ExcelExporter Instructions

Catalog

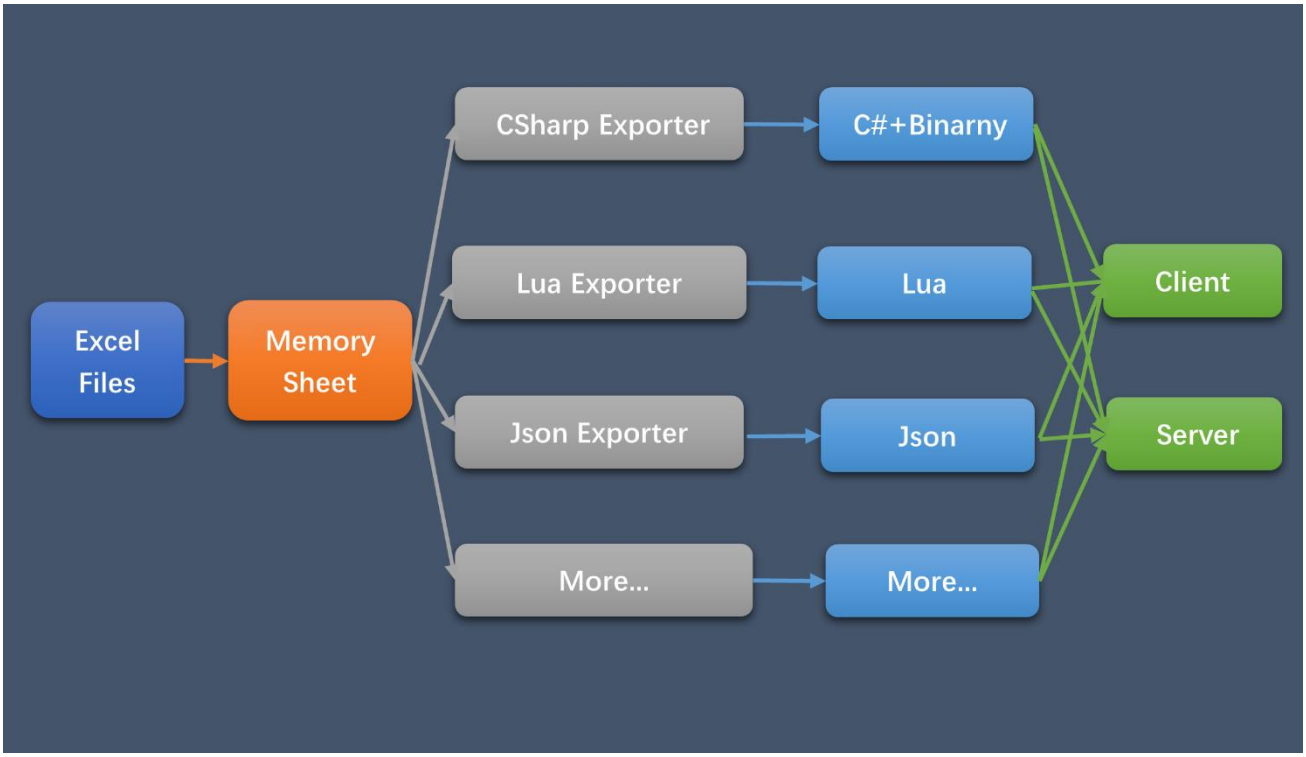
1. Introdution

ExcelExporter is a tool developed to solve practical project applications. In the past project development experience, the Excel file data export tools used generally have problems such as messy code, difficult to maintain, poor cross-platform support (generally not supporting macOS), and difficulty in exporting multiple programming languages. *ExcelExporter* was developed to solve these problems.

ExcelExporter supports Unity Editor and App usage for both macOS and Windows. It currently supports exporting code and data in C#+ binary files, Json, and Lua formats. Thanks to its good plug-in design, you can easily add new export formats without worrying about affecting other parts of the project. Support *byte, int, uint, long, ulong, float, double, string* and other data types and *arrays* with them as elements, namely int[], float[], string[], etc. *ExcelExporter* also supports a data type called *localizedstring*, which is used to support multi-language localization of text.

