SGL(Super Graphic Library) v4.1.0

From version 4.0.0, all tutorials are written in this README.

Introduction

V4.1.0 is the twenty-first version of SGL. V4.x focus on the User experience. This time SGL add sub window and simplify the widget callback usage. Besides, SGL add json operations for socket transferring. For the late 4.x version, SGL will implement more widget attributes to cover the requirements. Now the source code is opened and welcome any suggestions.

The main purpose of the SGL is to build an excellent graphic coding environment, wish all users have a good coding time!

Upcoming

Draggable widget and a entirely new input method are now developing.

Links

Instruction videos can be acquired from

https://v.douyu.com/author/PDAPmLyG87xN and

http://space.bilibili.com/33137499/#!/index

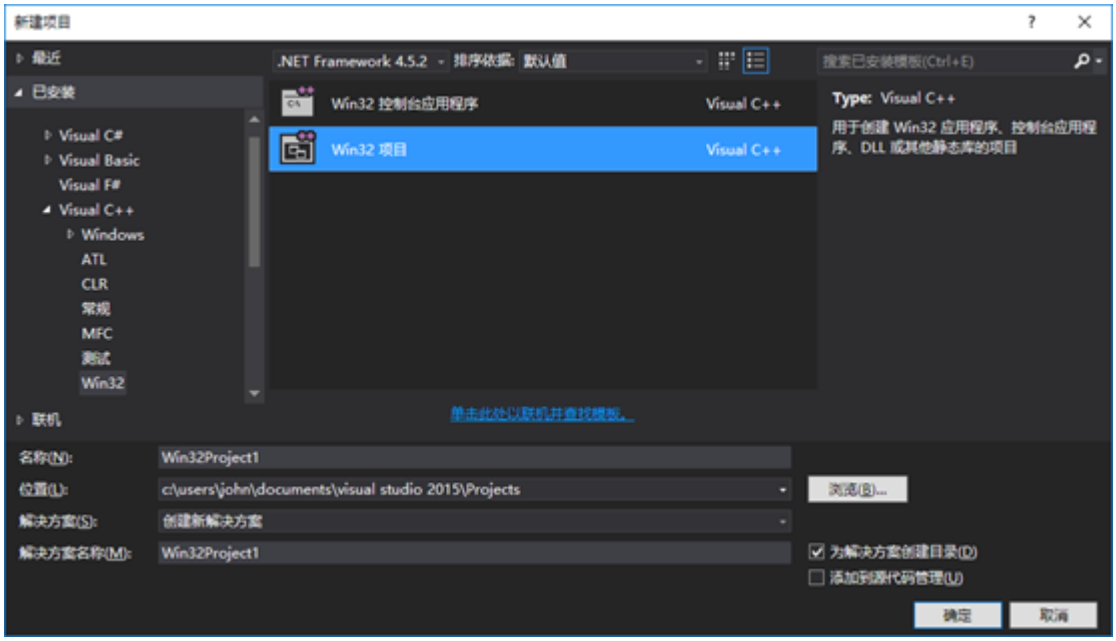
Configuration

The latest library zip can be acquired from GitHub only(https://github.com/SuperABC/SGL). Unzip the zip and put the "Library" folder in your favorite place.

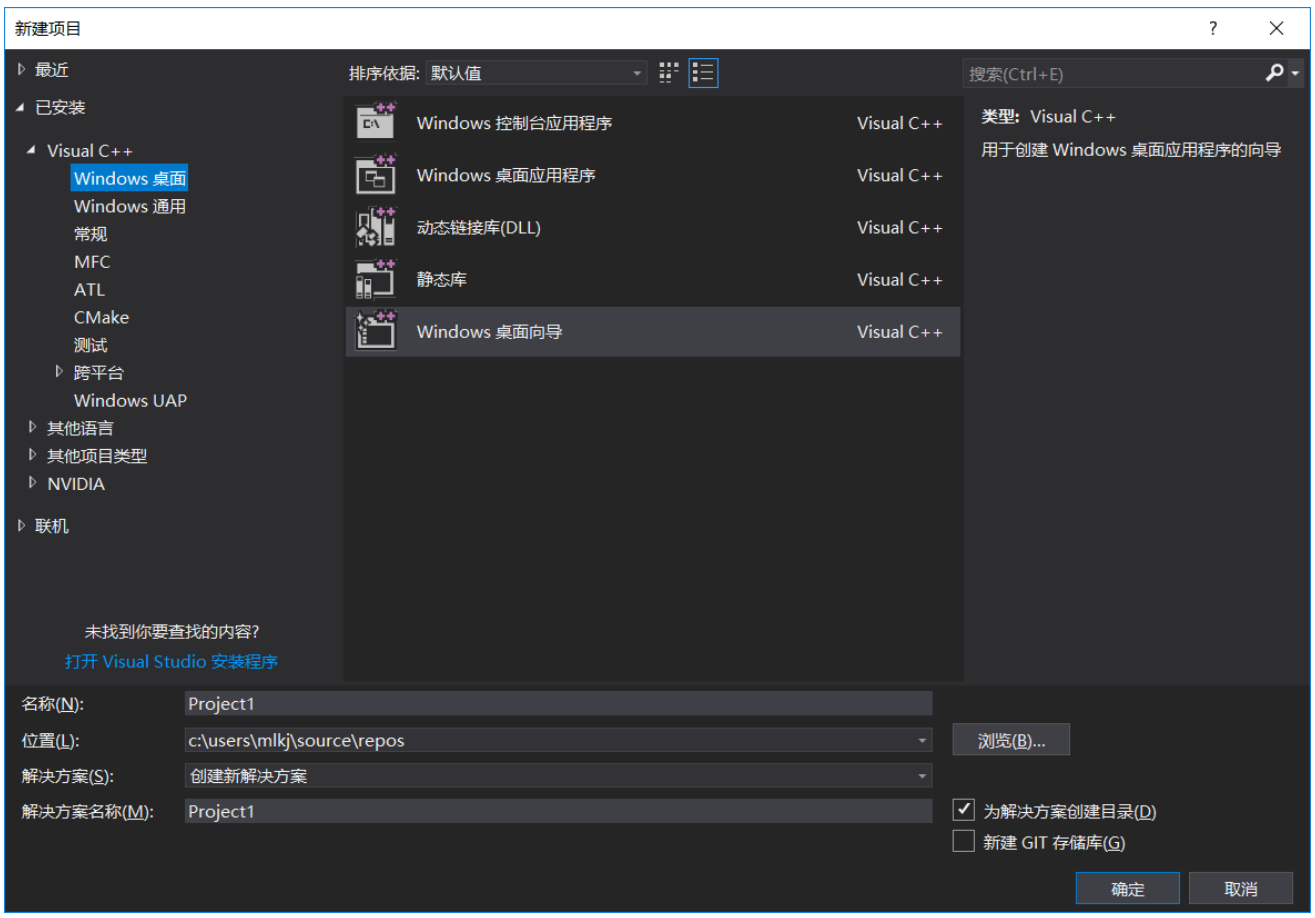
Visual Studio

Open VS and create a new project. Remember not to choose console application but win32 program so that there won't be a black frame when running.

