompts)

After receiving, the data will be stored in the STONE_RX_BUF buffer with a size of 200 bytes. After that, the user can choose whether to call the Instruction data parsing interface, or not.

If it is enable, it will split and analyze each part of this data, which is convenient for users to interpret and use.

The storage structure when parsing data is as follows:

```
typedef struct Recive{
unsigned int        cmd;
unsigned int        len;
int                 value;
float               float_value;
long                long_value;
unsigned char       data;
unsigned char       widget;
unsigned char       text;
}recive_group;
```

The structure variables are as follows:
recive_group STONER;

The calling formats are STONER.cmd; STONER.len; STONER.value; STONER.float_value ; STONER.long_value; STONER.data; STONER.widget; STONER.text

“STONE.cmd” will save the “cmd_code” value in the returned Instruction for subsequent analysis, and users can also call it according to their own needs.

“STONE.cmd” will save the “len” value in the returned Instruction for subsequent analysis, and users can also call it according to their own needs.

When there is a value in the command, the various types of values corresponding to the return data will be converted to the corresponding type of value storage according to the type of the value.

For example, the return value 0 or 1 of the switch widget will be stored in “STONER.value”.
The return value 66.66 or 10.00 for the slider widget is stored in “STONER.float_value”.

The percentage value 40% returned by the progress bar widget, which is stored in 4 bytes in the Instruction, so the corresponding long type in Arduino will be stored in “STONER.long_value”.

“STONER.data” will create storage space for system-related Instructions or Instructions without widgets.
For example: The Data returned when getting the system version number 53 54 3C 0 2 0 10 56 32 2E 30 2E 30 37 2E 32 31 31 31 31 39 52 43 3E 45 54 7E 98

The 56 32 2E 30 2E 30 37 2E 32 31 31 31 31 39 52 43 in this segment of data is the hexadecimal format of the version number.

Then the value pointed to by “STONER.data” after parsing is V2.0.07.211119RC

“STONER.widget” will create storage space for Instructions with widgets
For example: Data returned by the button widget named “button9” 53 54 3C 10 1 0 8 62 75 74 74 6F 6E 39 1 3E 45 54 E7 E0

The 62 75 74 74 6F 6E in this segment of data is the hexadecimal format of the string "button9"
At this point, the value pointed to by “STONER.widget” is the string "button9".

“STONER.text” will create storage for Instructions with text values.
For example: The data returned after entering the text “abcd” by the edit widget on the operation screen is 53 54 3C 10 70 0 C 22 65 64 69 74 32 22 3A 61 62 63 64 3E 45 54 7 D2

The 61 62 63 64 in this segment of data is the hexadecimal format of the string “abcd”.
At this point, the value pointed to by “STONER.text” is “abcd”.

In order to save memory, the above three pointer variables will dynamically apply for memory space according to the required length, and considering the user's calling needs, they will not be automatically released after parsing,

The requested memory space will be created when the data is received for the first time, and then the space of the previous data application will be released after the new data is received, and then the application will be re-applied.

Users can manually release the allocated space by calling the “stone_recive_free()”; interface, or call “stone_recive_free("name")”; #name optional value can be data, widget, text.

There are currently two processing options for users to choose when sending an Instruction:

Solution 1

Call the packaged sending Instruction interface. In the "stone_transport.c" file, the user can see all the existing function interfaces for sending Instructions. All interfaces are divided into categories based on the sending Instruction “cmd_code”, reducing memory usage and complexity, such as “set_value()”; The interface can be used to set the value of edit, set the value of label, set the value of slider, set the value of switch, etc.

There are annotations and examples on each interface, so please set the parameters to be passed when calling according to the corresponding example.

For example: When setting the value of the label1 widget to 66.66, the original Instruction is:
ST<{"cmd_code":"set_value","type":"label","widget":"label1","value":66.66,"format":"%.2f"}>ET

Then the calling format is “set_value("label", "label5", "66.66", "%.2f")”;
It should be noted that the numerical parameters in the send Instruction are also all string types in the library, such as the above "66.66".