You can use *ExcelExporter* as an independent project to publish App, or you can extract functional modules, integrate them into your own projects, and use them in the Unity Editor.

The design of ExcelExporter is shown in the figure:



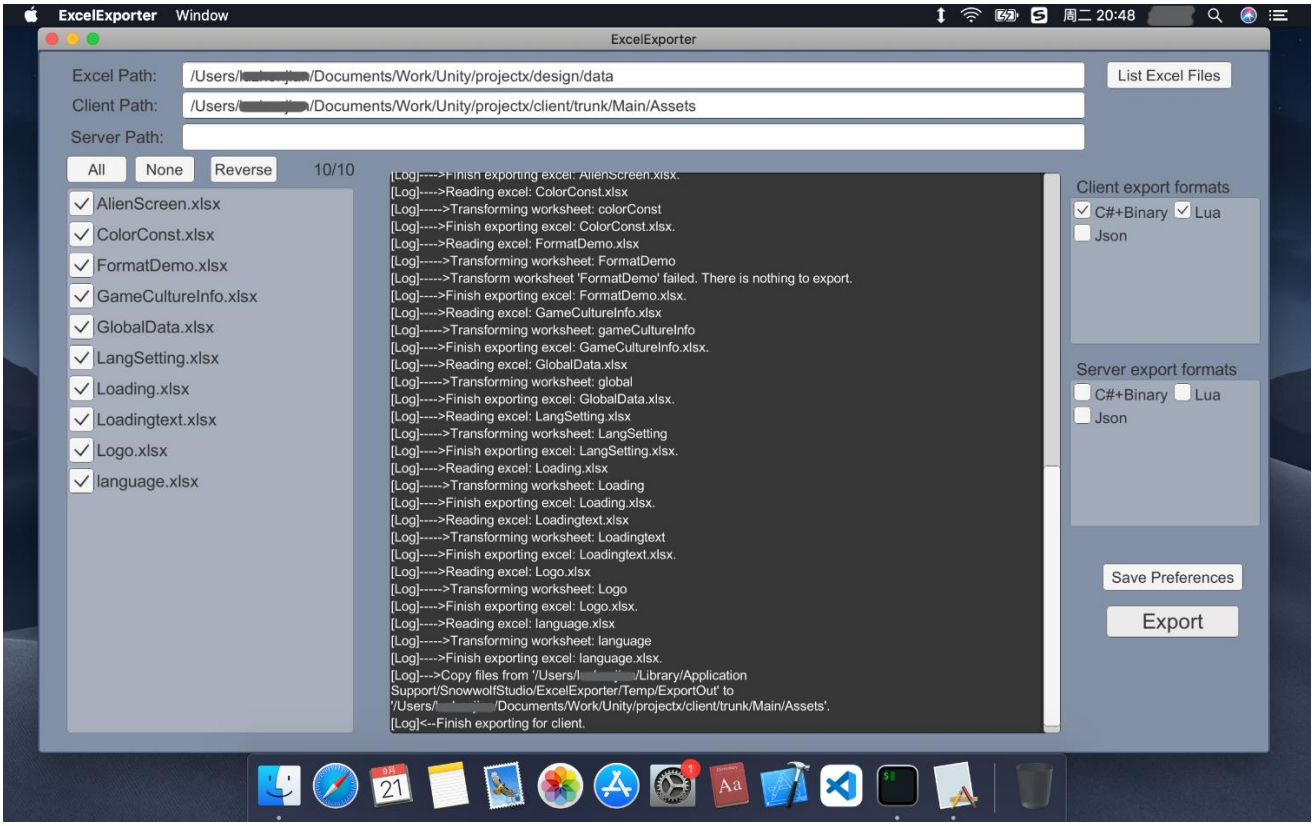
2. Supported Platforms

ExcelExporter is developed with Unity, and natively supports the release of App for cross-platform use. The Plugins and code used in this project conform to the *.NET Standard 2.0* cross-platform standard used by Unity.

2.1-Unity Editor（Windows + macOS）

The *ExcelExporter* code supports use in the Unity Editor of Windows and macOS, and macOS users can also support exporting Excel data, without VBA.

