**Instructions: how to add functions from C-files to Datagrok package**

*Prerequisites:* [Datagrok](https://datagrok.ai/help/develop/set-up-environment) and [Emscripten](https://emscripten.org/index.html) tools must be installed.

Suppose, there are two C-files lib1.c and lib2.c with functions that should be included to the package:

// lib1.c

int mySqr(int n){

return n \* n;

}

int tripleProduct(int a, int b, int c) {

return a \* b \* c;

}

int someFunction(){

return 2022;

}

// lib2.c

int myProd(int n, int m){

return n \* m;

}

float myCube(float n) {

return n \* n \* n;

}

Suppose, the functions mySqr, tripleProduct, myProd should be exported to the package.

**Instructions:**

1. Create Datagrok package: grok create MyNewPackage --js --jest and run in the folder MyNewPackage the following: npm install.

2. Create the folder vendor in the folder MyNewPackage.

3. Copy lib1.c and lib2.c files to the folder vendor

4. Prepare C-functions for export:

1. open the file lib1.c and add the following:

* #include <emscripten.h> at the beginning of the file;
* EMSCRIPTEN\_KEEPALIVE before each function that should be exported

1. repeat the same instructions with the file lib2.c.

You get the following:

// lib1.c

#include <emscripten.h> ← this is added

EMSCRIPTEN\_KEEPALIVE ← this is added

int mySqr(int n){ ← this functions is exported

return n \* n;

}

EMSCRIPTEN\_KEEPALIVE ← this is added

int tripleProduct(int a, int b, int c) { ← this functions is exported

return a \* b \* c;

}

int someFunction(){ ← this functions is not exported

return 2022;

}

// lib2.c

#include <emscripten.h> ← this is added

int myProd(int n, int m){ ← this functions is not exported

return n \* m;

}

EMSCRIPTEN\_KEEPALIVE ← this is added

int myCube(int n) { ← this functions is exported

return n \* n \* n;

}

5. Copy the file [module.json](https://drive.google.com/file/d/1YHFQbdaPnMPWDejumHuALj0KouHrTZyJ/view?usp=sharing) to the folder vendor. This file contains export settings:

{

"name": "", ← name of the C-library exported

"version": "0.0.1",← version of the C-library exported

"description": "", ← description of the C-library exported

"folder": "vendor", ← name of the folder with C-files

"source": [], ← list of C-files names

"nameOfLibFile": "", ← name of JS-file, which will be created by Emscripten tool

"exportName": "", ← name, which will be used in JS-file, which will be created by Emscripten tool

"moduleName": "",← name, which will be used in the file package.js

"optimizationMode": "-O0",← optimization mode

"nameOfFileForFunctionsData": "functionsData.json", ← name of file with exported functions descriptors, it will be created automatically

"packageFile": "..\\src\\package.js", ← package file: exported C-functions will be added to this file

"packageJsonFile": "..\\package.json", ← package file

"fileWithEmscriptenCommand": "command.txt" ← name of file with Emscripten command, it will be created automatically

}

REMARKS.

1. the items, which are marked with blue color, or blue-items should be filled (in the next section, we show an example);
2. green-items can be modified, but it is optional;
3. red-items should NOT be modified.

6. Modify the file module.json (actually, we work with blue-items):

{

"name": "TestExampleName",

"version": "0.0.1",

"description": "Some description",

"folder": "vendor",

"source": ["lib1.c","lib2.c"], ← requires correct names !!!

"nameOfLibFile": "newLib.js",

"exportName": "newLibExportName",

"moduleName": "NewModule",

"optimizationMode": "-O0",

"nameOfFileForFunctionsData": "functionsData.json",

"packageFile": "..\\src\\package.js",

"packageJsonFile": "..\\package.json",

"fileWithEmscriptenCommand": "command.txt"

}

REMARKS.

1. the text marked with green color is added;
2. data in the field "source" is of particular importance - correct names must be added;
3. further, in the file package.js a name from the field "moduleName" is used.

7. Copy the script [export.py](https://drive.google.com/file/d/15JryG3grXN-_WwlxkdrS9nLLIw9mgfE9/view?usp=sharing) to the folder vendor.