Solution 2

Call interface “STONE_JSON()”; This interface is a package function for sending Instructions. Each call will make a package, and the package will be stored in the send buffer after the package is completed.

For example: Set the text of “edit1” widget to “Hello Stone”, The original Instruction is:
ST<{"cmd_code":"set_text","type":"edit","widget":"edit1","text":"Hello Stone"}>ET

The calling formats are as follows:

```
STONE_JSON("cmd_code", "set_text");
STONE_JSON("type", "edit");
STONE_JSON("widget", "edit1");
STONE_JSON("text", "Hello Stone" , JSON_END);
```

After the above actions are completed, the package sending Instruction is completed. The rule is that the first parameter is the name in JSON format, and the second parameter is the value in JSON format, both in the form of strings, and the third parameter is an optional parameter. When the user completes the package this time, they need to add “JSON_END” at the end, which means the package is completed.

The send buffer of the above two solutions is “STONE_TX_BUF”, with a size of 200 bytes. The send implementation function has timeout processing. When the user continuously calls the send interface, ensure that each Instruction can be sent successfully.

It is recommended to use idle sending or RTOS sub-task sending to reduce the time occupied by sending.

3. Function Prototype and Use Example for Sending Instructions

1) Setting System Correlation

```
/ * Instruction interface for setting the system-related Instructions
  * function prototype:          void set_sys(char* m_cmd);
  * @param{m_cmd} m_cmd:   the correct value is "sys_reboot" or "sys_hello" or "sys_version"
  * Call the example:          set_sys("sys_reboot");
  * /
```

2) Setting System Sleep

```
/ * Instruction interface for the Instruction to set the screen backlight
  * function prototype:          void set_sleep(char* m_tf);
  * @param{m_tf} m_tf:          the correct value is”true” or ”false”
  * Call the example:          set_sys("sys_reboot");
  * /
```

3) Setting Buzzer

```
/ * Instruction interface for setting the buzzer time,In milliseconds
  * function prototype:          void set_buzzer(char* _time);
  * @param{_time} _time:        duration
  * Call the example:          set_buzzer("100");
  * /
```

4) Setting Backlight Brightness

```
/ * Instruction interface for setting backlight brightness, The value ranges from 0 to 100
  * function prototype:          set_brightness(_value)
  * @param{_value} _value:      the correct value is "0" to "100"
  * Call the example:          set_brightness("100");
  * /
```

5) Setting Touchscreen Calibration

```
/ * Instruction interface for setting Touch screen calibration (for resistive screens)
  * function prototype:          void set_touch_cal(void);
  * No need to pass parameters
  * Call the example:          set_touch_cal();
  * /
```

6) Clearing Touchscreen Calibration

```
/ * Instruction interface for remove Touch screen calibration (for resistive screens)
  * function prototype:          void clear_touch_cal(void);
  * No need to pass parameters
  * Call the example:          clear_touch_cal();
  * /
```

7) Touchscreen Testing

```
/ * Instruction interface for Touch screen test
  * function prototype:          void set_touch_test(void);
  * No need to pass parameters
  * Call the example:            set_touch_test();
  * /
```

8) Setting Widget Enabled State

```
/ * Instruction interface for the Instruction to set the enable state of the widget
  * function prototype:          void set_enable(char* m_name, char* m_tf, ...);
  * @param{m_name} m_name:      the widget whose value is the widget
  * @param{ m_tf} m_tf:         the correct value is”true” or ”false”
  * Call the example:            set_enable("switch1", "false");
  * /
```

9) Setting the Visible State of the Widget

```
/ * Instruction interface for the Instruction to set the enable state of the widget
  * function prototype:          set_visible(_name, _tf)
  * @param{_name} _name:        the widget whose value is the widget
  * @param{ _tf}_tf:            the correct value is”true” or ”false”
  * Call the example:            set_visible("button1", "false");
  * /
```

10) Setting Widget Coordinates

```
/ * Instruction interface for the Instruction to set the coordinate position of the widget
  * function prototype:          void set_coordinate(char* _name, char* _x, char* _y);
  * @param{_name} _name:        the widget whose value is the widget
  * @param{ _x} _x:              the value can be positive, negative or decimal
  * @param{ _y} _y:              the value can be positive, negative or decimal
  * Call the example:            set_coordinate("switch1", "100", "-200");
  * /
```