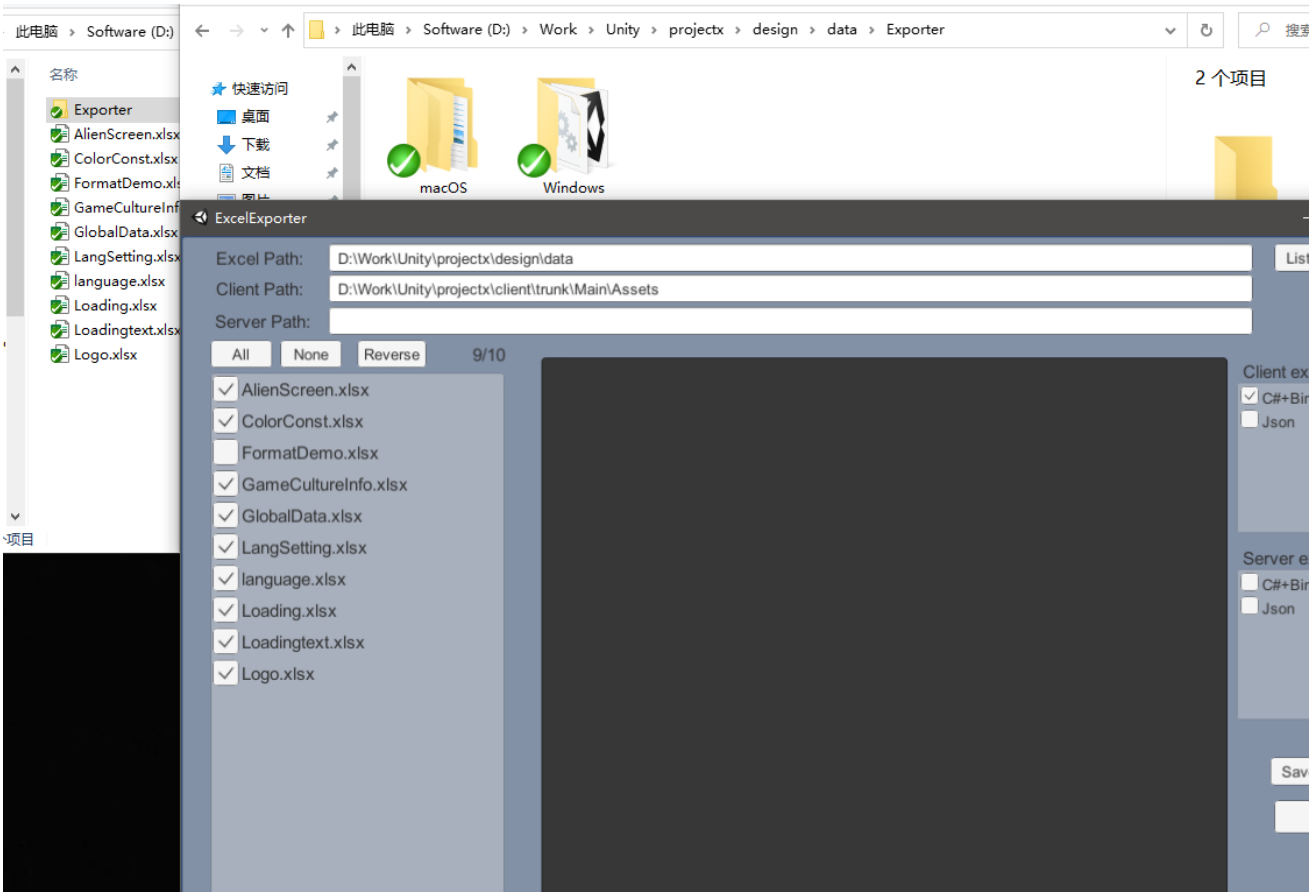
2.2-macOS Application



P2-1 macOS Application Example

ExcelExporter supports publishing as a macOS application. Note that Unity's **PlayerSettings->Api Compatibility Level** should not be set to .NET 3.5, it has been deprecated in Unity 2018 and above.

