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

In this document, it is shown how to export C-functions to Datagrok package using the export-script. Several examples are used in order to illustrate all steps.

Prerequisites: Datagrok and Emscripten tools must be installed.

Required tools: the script export.py and the file module.json. The first one provides an export of functions from C-files to Datagrok package. In json-file, there are settings of this script.

GitHub: link.

In general, the following actions must be performed:

1) create the folder `wasm` in the folder `package` and copy files export.py & module.json, as well as C-files that contain functions to be exported to package;
2) set parameters in the file `module.json`;
3) in each C-file, add `#include <emscripten.h>` at the beginning, the line `EMSCRIPTEN_KEEPALIVE` and preambula before each exported function;
4) run the script `export.py`;
5) add init-function of the module with exported functions to the init-function of the package.

After that, the exported function can be used in `package.js`. That's it!

Remark. Preparing the preambula before each C-function is of particular importance. It is similar to Datagrok functions annotation and consists of three parts: ***package***, ***mapping*** and ***update***.

For instance,

```
//name: sumOfColumns
//input: column col1
//input: column col2
//output: column result
//map: arr1 is col1
//map: arr2 is col2
//map: sum is new(length)
//map: length is col1.length
//update: result is sum
EMSCRIPTEN_KEEPALIVE
void sumOfArrays(float * arr1, float * arr2, float * sum, int
length)
```

In the ***package*** part, standard Datagrok package function annotation is presented. The ***mapping*** part contains one-to-one correspondence between ***C-function*** and ***package*** arguments. Note that the line `//map: sum is new(length)` specifies that `sum` is a newly created array of the length given in parenthesis, i.e. `length`. Finally, ***update*** part defines changes of ***package*** arguments after executing ***C-function***. For example, after the function `sumOfArrays` is called, `sum` contains sum of col1 and2 content, and the line "`//update: result is sum`" says that `result` should contain `sum`. If both C-function and its package analogue have the same output, then update part concerning this output is skipped.

Consider an example in order to illustrate the required steps.

Example.

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

```
// lib1.c

int minOfArray(int * array, int length)
{
    int result = array[0];

    for(int i = 1; i < length; i++)
        if( result > array[i] )
            result = array[i];

    return result;
}
```

```
// lib2.c

void doubleArray(float * array, int length)
{
    for(int i = 0; i < length; i++)
        array[i] = array[i] * 2;
}

void sumOfArrays(float * arr1, float * arr2, float * sum, int length)
{
    for(int i = 0; i < length; i++)
        sum[i] = arr1[i] + arr2[i];
}
```

The function `minOfArray` takes two arguments: array of integers (`array`) and its length (`length`), and returns the minimum value. The function `doubleArray` also takes two arguments: array of floats (`array`) and its length (`length`), and doubles each value in the array. Finally, the function `sumOfArrays` takes four arguments: three arrays of floats (`arr1, arr2, sum`) and their length (`length`), computes a sum of `arr1` and `arr2`, and stores the results in the array `sum`.

Suppose, one wants to apply these functions to Datagrok column processing as follows:

- the function `minOfArray` to compute the minimum value of the column;
- the function `doubleArray` to obtain column multiplied by 2;
- the function `sumOfArrays` to get a sum of columns.

Follow instructions:

0. Create package (currently, `demoPack`): `grok create demoPack --js --jest` and run in the folder `demoPack` the following: `npm install`.

1. Create the folder `wasm` in the folder `demoPack` and copy files `export.py`, `module.json`, `lib1.c` & `lib2.c` there.

2. Set parameters in the file `module.json`. Consider this process in more detail.

2.1) open the file `module.json`:

```
{
```

```
    "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": "wasm",  ← name of the folder with C-files, export script and its settings
    "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.

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

2.2) modify the file `module.json` (actually, we work with blue-items):

```
{
    "name": "TestExampleName",
    "version": "0.0.1",
    "description": "Some description",
    "folder": "wasm",
    "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.

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

3. Prepare C-functions for export:

3.1) the line `#include <emscripten.h>` is inserted at the beginning of each C-file;

3.2) the line `EMSCRIPTEN_KEEPALIVE` is inserted before each function that should be exported;

3.3) the required preambula is inserted before the line `EMSCRIPTEN_KEEPALIVE`

The following code is obtained in the files `lib1.c` and `lib2.c`:

```
// lib1.c

#include <emscripten.h>

//name: minOfColumn
//input: dataframe df
//input: column col
//output: int num
//map: array is col
//map: length is col.length
EMSCRIPTEN_KEEPALIVE
int minOfArray(int * array, int length)
{
    int result = array[0];

    for(int i = 1; i < length; i++)
        if( result > array[i] )
            result = array[i];

    return result;
}
```

```
// lib2.c

#include <emscripten.h>

//name: doubleColumn
//input: column col
//output: column result
//map: array is col
//map: length is col.length
//update: result is array
EMSCRIPTEN_KEEPALIVE
void doubleArray(float * array, int length)
{
    for(int i = 0; i < length; i++)
        array[i] = array[i] * 2;

}

//name: sumOfColumns
//input: column col1
//input: column col2
//output: column result
//map: arr1 is col1
//map: arr2 is col2
//map: sum is new(length)
//map: length is col1.length
//update: result is sum
EMSCRIPTEN_KEEPALIVE
void sumOfArrays(float * arr1, float * arr2, float * sum, int
length)
{
    for(int i = 0; i < length; i++)
        sum[i] = arr1[i] + arr2[i];
}
```

4. Run the script `export.py` in the folder `wasm`.

5. Open the file `package.js` and add init-function of the exported module to the init-function of the package. You get in the file `package.js` something like the following:

```
. . .
//tags: init
export async function init() {
  . . .
  await initNewModule();
```

```
}
. . .
```

Note that in the the function `init()` you call `await async init<moduleName>`, where **`moduleName`** is defined in the file `module.json` in the field `"moduleName"`. Currently, it is `NewModule,` so we call `initNewModule()`.

That's it! Exported functions can be applied.

It is proposed to add the following functions to the package file:

```
//name: doubleColumnTest
//input: dataframe df
//input: column col
export function doubleColumnTest(df, col) {
  df.columns.add(doubleColumn(col));
}

//name: sumOfColumnsTest
//input: dataframe df
//input: column col1
//input: column col2
export function sumOfColumnsTest(df, col1, col2) {
  df.columns.add(sumOfColumns(col1, col2));
}
```

Publish the package obtained and test it.