If you are using VS 2015, then choose the following item.

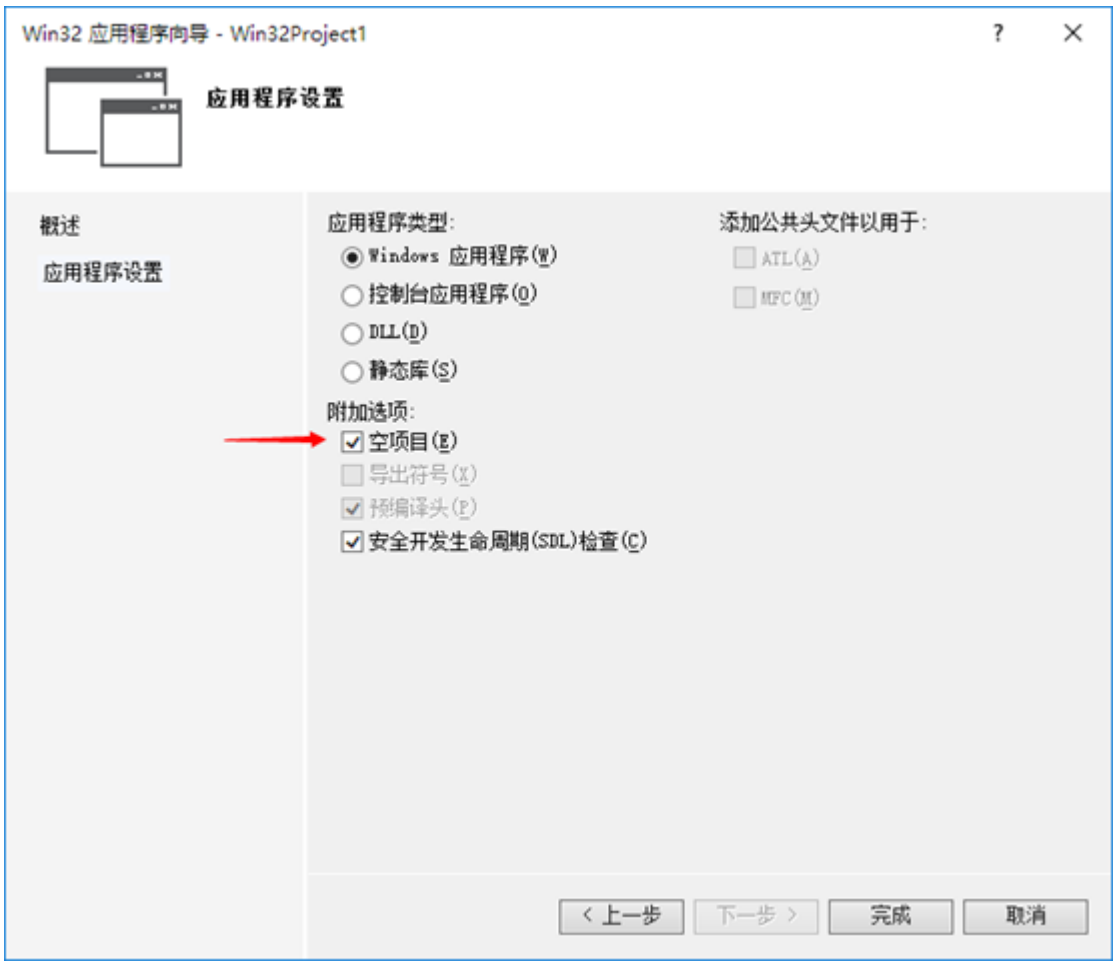


Or else, VS 2017, then choose the item shown below.