2.3-Windows Application



P2-2 Windows Application and folder structure

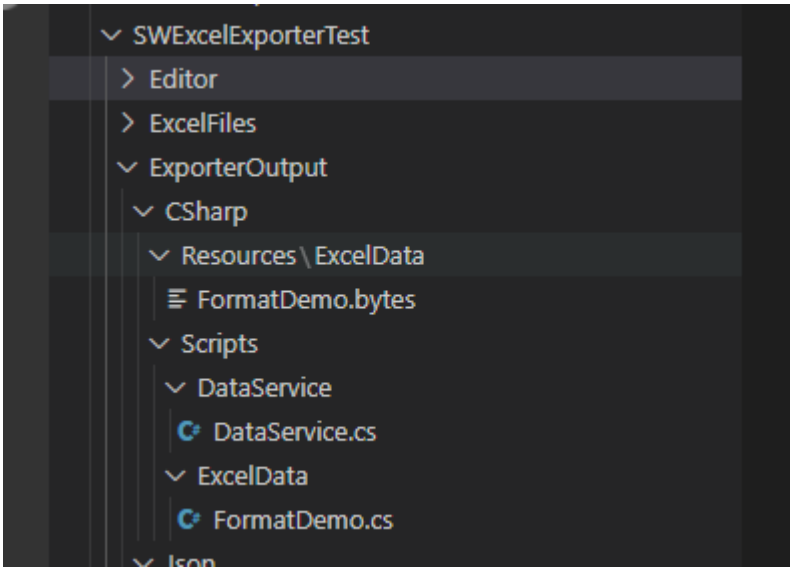
ExcelExporter supports publishing as a Windows application. Note that Unity's **PlayerSettings->Api Compatibility Level** should not be set to .NET 3.5, it has been deprecated in Unity 2018 and above.

3. Data Export Formats

3.1-C#+Binary

The format of this export format is to parse the data structure of the Excel table into a C# class, and the data is written into a binary file. In this way, when changing the Excel table structure, the C# code will not change, only the binary file will be updated, which will reduce the amount of code and compile time, especially when using IL2CPP. Binary files are pure data and have no redundant information, which greatly reduces the data size and increases the reading speed.

The application example is shown in **SWExcelExporterTest/ExporterOutput/CSharp** of the project. as the picture shows:



P3-1 C#+Binary Files Structure

```
ssets > SWExcelExporterTest > ExporterOutput > CSharp > Scripts > ExcelData > FormatDemo.cs > {} ExcelData > ExcelData.FormatDemo > RefreshLocalizationValues()
        0 references
        public static Item GetItem(string key)
        {
            Instance.m_Items.TryGetValue(key, out Item foundItem);
            #if UNITY_EDITOR
            if (foundItem == null)
            {
                UnityEngine.Debug.LogWarningFormat("{0} do not contains item of key '{1}'.", Instance.sheetName, key);
            }
            #endif
            return foundItem;
        }

        0 references
        public static IEnumerable<KeyValuePair<string, Item>> GetDict()
        {
            return Instance.m_Items;
        }

        4 references
        private Dictionary<string, Item> m_Items = new Dictionary<string, Item>();

        4 references
        public string sheetName => "FormatDemo";

        1 reference
        private void Init()
        {
            byte[] bytes = DataService.GetSheetBytes(sheetName);
            using (MemoryStream ms = new MemoryStream(bytes))
            {
                using(BinaryReader reader = new BinaryReader(ms))
                {
                    reader.ReadString(); //sheetName

                    //Read header
                    SheetHeader sheetHeader = new SheetHeader();
                    sheetHeader.ReadFrom(reader);
                    List<SheetHeader.Item> headerItems = sheetHeader.items;
```