11) Setting Widget State

```
/ * Instruction interface for the Instruction Widget state
  * function prototype:          void set_state(char* _name, char* _state);
  * @param{_name} _name:        the widget whose value is the widget
  * @param{_state} _state:       the value is "normal" or "pressed" or "disable"etc.
  * Call the example:            set_state("button1", "pressed");
  * /
```

12) Setting the Widget Background Image

```
/ * Instruction interface for the Set the background image
 * function prototype:                 void set_bg_image(char* _name, char* _image);
 * @param{_name} _name :               the widget whose value is the widget
 * @param{ _image} _image:             the value is the image name in the gallery
 * Call the example:                   set_bg_image("button1", "guage_bg");
 * /
```

13) Setting the Display Color Relative to the Widget Color

```
/ * Instruction interface for the Set the background color
 * function prototype:                 void set_color(char* _name, char* _object, char* _color);
 * @param{_name} _name:                the widget whose value is the widget
 * @param{_object} _object: the value is the color-related properties contained in the current widget
 * @param{ _image}_image:color the value is in ABGR format from high to low, expressed in decimal
 * Call the example:                   set_color("button1", "bg_color", "4278190335");
 * /
```

14) Opening Any Window

```
/ * Instruction interface for the Instruction to open a window
 * function prototype:                 void open_win(char* _name, ...);
 * @param{_name} _name:                the widget whose value is the widget
 * Call the example:                   open_win("window2");
 * /
```

15) Closing Any Window

```
/ * Instruction interface for the Instruction to close the window
 * function prototype:                 close_win(_name)
 * @param{_name} _name:                the widget whose value is the widget
 * Call the example:                   close_win("window1");
 * /
```

16) Returning to Upper-Level Window

```
/ * Instruction interface to return to the previous level of window
 * function prototype:                 void back_win(void);
 * No need to pass parameters
 * Call the example:                   back_win();
 * /
```

17) Returning to Any Upper-Level Window

```
/ * Instruction interface for the Instruction to close the window
 * function prototype:                 back_win_to(_name)
 * @param{_name} _name:                the widget whose value is the widget
 * Call the example:                   back_win_to("window1");
 * /
```

18) Returning to the Main Window

```
/ * Instruction interface for returning the main window Instruction
  * function prototype:          void back_home(void);
  * No need to pass parameters
  * Call the example:           back_home();
  * /
```

19) Setting the Text Content Displayed in the Widget

```
/ * Instruction interface for Setting the text Instructions
  * function prototype:          void set_text(char* _type, char* _name, char* _text);
  * @param{_type} _type:        the value is the type of the widget
  * @param{_name} _name:        the widget whose value is the widget
  * @param{ _text} _text:       the value is the text content to be set
  * Call the example:           set_text("label", "label5", "hello stone");
  * /
```

20) Setting the Value Displayed in the Widget

```
/ * Instruction interface for the set value Instruction
  * function prototype:          void set_value(char* _type, char* _name, char* _value, ...);
  * @param{_type} _type:        the value is the type of the widget
  * @param{_name} _name:        the widget whose value is the widget
  * @param{ _value} _value:     the value is the value to be set
  * @param{ ...} ...:           optional parameter, can be set or not set the format value of the value
  * Call the example:           set_value("slider", "slider2", "66");
  * Call the example:           set_value("label", "label5", "66.6", "%.2f");
  * /
```

21) Getting the Text Content Displayed in the Widget

```
/ * Instruction interface for getting text Instructions
  * function prototype:          void get_text(char* _type, char* _name);
  * @param{_type} _type:        the value is the type of the widget
  * @param{_name} _name:        the widget whose value is the widget
  * Call the example:           get_text("edit", "edit1");
  * /
```

22) Getting the Value in the Widget

```
/ * Instruction interface for the get value Instruction
  * function prototype:          void get_value(char* _type, char* _name);
  * @param{_type} _type:        the value is the type of the widget
  * @param{_name} _name:        the widget whose value is the widget
  * Call the example:           get_value("edit", "edit2");
  * /
```