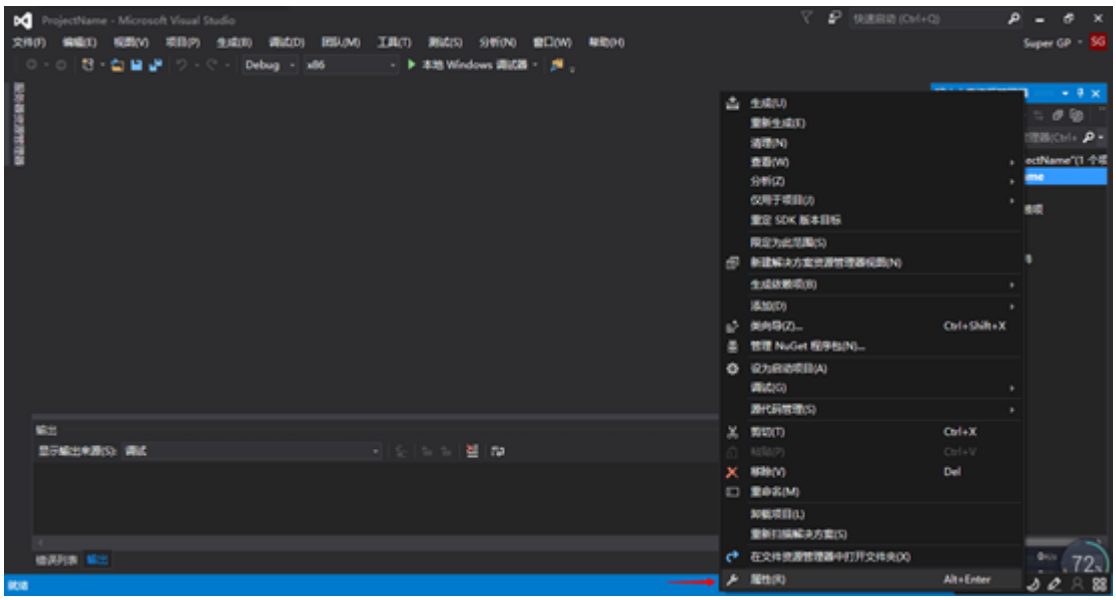


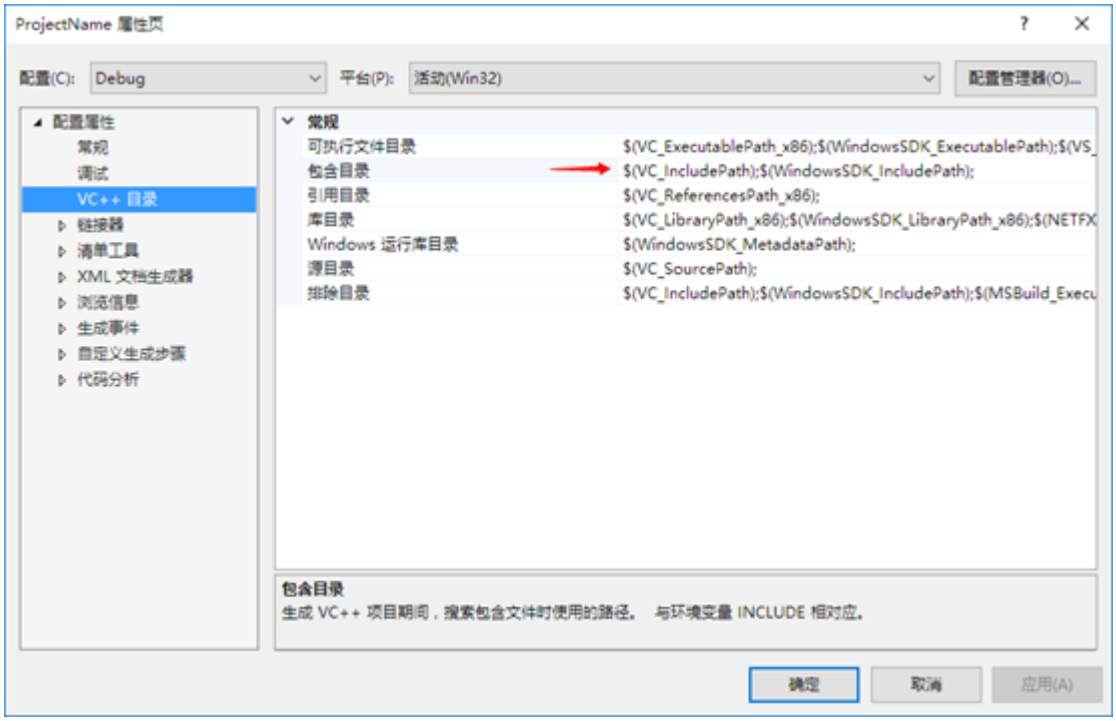
After selecting the right project type, the following steps are almost the same.

Set attributes properly.

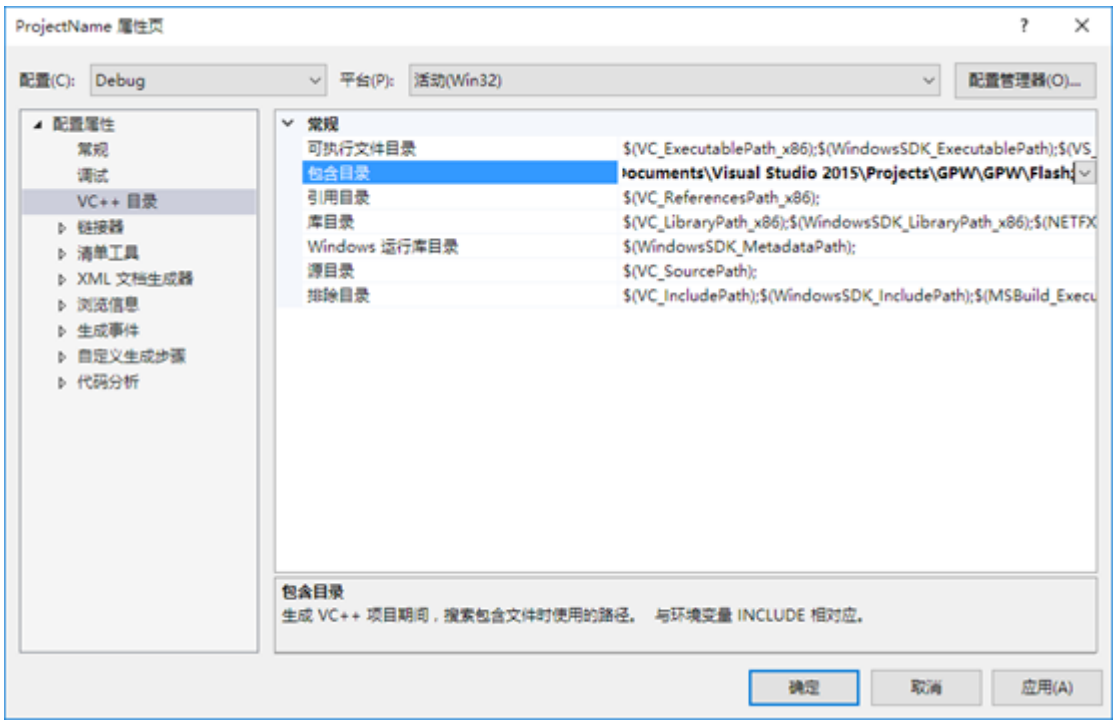


Then click your project with the right mouse button to open the attribute window.

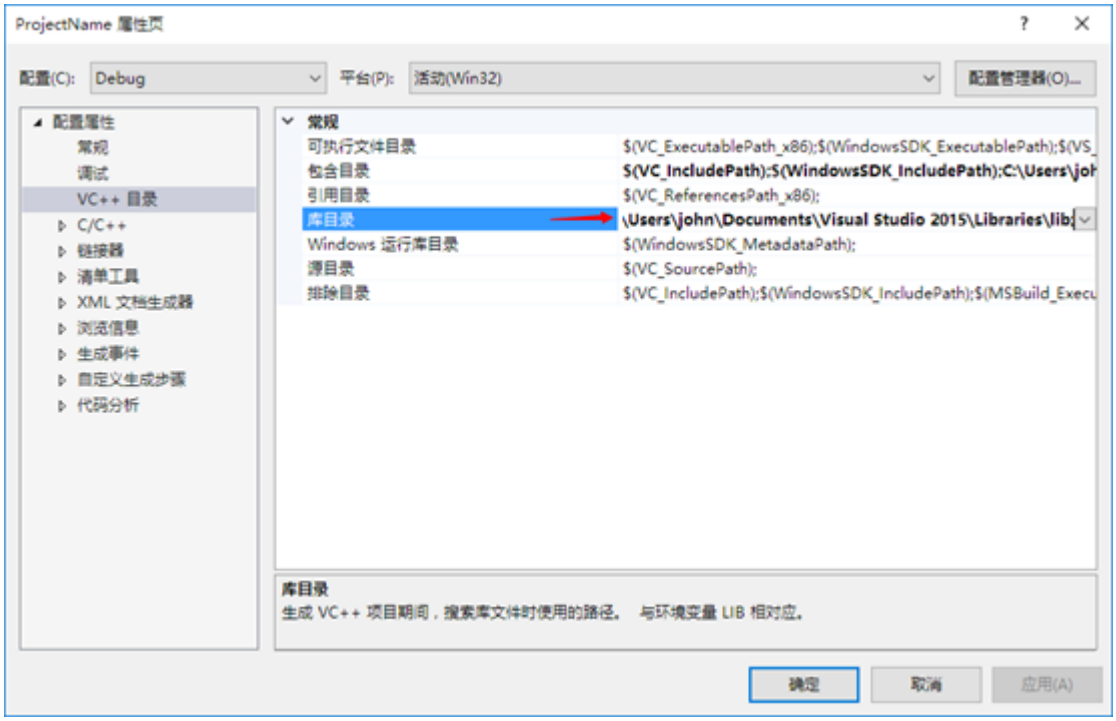




Add your folder which includes winsgl.h. If more than one paths need to be added, separate them with semicolon.