P3-2 C# file FormatDemo.cs

```
FormatDemo.bytes.hexdump ×
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000: 0A 46 6F 72 6D 61 74 44 65 6D 6F 0E 00 00 00 03    .FormatDemo.....
00000010: 6B 65 79 06 73 74 72 69 6E 67 06 69 6E 74 56 61    key.string.intVa
00000020: 6C 03 69 6E 74 07 75 69 6E 74 56 61 6C 04 75 69    l.int.uintVal.ui
00000030: 6E 74 07 6C 6F 6E 67 56 61 6C 04 6C 6F 6E 67 08    nt.longVal.long.
00000040: 75 6C 6F 6E 67 56 61 6C 05 75 6C 6F 6E 67 07 62    ulongVal.ulong.b
00000050: 79 74 65 56 61 6C 04 62 79 74 65 08 66 6C 6F 61    yteVal.byte.floa
00000060: 74 56 61 6C 05 66 6C 6F 61 74 09 64 6F 75 62 6C    tVal.float.doubl
00000070: 65 56 61 6C 06 64 6F 75 62 6C 65 06 73 74 72 56    eVal.double.strV
00000080: 61 6C 06 73 74 72 69 6E 67 07 6C 73 74 72 56 61    al.string.lstrVa
00000090: 6C 0F 6C 6F 63 61 6C 69 7A 65 64 73 74 72 69 6E    l.localizedstrin
000000a0: 67 08 69 6E 74 41 72 72 61 79 05 69 6E 74 5B 5D    g.intArray.int[]
000000b0: 0A 66 6C 6F 61 74 41 72 72 61 79 07 66 6C 6F 61    .floatArray.floa
000000c0: 74 5B 5D 0B 73 74 72 69 6E 67 41 72 72 61 79 08    t[].stringArray.
000000d0: 73 74 72 69 6E 67 5B 5D 0D 63 6C 69 65 6E 74 4F    string[].clientO
000000e0: 6E 6C 79 56 61 6C 05 66 6C 6F 61 74 06 00 00 00    nlyVal.float....
000000f0: 04 6B 65 79 31 0C 00 00 00 7B 00 00 00 C0 1D FE    .key1....{...@.~
```

P3-3 Binary file FormatDemo.bytes

3.2-Json

From the **SWExcelExporterTest/ExporterOutput/Json** directory of the project, you can find the export example of **FormatDemo.json**. As shown below:

FormatDemo.json

Assets > SWExcelExporterTest > ExporterOutput > Json > FormatDemo.json > ...

```
    },
    "key1"        : {
        "key" : "key1",
        "intVal" : 12,
        "uintVal" : 123,
        "longVal" : -123456,
        "ulongVal" : 13212321,
        "byteVal"  : 1,
        "floatVal" : 123.456001281738,
        "doubleVal" : 567.891,
        "strVal"    : "Normal string.",
        "__lKey_lstrVal" : "Client_1000",
        "intArray"       : [
            1,
            2,
            3
        ],
        "floatArray"     : [
            1.5,
            2.29999995231628,
            4.5
        ],
        "stringArray"    : [
            "Hello world",
            "\" Hello world 2\""
        ],
        "clientOnlyVal"  : 123.456001281738
    },
    "key2"        : {
        "key" : "key2",
        "intVal" : 13,
```

P3-3 Json file FormatDemo.json

3.3-Lua

From the **SWExcelExporterTest/ExporterOutput/Lua** directory of the project, you can find the export example of **FormatDemo.lua**. As shown below.

FormatDemo.lua ×

Assets > SWExcelExporterTest > ExporterOutput > Lua > Lua > ExcelData > FormatDemo.lua > ...

```lua
module("ExcelData")
local fileName = "FormatDemo"
local GetLocalizedText = DataService and DataService.GetLocalizedText

FormatDemo={
    --["header"]={key="string",intVal="int",uintVal="uint",longVal="long",ulor
    ["key1"]={key="key1",intVal=12,uintVal=123,longVal=-123456,ulongVal=132123
    ["key2"]={key="key2",intVal=13,uintVal=124,longVal=133133133,ulongVal=1232
    ["key3"]={key="key3",intVal=14,uintVal=125,longVal=434324,ulongVal=1232132
    ["key4"]={key="key4",intVal=15,uintVal=126,longVal=13214,ulongVal=12321321
    ["key5"]={key="key5",intVal=16,uintVal=127,longVal=143,ulongVal=12321321,b
    ["key6"]={key="key6",intVal=17,uintVal=128,longVal=1213,ulongVal=12321321,
}

function InitLocalizedTexts_FormatDemo()
    if not GetLocalizedText then return end
    for k,v in pairs(FormatDemo) do
        v.lstrVal = GetLocalizedText(fileName, v.__lKey_lstrVal or "")
    end
end

InitLocalizedTexts_FormatDemo()

return FormatDemo
```

P3-4 Lua file FormatDemo.lua

3.4-More

In addition to the above format, you can also implement the format you need by extending the *Exporter* base class.

