mexPort

A toolkit for rapid porting from Matlab to C/C++

mexPort is a productivity tool for simplifying the creation of Matlab .mex files. Matlab .mex files enable accessing C/C++ code directly from your Matlab environment through regular Matlab function calls.
mexPort helps by automatically creating Xcode or MS Visual Studio (MSVC) projects and by providing a generic library called **`mexPort.h`**, which greatly simplifies converting N-dimensional Matlab data to and from C or C++ data types. `mexPort.h` addresses:

- Generic representation of any native C or C++ data types (*e.g.* float, int, etc.) up to virtually any dimensionality
- Complex numbers
- Conversion between Matlab and C or C++ memory representations (*i.e.* row vs. column vectors)
- Strings
- User-transparent, recursive handling of Any-Dimensional data arrays.

Thus, the users can focus on the implementation and testing of their actual algorithm.

Further, you will find examples of how to conveniently call Matlab functions directly from anywhere in your C or C++ code. This can be very a useful tool if you would, for example, like to plot the content of a C++ `std::vector<float>`, etc. It is very simple to extend this interface further to call your own Matlab scripts from your C or C++ library.

1 Create new project

If you are using this tool for the first time, start off by reading Sec. 5 Initial setup.

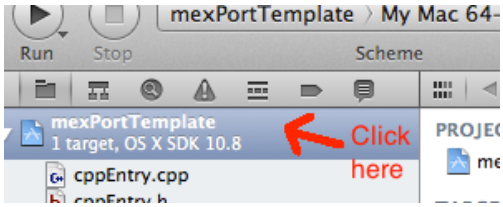
Suppose you would like to build a .mex file named helloMexAlgorithm.

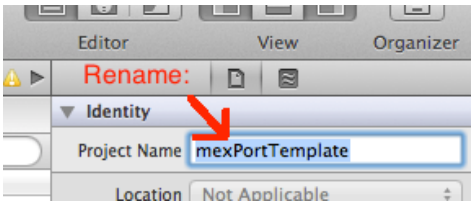
- At the Matlab prompt, go to your `mexPort` folder.

- Type **`createMexPortProject('helloMexAlgorithm')`**. This script will create a new project folder under `mexPort_projects` and an Xcode or MSVC project. The Xcode project will open instantly, on Windows an Explorer window will pop up, simply open the file AllProjects.sln in MSVC and then select "*Add existing project*". Then, select the new project from the respective folder inside `mexPort_projects`.

- Next, we need to rename the project in the IDE. The project name determines the name of the produced .mex file. Naturally one would choose the same name that was given to createMexPortProject, *e.g. helloMexAlgorithm*. In Xcode, make sure the project *mexPortTemplate* is selected (on the left side of Xcode, see Fig. below). To the right of Xcode, under *Identity*, **rename the project** (the name is case-sensitive).

Confirm the renaming of target, etc.. In Matlab you may later call the .mex file after appending the current Build configuration (helloMexAlgorithm_Debug or helloMexAlgorithm_Release).





In MSVC, simply rename the project after it was added to the solution.

Now you are all set, you should be able to build the project (⌘B or F7). The output (either helloMexAlgorithm_**Debug**.mexmaci or helloMexAlgorithm_**Release**.mexmaci, or the respective .mexw file) is placed into `mexPort_projects/Matlab/`. In Xcode this location can be changed in the B*uild settings -> Build phases, -> Run script*; in Windows look in the project settings under *Build Events->Post Build Events*. The .mex file can now be called like any other ordinary Matlab function (by default without parameters or return values), for example:

```
[a, b] = helloMexAlgorithm_Debug(c, d, e) or
[a, b] = helloMexAlgorithm_Release(c, d, e)
```

Note: Debug log output is always preceded with the string "`Warning:`". Just ignore this.

2 Tools for importing/exporting data – mexPort.h

In order to implement your algorithm, two more steps are required; porting data between Matlab and C or C++ data types as well as to do the actual algorithm implementation. When you call a mex function from Matlab with parameters, Matlab's Mex API provides these data as an array of structs of type `mxArray`. In order to make them available to our C/C++ code, we need to import the data from the `mxArray` structs into ordinary C/C++ data types. Matlab stores all its data in memory as column vectors, which is not what one would expect in C or C++. A conversion is necessary. Eventually, we need to export our algorithm's results back to `mxArray` structs in order to make them available in Matlab as return values. **`mexPort.h`** provides several functions that help taking care of all the 'under-the-hood'-issues. In the templates provided, your starting point for porting the data is `mexFunction(…)` in **`cppEntry.cpp`**.