8. Execute the script export.py. It performs ALL the routines concerning export of C-files to Datagrok package. The following is obtained:

1. C-files, which are added to “source” of module.json, have been parsed, and descriptors of exported functions have been obtained (this data can be found in the file functionsData.json in the folder vendor);

The file functionsData.json contains something like the following:

{

"lib1.c": {

"mySqr": {

"type": "int",

"arguments": [

[

"int",

"n"

]

]

},

"tripleProduct": {

"type": "int",

"arguments": [

[

"int",

"a"

],

[

"int",

"b"

],

[

"int",

"c"

]

]

}

},

"lib2.c": {

"myCube": {

"type": "int",

"arguments": [

[

"int",

"n"

]

]

}

}

}

One can see the exported functions descriptors: source C-file, name, return type, type and name of all arguments.

This file can be used in order to check functions data. Note that just functions annotated with EMSCRIPTEN\_KEEPALIVE are exported.

1. Text file command.txt that contains Emscripten command. Currently, it is

emcc -O0 lib1.c lib2.c -o newLib.js -s WASM=1 -s ALLOW\_MEMORY\_GROWTH=1 -s MODULARIZE=1 -s EXPORT\_NAME="newLibExportName" -s EXPORTED\_FUNCTIONS=["\_mySqr","\_tripleProduct","\_myCube"] -s EXPORTED\_RUNTIME\_METHODS=["cwrap"]

The script export.py executes this command automatically. So, you do NOT need to execute it!

1. The file newLib.wasm that contains C-functions compiled by Emscripten. Note that the name “newLib” coincides with that one, which is specified in the field "nameOfLibFile" of the file module.json with export settings.
2. The file newLib.js created by Emscripten. Note that at the end of this file there is the following:

...

var NewModule = undefined;

async function initNewModule() {

if (NewModule === undefined) {

console.log("Wasm not Loaded, Loading");

NewModule = await newLibExportName();

} else {

console.log("Wasm Loaded, Passing");

}

}

These lines are added by the script export.py. The main part of newLib.js is generated by Emscripten.

1. In the file package.json, one can find the following:

"sources": [

"vendor/newLib.js"

]

This is added automatically.

1. Optionally, one can test the current result as follows:

* create in the folder vendor test file test.js:

// test.js

var factory = require('./newLib');

factory().then((instance) => {

// 1-st function from lib1.c

let n = 13;

console.log(n + "^2 = " + instance.\_mySqr(n));

// 2-st function from lib1.c

let x = 4;

let y = 8;

let z = 16;

console.log(x + " \* " + y + " \* " + z + " = " + instance.\_tripleProduct(x, y, z));

// 1-st function from lib2.c

a = 2;

console.log(a + "^3 = " + instance.\_myCube(a));

});

* run it using Node: node test.js
* the result is

13^2 = 169

4 \* 8 \* 16 = 512

2^3 = 8

1. The file package.js is appended with the following:

// C-FUNCTIONS

// Functions from lib1.c

//name: mySqr

//input: int n

//output: int result

export async function mySqr(n) {

await initNewModule();

return NewModule.\_mySqr(n);

}

//name: tripleProduct

//input: int a

//input: int b

//input: int c

//output: int result

export async function tripleProduct(a, b, c) {

await initNewModule();

return NewModule.\_tripleProduct(a, b, c);

}

// Functions from lib2.c

//name: myCube

//input: int n

//output: int result

export async function myCube(n) {

await initNewModule();

return NewModule.\_myCube(n);

}

9. Now, we test them. In the file package.js, the following is modifications are performed (actually, output… is removed and return … is replaced with alert… ):

// C-FUNCTIONS

// Functions from lib1.c

//name: mySqr

//input: int n

export async function mySqr(n) {

await initNewModule();

alert( NewModule.\_mySqr(n) );

}

//name: tripleProduct

//input: int a

//input: int b

//input: int c

export async function tripleProduct(a, b, c) {

await initNewModule();

alert( NewModule.\_tripleProduct(a, b, c) );

}

// Functions from lib2.c

//name: myCube

//input: int n

export async function myCube(n) {

await initNewModule();

alert( NewModule.\_myCube(n) );

}

Next, publish the package: npx webpack && grok publish dev

Go to <https://dev.datagrok.ai/packages>

Test package MyNewPackage: test functions mySqr, tripleProduct, myCube!