4. Supported Data Types

4.1-Excel Form Formats

The Excel format used by the *ExcelExporter* can be found in **SWExcelExporterTest/ExcelFiles/FormatDemo.xlsx**. *ExcelExporter* supports the export format of multiple Excel worksheets in one Excel file. Therefore, the exported file name is name of the Excel worksheet instead of Excel file name.

P3 | //"[ClientOnly]" marks indicate this column will only be exported to client.

	A	B	C	D	E	F	G	H	I	J	K
1											
2	//Use "#------>" to start and "#<------" to stop payload.										
3	#------>							//use (//) to comment this column			
4	key	intVal	uintVal	longVal	ulongVal	byteV	floatVal	//unuse	doubleV	strVal	lstrVal
5	string	int	uint	long	ulong	byte	float		double	string	localizedstrin
6	key1	12	123	-123456	13212321	1	123.456	nothing	567.89	Normal string.	Client_1000
7	key2	13	124	133,133,133	12321	0	124.456	nothing	568.89	""String with quotation mark""	Client_1001
8	key3	14	125	434324	123213213	1	125.456	nothing	569.89	String with space to trim.	Client_1002
9	key4	15	126	13214	123213213	0	126.456	nothing	570.89	" String with space to use."	Dialog_1003
10	//use (//) to comment this row							nothing			
11	key5	16	127	143	12321321	1	127.456	nothing	571.89	"Line with explicit linefeed\n and implicit linefeed"	Dialog_1004
12	key6	17	128	1213	12321321	0	128.456	nothing	572.89	你好&こんにちは&안녕하세요--CJK	Dialog_1005
13											
14											
15											
16											
17											
18											
19											
20											
21											
22	#<------										
23											
24											
25											

P4-1 FormatDemo.xlsx Example 1

e exported to client.

L	M	N	O	P
//numeric array use (,) as separator		//string array use (|) as separator	//"[ServerOnly]" marks indica	//"[ClientOnly]" marks indicate t
intArray	floatArray	stringArray	[ServerOnly]serverOnlyVal	[ClientOnly]clientOnlyVal
int[]	float[]	string[]	int	float
1, 2, 3	1.5, 2.3, 4.5	"Hello world" | "" Hello world 2""	12	123.456
0, 7,	0.5	Hello | world | Again	13	124.456
2, 1010, 333	1.5, 2.1	Hi, My name is Jason.| 你好， 杰森。	14	125.456
444, 1221, 0	3.1, 4.4, 2.2	"Hello World" | "Hello \n World"	15	126.456
11	9.9, 22, 34	Hello world	16	127.456
123, 12321, 123213, 414	16, 17, 19	|	17	128.456

P4-2 FormatDemo.xlsx Example 2

4.2-Primitive Types

ExcelExporter supports commonly used primitive types, and currently supports *byte, int, uint, long, ulong, float, double, string* and other primitive types. You can also extend your own types in *CellValueType*.

4.3-Array

ExcelExporter supports the primitive types listed in Chapter 4.2 as elements of array. Note that in the Excel table structure, *numeric arrays use commas (,) as the separator, and string arrays use (|) as the separator*. This is because the character the comma (,) is very common in the string, and using (,) as the separator will cause a lot of unnecessary trouble.

4.4-localizedstring

ExcelExporter defines a string type called *localizedstring*, which is actually a string type, which represents the Key of the localized string.

For example, if you use a localized string in your program, you can follow the procedure below to perform the operation:

1. In the Excel sheet, use the string type to represent the key of the localized string, for example, the name is *lstringKey*.

2. At runtime, read the *lstringKey* that from the Excel table, and then write GetLocalizedText(xx.lstringKey) wherever it is called.