23) Setting Maximum Value

```
/ * Instruction interface for the get value Instruction
 * function prototype:          void set_max(char* _type, char* _name, char* value);
 * @param{_type} _type:        the value is the type of the widget
 * @param{_name}_name:         the widget whose value is the widget
 * @param{value} value:        the value is a numeric value, in the form of a string
 * Call the example:           set_max("slider", "slider2", "500");
 * /
```

24) Setting Display Text Content

```
/ * Instruction interface for displaying text Instructions
 * function prototype:          void show_text(char* _name, char* _tf);
 * @param{_name}_name:         the widget whose value is the widget
 * @param{_tf} _tf:            the correct value is”true” or ”false”
 * Call the example:           show_text("progress_bar1", "true");
 * /
```

25) Getting Progress Bar Percentage

```
/ * Instruction interface for the Get Progress Bar Percentage Instruction
 * function prototype:          void get_percent(char* _name);
 * @param{_name}_name:         the widget whose value is the widget
 * Call the example:           get_percent("progress_bar1");
 * /
```

26) Setting Minimum Value

```
/ * Instruction interface for the set minimum Instruction
 * function prototype:          void set_min(char* _type, char* _name, char* value);
 * @param{_type} _type:        the value is the type of the widget
 * @param{_name}_name:         the widget whose value is the widget
 * @param{value} value:        the value is a numeric value, in the form of a string
 * Call the example:           set_min("slider", "slider2", "1");
 * /
```

27) Setting Step Value

```
/ * Instruction interface for setting slider step value Instruction
 * function prototype:          void set_step(char* _name, char* value);
 * @param{_name}_name:         the widget whose value is the widget
 * @param{value} value:        the value is a numeric value, in the form of a string
 * Call the example:           set_step("slider1", "1");
 * /
```

28) Playing Animation Widget

```
/ * Instruction interface for setting the picture animation playback Instruction
  * function prototype:          void set_play(char* _name,...);
  * @param{_name _name:         the widget whose value is the widget
  * Call the example:           set_play("image_animation1");
  * /
```

29) Pausing Animation Widget

```
/ * Instruction interface for setting the picture animation playback Instruction
  * function prototype:          set_pause(char* _name);
  * @param{_name} _name:        the widget whose value is the widget
  * Call the example:           set_pause("image_animation1");
  * /
```

30) Stopping Animation Widget

```
/ * Instruction interface for setting the picture animation playback Instruction
  * function prototype:          set_stop(char* _name);
  * @param{_name} _name:        the widget whose value is the widget
  * Call the example:           set_stop("image_animation1");
  * /
```

31) Setting Format

```
/ * Instruction interface for setting format Instructions
  * function prototype:          void set_format(char* _type, char* _name, char* _format);
  * @param{_type}_type:         the value is the type of the widget
  * @param{_name} _name:        the widget whose value is the widget
  * @param{_format} _format:    the value is ”%d” or ”%02d” or ”%03d” or ”%04d” or ”%05d” or ”%06d”
                                or ”%f” or ”%.1f” or ”%.2f” or ”%.3f” or ”%.4f” or ”%.5f” or ”%.6f”
  * Call the example:           set_format("image_value", "iamge_value1", "%.2f");
  * /
```

32) Setting the Picture Displayed by the Picture Widget

```
/ * Instruction interface for setting picture Instructions
  * function prototype:          void set_image(char* _type, char* _name, char* _image);
  * @param{_type} _type:        the value is the type of the widget
  * @param{_name} _name:        the widget whose value is the widget
  * @param{_image} _image:      the value is the image name in the gallery
  * Call the example:           set_image("image_value", "iamge1", "light_on");
  * /
```

33) Setting the Animation Playback Interval

```
/ * Instruction interface for the set picture animation interval Instruction
  * function prototype:                void set_interval(char* _name, char* _interval);
  * @param{_name}_name:                the widget whose value is the widget
  * @param{_interval}_interval:        the value is a numeric value, in the form of a string
  * Call the example:                  set_interval("image_animation1", "100");
  * /
```