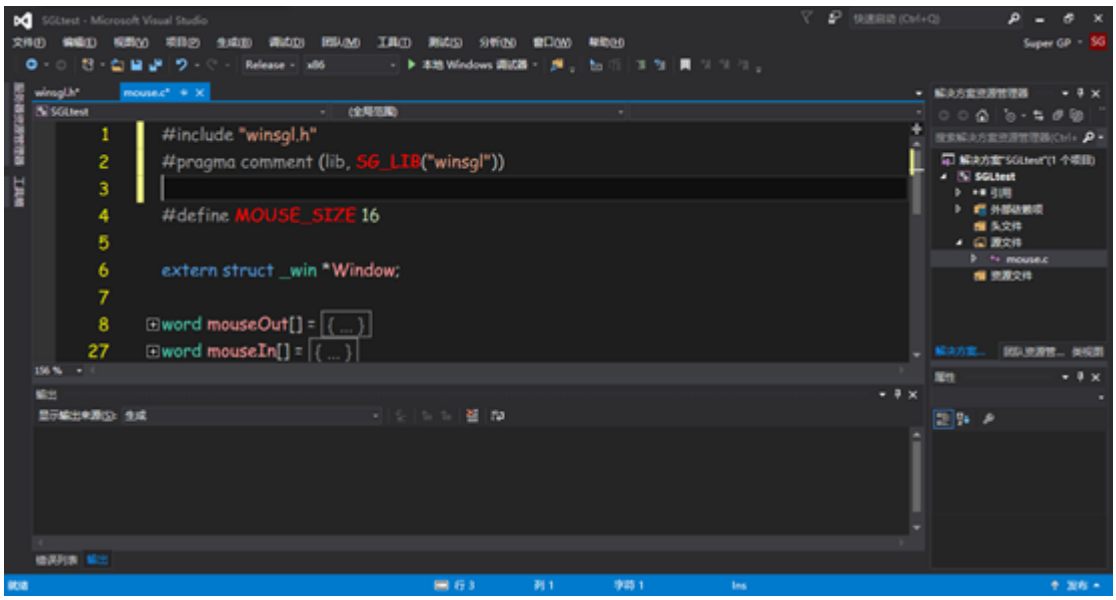


Add your folder which includes winsgl.lib/winsgld.lib. Again, if more than one paths need to be added, separate them with semicolon.



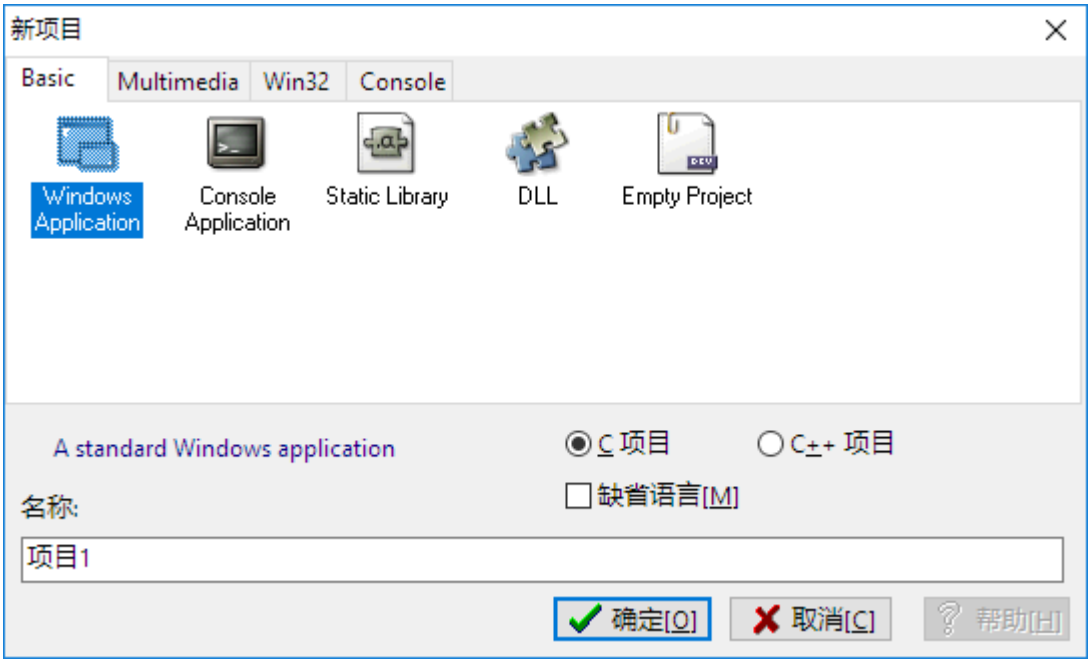
Here comes the last step. We need to link the lib file to the project. The simplest method is to add one line as follow:

```
#pragma comment (lib, SG_LIB("winsgl"))
```



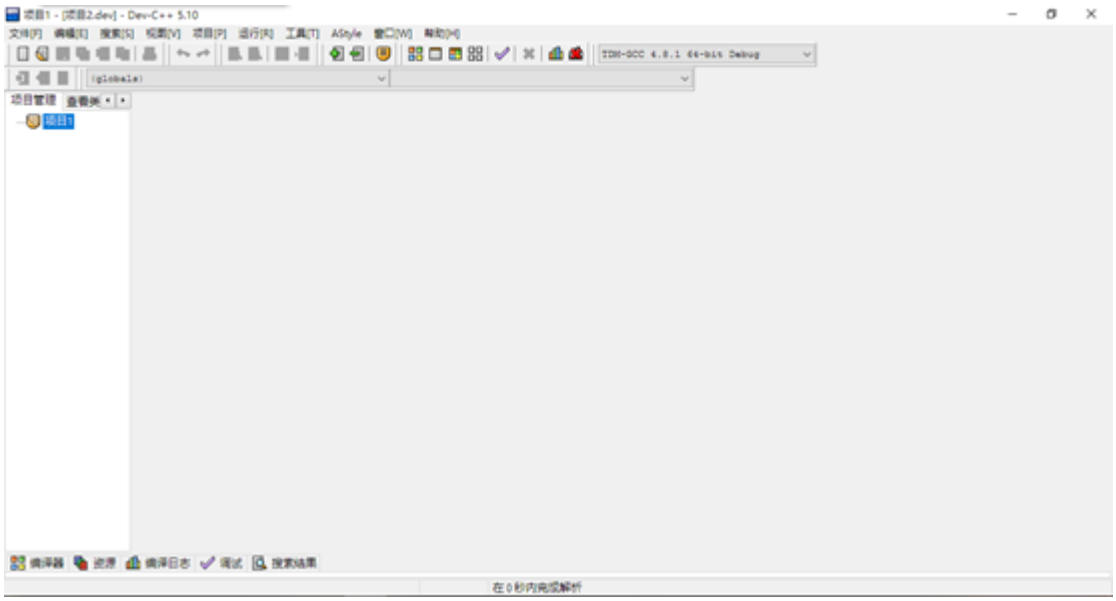
Dev-cpp

Open Dev-cpp and create a new project. Remember that we need to create a project not a source file.