The *localizedstring* type simplifies this process. For users, the key is filled in the Excel sheet, and then, the string has been localized implicitly when was read at runtime. The following figures show the generation process of the *localizedstring* named *lstrVal.*

```
2 references
public string __lKey_lstrVal;
1 reference
public string lstrVal;
```

P4-3 Define implicit key **__lkey_lstrVal**

```
else if (j == lstrValIndex)
{
    newItem.__lKey_lstrVal = reader.ReadString();
}
```

P4-4 Read string data from Excel to **__lkey_lstrVal**

```
public void RefreshLocalizationValues()
{
    foreach(var kv in m_Items)
    {
        Item item = kv.Value;
        item.lstrVal = DataService.GetLocalizedText(sheetName, item.__lKey_lstrVal);
    }
}
```

P4-5 Use **__lkey_lstrVal** to obtain localized string for **lstrVal**

5、App Workflow

The process of App operation can be operated in the order of 5.1-5.5.

5.1-Choose Paths

P5-1 Choose Paths

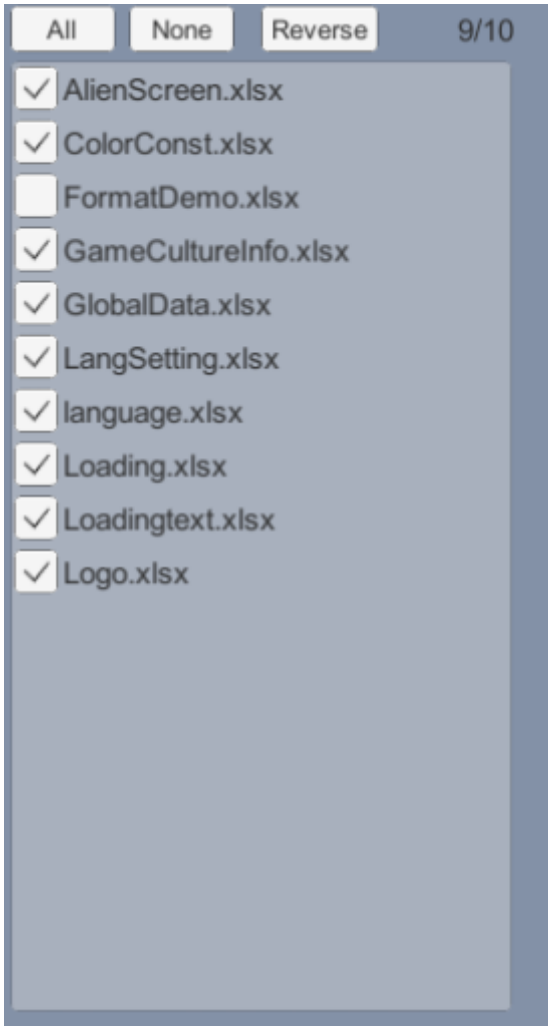
Excel Path represents the directory where Excel documents are stored. All Excel files (only .xlsx) in this directory will be listed.

Client Path represents the output path of the client, generally the client project path, blank means that the client file will not be exported.

Server Path represents the output path of the server, and empty means that the server file will not be exported.

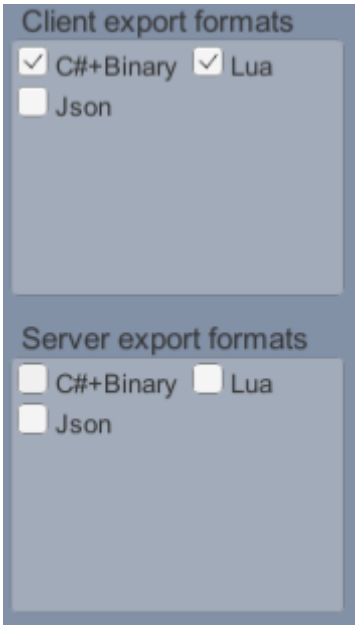
Note: After entering the **Excel Path**, the program will usually list the Excel files automatically. If not, you can press the **List Excel Files** button to list the Excel files yourself.