2.1 N-dimensional arrays

The two functions **`importFromMex`** and **`exportToMex`** are your friends. Both functions are dealing with either plain C arrays or C++ `std::vector<>`s of any dimensionality:

Matlab array dimensions	C	C++
1	T*	vector<T>
2	T**	vector<vector<T>>
3	T***	vector<vector<vector<T>>>
…	…	…

where T can be any native C or C++ data type such as `int`, `float`, `double`, `char`, etc.

Examples:

```
vector<vector<int>> arrVec;
importFromMex(prhs[0], &arrVec);
```

For the C variants `mexPort.h` handles memory allocations, but the user is responsible to free the memory later. An additional parameter holding the size of each dimension is therefore necessary, *e.g.*:

```
short*** arr(0); size_t* dims(0);
importFromMex(prhs[0], &arr, &dims);

...

freeNDimArray(arr, dims);
```

The user needs to make sure to match the dimensionality of Matlab arrays and the respective C or C++ data types. Otherwise mexPort will complain at runtime.

Complex data is stored interleaved.

Exporting data to Matlab is done in a similar way:

```
// C++ examples
exportToMex(x1, &(plhs[0]));
exportToMex(x1_c, &(plhs[1]), mxCOMPLEX);
exportToMex(x1_ct, &(plhs[2]), mxCOMPLEX, isColumnVector);

// C examples
exportToMex(a1, sz1, &(plhs[3]));
exportToMex(a1_c, sz1_c, &(plhs[4]), mxCOMPLEX);
exportToMex(a1_ct, sz1_ct,&(plhs[5]),mxCOMPLEX, isColumnVector);
```

Here the user may provide optional parameters to indicate whether the exported data are to be interpreted as complex values (default no) or whether in the case of a one-dimensional array, the resulting Matlab variable should be a column or a row vector (default row vector).

2.2 Scalar data types

```
float floatVal = portFromMex<float>(prhs[0]);
portToMex(xInt, &(plhs[0])); // a scalar value
```

In order to handle complex scalar values, use the array version of the algorithms (see 2.1).

2.3 String values

```
std::string str;
portStringFromMex(prhs[0], &str);
// (C string accessible through str.c_str())

portStringToMex(str, &(plhs[0]));
```

3 Calling Matlab functions from C/C++

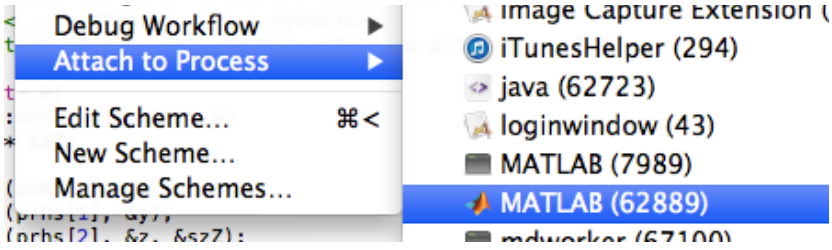
It is possible to call any Matlab function such as **`plot`** directly from within C/C++. See `utils.cpp` and `utils.h` for examples on how to do this, *e.g.*:

```
float* x;
const size_t szX = ...
...
plot(x, szX);
```

4 Typical workflow, Debugging, Contributing to mexPort, etc.

For trying out mexPort, simply pull or get mexPort from gitHub. For more serious projects we recommend you to include mexPort as a subtree into your own git repository.

If you wish to **debug** the C/C++ code, this can simply be done by attaching your IDE (Xcode or MSVC) to Matlab and then setting a breakpoint in your code. In your IDE, select *Attach to Process* (in Xcode from *Product*, in MSVC from *Debug*), then choose the *Matlab* executable. Set your breakpoints in Xcode and call your function from Matlab as before.