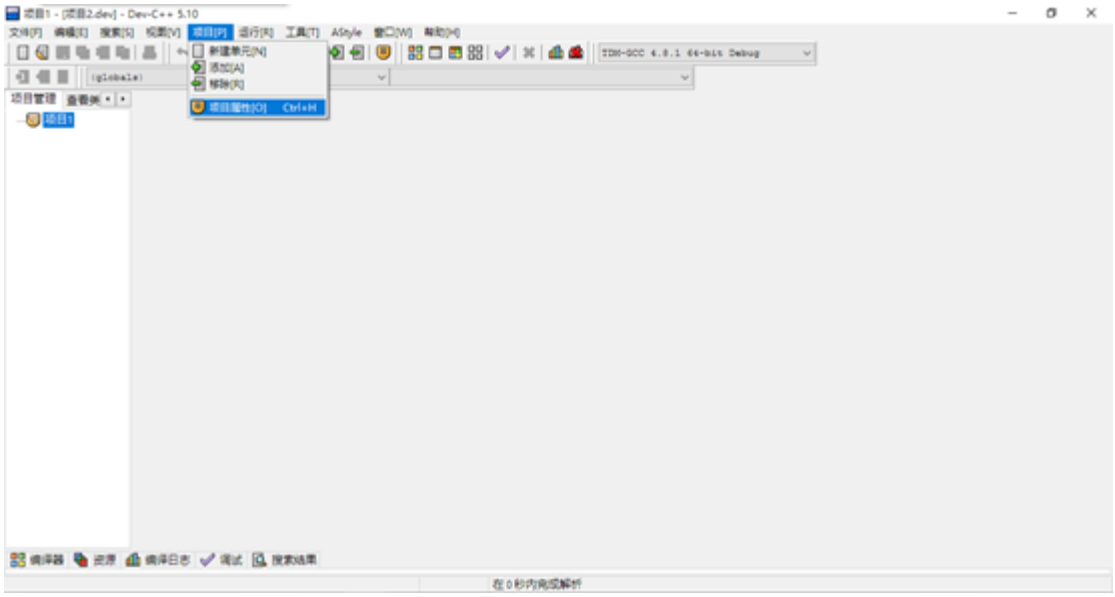


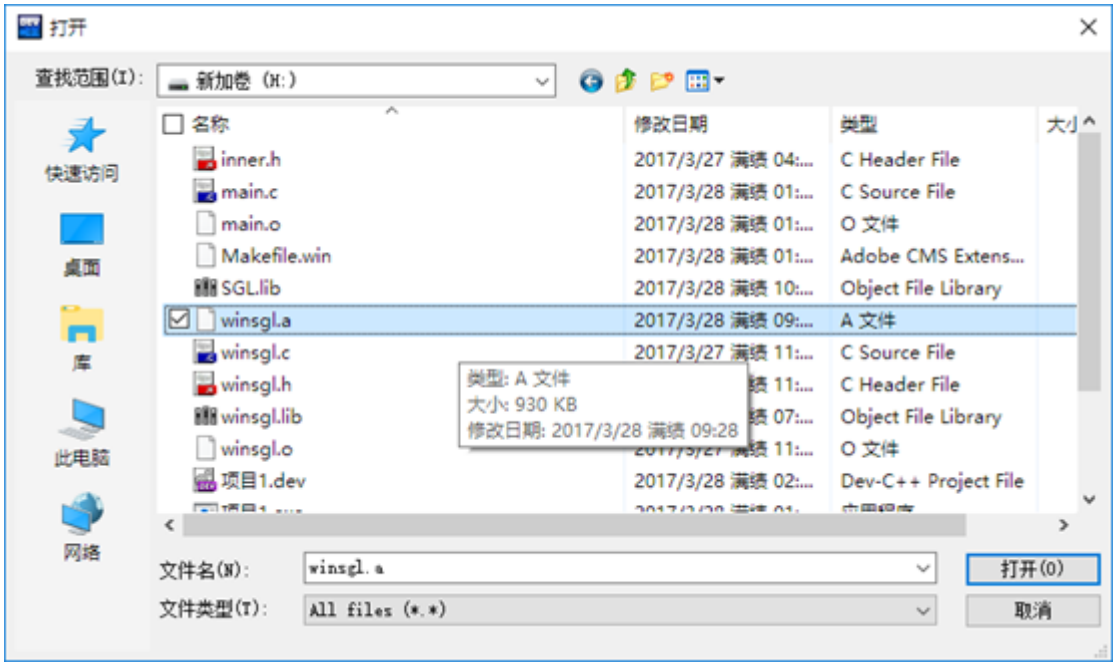
Then we choose Windows Application. It doesn't matter you choose C or C++. Then choose a directory to save your project.

Then clear the default project.

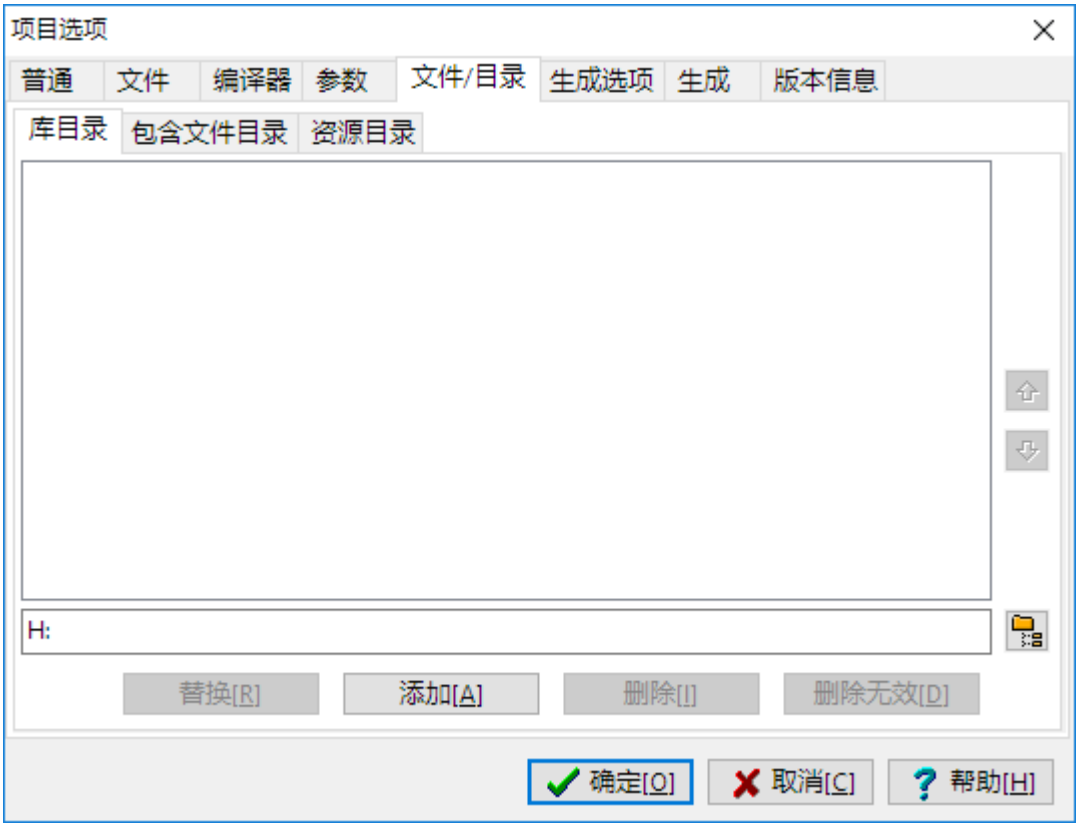


choose项目->项目属性->参数->加入库或对象 and select winsgl.a. The format of static library of mingw is .a not .lib.