5.2-Choose Excel Files To Process



P5-2 Choose files to export

5.3-Choose Data Export Formats



P5-2 Choose data export formats for server and client

5.4-Save Preferences/Configuration

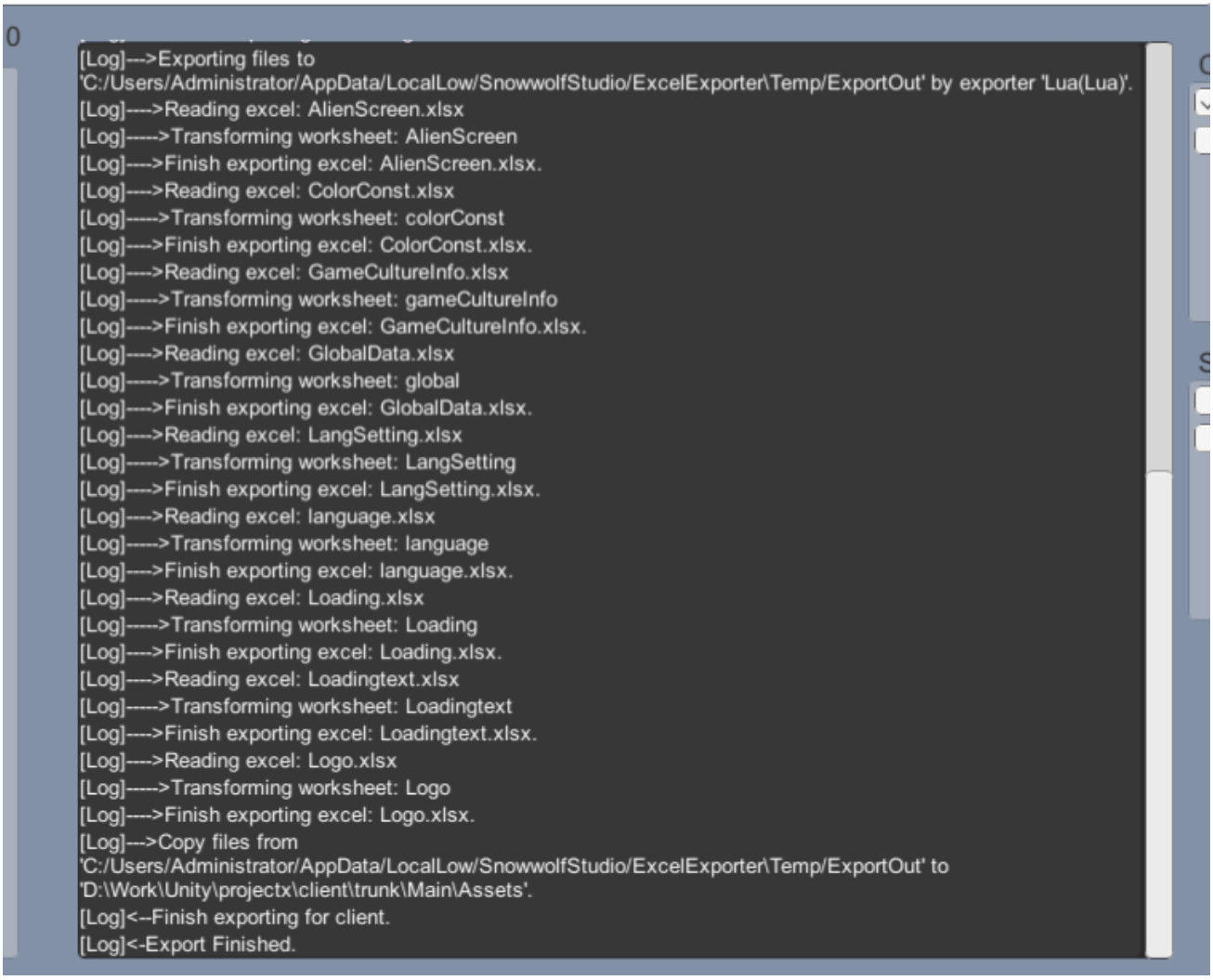
The function parameters mentioned in 5.1~5.3 can be saved as preferences, just press the **Save Preferences button**. In addition, if you press the **Export button** to export the file, the above preferences will also be automatically saved.

5.5-Do Export

P5-3 Save Preferences Button and Export Button

Use the **Export button** to export the file. The output file will be found in the directory specified in the above step.

5.6-Realtime Log View



P5-4 Realtime Log View

You can view the print logs of the export process, and you can view information such as exceptions, errors, warnings, etc., to facilitate tracking of problems.