If you would like to help improve mexPort you may use another project called mexPort_dev, which can also be found on gitHub. It contains the mexPort library as a git subtree and allows to easily develop new features with respective tests.

5 Initial setup

The Xcode and MSVC projects need to be told the location of your Matlab installation. For convenience, the projects rely on an OS environment variable named MEXPORT_MATLABPATH (an alternative is described below). First, if necessary, close all open Xcode or MSVC windows. At the Matlab prompt type `matlabroot` in order to find out the exact location of your Matlab installation.

In OS X, open a Terminal window and type (or copy/paste):

```
launchctl setenv MEXPORT_MATLABPATH PathToYourMatlab
```

where you replace *PathToYourMatlab* with the path to Matlab on your system (*e.g.* `/Applications/MATLAB_R2013a.app`). To check whether the variable was set correctly, restart the terminal and type `export`. This variable will persist until you log off. In order to make this variable persistent, type (or copy/paste):

```
sudo mv /etc/launchd.conf .
echo setenv MEXPORT_MATLABPATH PathToYourMatlab >> launchd.conf
sudo mv launchd.conf /etc/
```

Again, replace *PathToYourMatlab* with the path to Matlab on your system. In case the file does not exist on your system, omit the first line. This will make the variable `MEXPORT_MATLABPATH` persistent on your system (after restarting OS X) by appending it to the file `/etc/launchd.conf`.

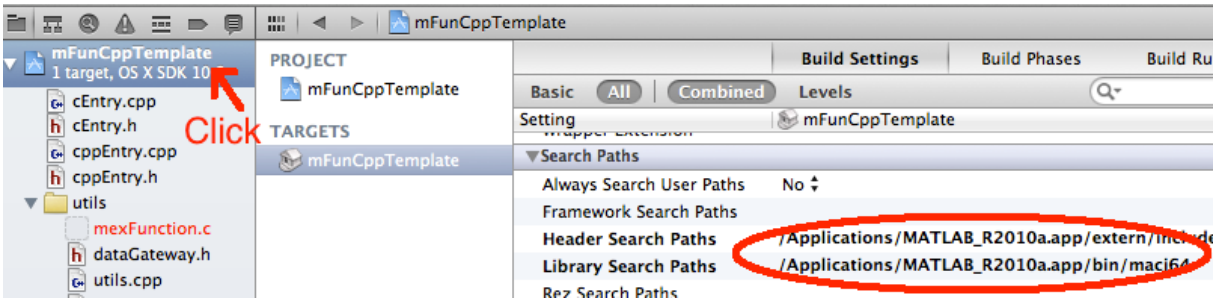
On a Windows machine, right-click on 'Computer', select Properties and then 'Advanced System Settings'. Here, create a new environment variable named `MEXPORT_MATLABPATH` and set its value to something similar to 'C:\Program Files\MATLAB\R2012a'. Again, find the correct path on your system using the command `matlabroot`.

Continue with the description in Sec. 1 Create new project

6 Troubleshoothing

6.1 Matlab path

In case the setting of the environment variable telling your IDE where Matlab is does not work, you can also tell this manually to Xcode or MSVC. In Xcode, on the left hand side, select the project. The project settings should open. Find the settings called **Header Search Paths** and **Library Search Paths** and replace the beginning of each with the path to your Matlab installation. Hint: You can find this path by typing `matlabroot` at the Matlab prompt.



In MSVC, in the project settings look for "Additional Include Directories" under C/C++ and "Additional Library Directories" under Linker to find the appropriate locations.

6.2 extern "C"

Examples of calls to Matlab functions from C/C++ are implemented in `utils.cpp`. This file is placed into a folder that is included by C and also C++ projects. For this reason it is necessary to include `utils.h` as C files, which unfortunately makes C++ function overloading of the utility functions impossible. If you are planning to only work with C++ implementations, you should remove the extern C qualifier.

6.3 Linker error – can't write output file

It may happen that the linker complains that it can't write the output file (.mexmaci64, .mexw32, etc.). Typically this is due to Matlab retaining a lock on the file after having run it. Simply run '`clear all`' or '`clear helloMexAlgorithm_Debug`' (or whatever your .mex file is called) at the Matlab prompt and the lock should have been released.