Then choose 文件/目录 and input the folder which contains winsgl.h. Click add.



Here we finish the configuration. Just compile it and run, and we can enjoy our coding life.

Instruction

Basic Structure

In Sample->Initialize->empty.c

The **sgSetup()** function is to initialize the system and malloc some pointers. It will run only once in the beginning of the program. Remember not to draw anything in this function, or it will cause a crash.

The **sgLoop()** function is the main loop. It will run infinitely which means `while(1)sgLoop();` In the latest version, sgLoop() will definitely run one time per 10ms.

The **initWindow(int width, int height, char *title, int mode)** function is to initialize the window with its width, height and title. If parameter mode is BIT_MAP, we enter bit mode, or else if parameter mode is TEXT_MAP, we enter text mode. After v2.0.0, the parameter width should be a integer which can be divide by 8, or it will cause some strange bugs.

In Sample->Initialize->colorful.c

Use **setColor(int r, int g, int b)** to set one RGB color. Then use **clearScreen()** to fill the whole screen with the color set before.

Canvas

In Sample->Draw->pixel.c

The function **putPixel(int x, int y)** is to draw a point on the screen. The coordinate (0, 0) is the top-left point. And the coordinate(Screen->buffer->sizeX, Screen->buffer->sizeY) is the bottom-right point.

The function **getPixel(int x, int y)** is to get the color of the point (x, y). The return type RGB is defined in winsgl.h

In Sample->Draw->figure.c

The function **putQuad(int x1, int y1, int x2, int y2, int mode)** is to draw a rectangle with top-left point (x1, y1) and bottom-right point (x2, y2).

The function **putCircle(int x, int y, int r, int mode)** is to draw a circle with center (x, y) and radius r.

The function **putEllipse(int xc, int yc, int a, int b, int mode)** is to draw an oval with center (xc, yc) and semi axis a on direction x, b on direction y.

The parameter mode in both function can be set as SOLID_FILL or EMPTY_FILL. SOLID_FILL means to fill the whole figure with the set color while EMPTY_FILL means just draw the outline.

The function **putLine(int x1, int y1, int x2, int y2)** is to draw a line between(x1, y1) and (x2, y2).

The function **floodFill (int x, int y, RGB c)** is to fill the area connected to (x, y) it will stop at the points with color c.

In Sample->Draw->bmp.c

The function **loadBmp(int x, int y, char *filename)** is to load a bmp file named “filename” with its top-left at (x, y).

The function **setResizeable()** is to lock the size of one pixel which means that the image will not zoom when window size changed. It must be called in the sgSetup function.

The function **resizeFuntion(void (*func)(int x, int y))** is to set the callback function. Thus, when window size changed, the parameter func will be called.

The function **getImage (int left, int top, int right, int bottom, bitMap *bitmap)** is to copy an area of the screen. The points information is saved in bitmap. So it need to be declared and malloced by the users. Remember left must be no greater than right and as well as top and bottom. After get and put the Image, bitmap->data must be freed, or it will cause memory leak.

The function **putImage (int left, int top, bitMap *bitmap, int op)** is to paste the points saved in bitmap. The parameter op can be set as COPY_PUT, AND_PUT , OR_PUT, XOR_PUT and NOT_PUT. To learn more about it, please read Sample->Advance->mouse.c.

In Sample->Draw->string.c

The function **putString (const char *str, int x, int y)** is to put down a string with its left-top corner (x, y).

The function **putChar(char ch, int x, int y)** is similar to putSring function, but it uses the old font library so the character put by this function may be a little ugly.

The function **putNumber(int n, int x, int y, char lr)** is to put down a number string. If lr is ‘l’, then (x, y) is the left-top point of this string. Else if lr is ‘r’, then (x, y) is the right-top point of this string.

In Sample->Draw->function.c

The function **funcMap(int x1, int x2, int y1, int y2, float(*vect)(float x))** is to draw the curve of the given function “vect”. [x1, x2] is its domain and [y1, y2] is its codomain. The base point is the left-top corner of the window.

I/O System

In Sample->Bios->key.c

The function **initKey()** need to be added in the sgSetup() if other key functions will be used.

The function **bioskey(int cmd)** is the main key function. If cmd is 1, the return value is whether the key buffer is empty. Else if cmd is 0, the return value is the earliest key information. If the highest digit of the information is 1, it’s a key-up signal. If the highest digit of the information is 0, it’s a key-down signal. That is, when one key is pressed, a down-key will be send to the

key buffer, and then a up-key will be send to the key buffer. The low 8 digit are the ascii of each key. Remember never use biosKey(0) separately, it must appear in if(biosKey(1)){} branch.

The function **clearKeyBuffer()** is used to clear the key buffer.

In Sample->Bios->ascii.c

Compile and run it. We can see the exact key number after pressing and releasing each key. When pressing the key, the key number is its ascii code. But when releasing the key, the key number is defined by Microsoft Windows. Remember to close the Chinese input method when running the program.

In Sample->Bios->mouse.c

The function **initMouse()** is similar to initKey().

The function **mousePos()** returns the current mouse position.

The function **mouseStatus(int b)** is used to get each button's status. Parameter b can be set to SG_LEFT_BUTTON or SG_RIGHT_BUTTON or SG_MIDDLE_BUTTON. The return value is either SG_DOWN or SG_UP.

The function **biosMouse(int cmd)** is similar to biosKey(int cmd). But its return value is vectThree. The first two int is the coordinate of the click, and the third int is which button is clicked. The same requests as biosKey() function.

The function **clearMouseBuffer()** is similar to clearKeyBuffer().

In Sample->Advance->vect.c

This part is to emulate DOS interrupts.

The function **getVect(int intn)** returns the current interrupt function of intn. In DOS, int8 represent clock interrupt and int9 represent keyboard interrupt. If intn is 8, it returns the current clock interrupt function. And if intn is 9, it returns the current keyboard interrupt function.

The function **setVect(int intn, vect v)** is used to set a new interrupt function to intn.