34) Setting the Animation Loop

```
/ * Instruction interface for setting the loop Instruction
  * function prototype:                void set_loop(char* _name, char* _tf);
  * @param{_name}_name :               the widget whose value is the widget
  * @param{_tf}_tf :                   the correct value is ”true” or ”false”
  * Call the example:                  set_loop("image_animation1", "true");
  * /
```

35) Setting the Start Position and End Position of Animation Playback

```
/ * Instruction interface for setting the loop Instruction
  * function prototype:                void set_range(char* _name, char* start, char* end);
  * @param{_name}_name:                the widget whose value is the widget
  * @param{start} start:               the value is a numeric value, in the form of a string
  * @param{end} end:                   the value is a numeric value, in the form of a string
  * Call the example:                  set_range("image_animation1", "1" , "10");
  * /
```

36) Setting the Way of Drawing an Image

```
/ * Instruction interface for the set picture display method Instruction
  * function prototype:                void set_draw_type(char* _type, char* _name, char* draw_type);
  * @param{_type} _type:               the value is the type of the widget
  * @param{_name _name:                the widget whose value is the widget
  * @param{draw_type} draw_type:the value is a numeric value from 0 to 11, in the form of a string
  * Call the example:                  set_draw_type("image", "image1" , "2");
  * /
```

37) Setting the Zoom Ratio of the Image

```
/ * Instruction interface for the set image scaling Instruction
  * function prototype:          void set_scale(char* _name, char* _x, char* _y);
  * @param{_name} _name:         the widget whose value is the widget
  * @param{ _x} _x :              the value can be a positive number or a decimal, in the form of a string
  * @param{ _y} _y :              the value can be a positive number or a decimal, in the form of a string
  * Call the example:            set_scale("image2", "0.5" , "1");
  * /
```

38) Setting the Rotation Angle of the Image

```
/ * Instruction interface for the set picture rotation angle Instruction
 * function prototype:                void set_rotation(char* _name, char* _rotation);
 * @param{_name}_name:                the widget whose value is the widget
 * @param{ _rotation} _rotation:      the value is a numeric value, in the form of a string
 * Call the example:                  set_rotation("image1", "180");
 * /
```

39) Setting the Current Selection of the Text Selector

```
/ * Instruction interface for setting the current selector Instruction of the text selector
 * function prototype:                void set_selected(char* _name, char* _selected);
 * @param{_name}_name :               the widget whose value is the widget
 * @param{_selected}_selected:        the value is a numeric value, in the form of a string
 * Call the example:                  set_selected("text_selector1", "20");
 * /
```

40) Getting the Current Selection of the Text Selector

```
/ * Instruction interface to get the current selector Instruction of the text selector
 * function prototype:                void get_selected(char* _name);
 * @param{_name}_name:                the widget whose value is the widget
 * Call the example:                  get_selected("text_selector1");
 * /
```

41) Setting Date

```
/ * Instruction interface for the set date Instruction
 * function prototype:                void set_date(char* _name, char* _date);
 * @param{_name}_name:                the widget whose value is the widget
 * @param{ _date} _date:              the value is a date, in the form of a string
 * Call the example:                  set_date("2022-02-22 22:22:22");
 * /
```

42) Getting Date

```
/ * Instruction interface for the get date Instruction
 * function prototype:                void get_date(char* _name);
 * @param{_name}_name :               the widget whose value is the widget
 * Call the example:                  get_date("clock1");
 * /
```

43) Getting the Currently Checked Radio Button

```
/ * Instruction interface for the get checked Instruction
 * function prototype:                void get_checked(char* _name);
 * @param{_name}_name:                the widget whose value is the widget
 * Call the example:                  get_checked("check1");
 * /
```

44) Setting the Rotation Angle of the Pointer

```
/ * Instruction interface for the set angle Instruction
  * function prototype:              void set_angle(char* _name, char* _angle);
  * @param{_name}_name:             the widget whose value is the widget
  * @param{_angle}_angle:           the value is a numeric value, in the form of a string
  * Call the example:               set_angle("pointer1", "10");
  * /
```