The function **dosInt(int intn, int *ret)** is used to get the current key ascii when in keyboard interrupt function.

The function **setFreq(int f)** is used to set the frequency of clock interrupt. That is, every second the clock interrupt function will run f times.

In Sample->Bios->file.c

The function **selectFile(char name[], char start[], char format[], int idx)** is to use windows frame to choose a file. However, it just give the file name that user selected and put it in name[]. Parameter start is where to start search and format is the accepted file format. For example, format[] can be "All files\0.\0C/C++ file\0*.h;*.c;*.cpp\0\0". Note * can be any string so use it to represent file name and separate type name and type format array. The parameter idx is the default selection of the given format list.

The function **selectSave (char name[], char start[], char format[], char def[], int idx)** is to use windows frame to save a file. Remember that it just put the input file name into parameter name[] other than save any files. Parameter def is the format that append to the user input if the input does not have a suffix. Other parameters are similar to those in selectFile.

The function **selectDir (char name[], char start[])** is to use windows frame to select a directory. It puts the selected directory into parameter name[].

The function **debugf(const char *format, ...)** is like printf, but the output is not shown in the black rectangle, it is shown in the debug output window. In VS, this window usually stays at the bottom of the screen.

Time Control

In Sample->Time->time.c

The function **delay(int t)** is to wait for a while. Parameter t is the microseconds that will be waited. Take notes that when in delay function, the picture on the screen won't change. That is, do not put delay functions among drawing functions.

The function **delayBegin()** and **delayEnd(int t)** are used in pairs. The former set the start time and the latter set the end time. When the program run into delayEnd, if the time gap between delayBegin and delayEnd is less than t, the program will wait. Or else, don't wait.

The function **random(int n)** returns a number between 0 and n-1 randomly.

Winapi Tools

In Sample->Advance->mouse.c

The function **hideMouse()** is to hide the default mouse icon of Windows. In the latest version, this function needs to be added in sgLoop().

The function **showMouse()** is to show the default mouse icon of Windows. In the latest version, this function needs to be added in sgLoop().

In Sample->Advance->buffer.c

The function **setActivePage(int page)** is to set the active page. Parameter page can be either 0 or 1. Then all drawings will operate on this page.

The function **setVisualPage(int page)** is to set the visual page. Parameter page can be either 0 or 1. Then this page will be shown on the screen.

In Sample->Advance->clipboard.c

The function **copyText (const char *str)** is to copy str into system clipboard.

The function **pasteText ()** returns the content of the system clipboard. System clipboard is global which means it can transfer string between programs.

In Sample->Advance->clipboard.c

The function **setMouse (int x, int y)** is set the cursor position directly.

The function **getWidth (int obj)** is to get the width of the given obj. The parameter can be SG_WINDOW or SG_SCREEN while the size of screen never changes but the size of window can be changed by user.

In Sample->Advance->menu.c

The function **alertInfo(const char *info, const char *title, int mode)** is to show some message in a new small window. Parameter title is the title for this new small window and info is its content. The value of mode can be one of those in enum _alert.

The function **initMenu()** is to initialize the main menu list in the top of the window. The return value is the id of main menu list.

The function **addMenuItem(const char *title, int id, void(*func)())** is to add one item to the parent menu list with the given id. What will happen after the item is clicked is in the given function pointer func.

The function **addMenuList(const char *title, int id)** is to add one menu list to the parent menu list with the given id. The return value is the new list id.

The function **addMenuSeparator(int id)** is to add a separator in the menu list with id.

The function **addTray()** is to add an icon in the right-bottom of screen.

The function **hideToTray()** is to minimize the window and hide it to tray. If we click the icon in the tray, the window will appear again.

The function **restroeFromTray()** is the inverse steps of hide. It shows the window again.

The function **initTrayMenu ()** to initialize the tray menu list which will appear after right click the tray icon. The return value is the tray menu id.

The function **addTrayMenuItem(const char *title, int id, void(*func)())** is similar to addMenuItem.

The function **addTrayMenuList(const char *title, int id)** is similar to addMenuList.

The function **int addTrayMenuSeparator(int id)** is similar to addMenuSeparator.

In Sample->Advance->thread.c

The function **createThread (vect func)** is to execute the func in a new thread. The new thread and the main thread execute parallelly.

In Sample->Advance->timer.c

The function **timerThread(vect func, int millis, int time)** is to set a timer which execute func every millis of millisecond. After the given times, it stops automatically.

Internet

In Sample->Socket->client.c and server.c

The function **createServer(int port)** is to create a server with the given port. Different program on a single computer should have different ports. The return value is the server socket.

The function **createClient(const char *server, int port)** is to create a client and connect to server:port. For default, localhost is 127.0.0.1. The return value is the client server.

The function **acceptOne (SOCKET server)** is for server to wait for an connection. The return value is the connection socket. So far, there are three different sockets.

The function **socketSend(SOCKET s, char *buffer)** is to send data via a connection socked.

The function **socketReceive(SOCKET s, char *buffer, int len)** is to wait for the sending. The data received is put into buffer with max length len. If the connection is broken, the return value is SG_CONNECTION_FAILED, or else SG_NO_ERORR.

The function **closeSocket(SOCKET s)** is to close one connection socket after one connection failure or a compulsory close.

Text Mode

In Sample->Write->hello.c

The function **setBfc(int bgc, int fgc)** is similar to setColor. In this function, bgc stands for background color, fgc stands for foreground color. These two parameters can be set from 0 to 15, each of which stands for one color in the enum _color branch.

The function **clearText()** is similar to clearScreen. In this function, all chars on the screen will be set to '\0', and their color is the one that set in setBfc.

The function **writeChar**(char c, int x, int y) is to put one character at position (x, y) with the color set in setBfc.

In Sample->Write->color.c

The function **writeString(char *s, int x, int y)** is similar to putString. This time, the string can change its line automatically.

The function **setCharFgc(char color, int x, int y)** is to change the foreground color of the char in position (x, y). The parameter color can be set from 0 to 15, each of which stands for one color in the enum _color branch.

The function **setCharBgc(char color, int x, int y)** is to change the background color of the char in position (x, y). The parameter color can be set from 0 to 15, each of which stands for one color in the enum _color branch.

The function **setCharColor(char color, int x, int y)** is to change the background and foreground color of the char in position (x, y). The parameter color can be set from 0 to 255, the high 4 bit is the background color and the low 4 bit is the foreground color.

In Sample->Write->move.c

The function **getText(int left, int top, int right, int bottom, textMap *text)** is similar to getImage. And the notes are same.

The function **putText(int left, int top, textMap *text)** is similar to putImage. And the notes are same.

Widgets

The widget struct is defined as below.

```
typedef struct _w{
    enum _control type;

    vec2 pos;
    vec2 size;

    int style;
    int status;
    int visible;
    int priority;
    int valid;

    int hide;
    int value;
    SGstring name;
    void *content;
    struct _w *associate;
    struct _w *next;
    struct _w *child;

    RGB bgColor, passColor, pressColor, fgColor;
    bitMap bgImg;

    mouseMoveCall mouseIn, mouseOut;
    mouseClickCall mouseDown, mouseUp;
    mouseClickUser mouseUser;
    keyPressCall keyDown, keyUp;
    keyPressUser keyUser;

}widgetObj;
```

enum _control type is the type of this widget. It will be initialized when newWidget() works, so do not assign for this element.

vecTwo pos is the coordinate of the left-top corner.

vecTwo size contains the width and height of this widget.

int style is the style of its appearance. Now SGL only supports SG_DESIGN.

int status is to record whether it is passed or not , whether it is pressed or not, whether it is selected or and whether it is focused or not. Programmers should not change its value, too.

int visible is to control whether this widget is active or not. Programmer should not touch its value. Use showWidget() and ceaseWidget() to change its value conveniently.

int priority is used by the system. Programmers should not change its value.

int valid is whether this widget need to redraw or not. Set this value to non-zero and the widget will be refreshed.

int hide has different meanings in different type of widgets.

int value has different meanings in different type of widgets.

SGstring name is the name of this widget. We sometimes use getWidgetByName so the name should be exclusive which means that different widgets must not have the same name.

SGstring content is the string contains the message of this widget. It has different meanings in different type of widgets, too.

struct _w *associate points to another widget which associate with this widget. Association between different types of widget has different meanings.

struct _w *child is used when type is SG_COMBINED, this pointer points to the first child widget. All child widgets are linked as a list.

struct _w *next is used when this widget is a child widget, this pointer points to the brother widget. The list will be built by system so programmers needn't to concern about this.

The four colors can be assigned by programmers now. bgColor is the background color when no events happen. passColor is the background color when mouse passing above and pressColor is the background color when mouse pressed on it. fgColor is usually the text color.

bitMap bgImg is the background image of this widget.

mouseUser and keyUser is the functions to execute when mouse clicked and when key pressed.

The function **newWidget(int type, SGstring name)** is to get a widget pointer with default values. Parameter type is one in enum _control defined in winsgl.h and name is the unique identifier of this widget.

The function **newCombinedWidget(int num, SGstring name, ...)** is to get a combined widget pointer. The first parameter num is the total sub-widgets it will have so the ... should be replaced by widgets of this num.

The function **registerWidget(widgetObj *obj)** is to tell the system that obj has been fully prepared and should be shown on the screen. Then the program will copy the block which pointed by obj. Remember that the widget pointer in the system does not equal to the parameter obj.

The function **getWidgetByName(char *name)** returns the widget pointer with the identifier name. In this way we can change the attributes of each widget. Thus when registering, name should be unique.

The function **showWidget(char *name)** is to set the visibility to true.

The function **ceaseWidget(char *name)** is to set the visibility to false.

The function **deleteWidgetByName(char *name)** is to remove the widget with identifier name for ever.

The function **moveWidgetByName(char *name)** is to change the position of a widget. The moving operation is not simply change the pos element, so use this function instead of changing the pos value directly.

For more callback function information, please refer to the instruction video.

Now SGL has finished many types of widgets.

SG_BUTTON is a button for users to click. It's a rectangle by default. When clicked, those mouseClickUser functions will execute. Programmers just need to write another function to respond.

SG_INPUT is an input line for users to input a string. It's a rectangle by default. The content of the struct is the string that the user input.

SG_DIALOG is to show some message on the screen, like the messageBox in WINAPI. A small rectangular window will appear when showWidget() executes. The content of the struct is the string that shown in it. Programmer just need to decide when to call showWidget() is enough. When clicked the crossing icon, the widget will be ceased automatically.

SG_OUTPUT is a rectangle to show some message. Again the content is the string that it shows.

SG_LIST is a dropdown list for users to select one. The content is the names of the items separated by '\0'. So when initializing, we need to use memcpy not strcpy. The value of the struct is the index of the chosen item.

SG_LABEL is a one-line string. The content of the struct is what it showed.

SG_CHECK gets a Boolean value whether the user chose this item or not. The value of the struct shows if it is selected. The content of the struct will be shown on the right of the circle.

SG_PROCESS is a process bar. The value of it is the percent shown on the screen. Unlike SG_DIALOG, it won't be ceased automatically. The content of the struct is similar to the one in SG_DIALOG.

SG_OPTION is a list appears when click the right button of the mouse. If the mouse position is in its associate widget, the list will appear. Like SG_LIST, the names of the items should be separated by '\0' in the content of the struct.

SG_DRAG is a bar to adjust values. The max value is 100 so it can be also seen as percentage. Drag its button to change the value and get it through code.

SG_SCROLLVERT is a scroll bar which usually set on right and bottom. The attribute hide is the range for scroll and value is the temporary position. Note that value is set between 0 and hide-1.

SG_COMBINED is a widget linked list. The link pointer is attribute next. The pos of the root can be concerned as the base coordinate.

Sub Windows

In Sample->Advance->window.c

The function **createWindow(int width, int height, const char *title, vect setup, vect loop)** is used to pop up a new window. Set its width and height and title. Then two functions setup and loop are similar to sgSetup and sgLoop, a difference in setup is that there isn't initWindow in it. The return value is the window id. This value should be saved for further painting.

The function **closeWindow(int id)** is used to close the window with given id.

The function **startSubWindow(int id)** tells the system that instructions below manipulate the sub window with the given id. Only the sub window need to call this function explicitly. If the main window is being drawn, nothing need to add.

The function **endSubWindow()** tells the system the code has finished sub window drawing. Remember that there's a lock between startSubWindow and endSubWindow. That means, when one window is drawing, other window cannot break this process. They need to wait until the temporary window finish its drawing. So there are two rules to obey. One is that no more startSubWindow and endSubWindow should be add between one startSubWindow and endSubWindow. The other is that never draw sub windows in sgLoop. Try to set global values and draw in the loop of those sub windows' own.

Data Operation

In Sample->Data->json.c

The function **createJson()** is to create an empty json object.

The function **createJsonArray()** is to create an empty json array.

The function **freeJson(struct JSON *json)** is to free the memory of the given json.

The function **readJson(const char *json)** is to build a json object or json array with the given string.

The function **writeJson(struct JSON *json))** is to give the string of the given json.

The function **getContent(struct JSON *json, const char *name)** is to get one item from the given json object with its name.

The function **getElement(struct JSON *json, int idx)** is to get one item from the given json array with its index.

The function **deleteContent(struct JSON *json, const char *name)** is to delete the item from the given json object with its name.

The function **deleteElement(struct JSON *json, int idx)** is to delete the item from the given json array with its index.

The function **setContent(struct JSON *json, const char *name, type value)** is to modify the given json. If the item is found in it, then change its value. Or else, no such element found, build one and insert to it.

The function **setElement(struct JSON *json, int idx, type value)** is to modify the given json. If the index is in the element range, then change its value. Or else, build one and insert to the first when index is smaller than zero or insert to the last when index is greater than the size of json array..