Creating a Scilab Toolbox

This document explains the important and “not so easily available on the internet” steps involved in creating a Toolbox in Scilab. This document considers the use of Scilab 6.x.x version. The code repository for this document will work only for Scilab 6.x.x.

There are many other resources available on the internet provided officially by scilab.org about creating scilab toolboxes. These are important to refer to in order to understand the directory structure and the API. However, these cover a bigger picture and are very difficult to follow for a beginner. This document will confine the objective to a well-defined case which works in most cases.

There are basically 2 different ways of creating a Toolbox in Scilab. These are

1. Creating a toolbox out of functions written in Scilab language
2. Creating a toolbox out of external functions written in C language

Rather than creating a toolbox from scratch refer to the github repository: https://github.com/FOSSEE/Scilab6-Test-Toolbox
Download or clone this repository. You will get a **Scilab6-Test-Toolbox-master** directory. We will call this directory as “tbx_root” directory

Important

The mentioned github repository, contains Scilab Toolbox code but it does not contain the external functions written in C language. This is because the C function is treated as an external library and is not really a part of the toolbox. Hence, before you proceed with using the toolbox do the following.

1. Download **external-library.zip** file from https://scilab.in/fossee-scilab-toolbox
2. Extract its contents
3. Copy the **thirdparty** directory and paste it in the **Scilab6-Test-Toolbox-master** directory

Setting up build environment for Windows

It is highly recommended to use the open source MinGW compiler over Microsoft Visual Studio. Following are the instructions that you can use to set up the open source build environment

1. MinGW toolbox is available with scilab and can be easily installed. It also requires a specific version of the compiler package provided by Equation Solution, that has to be installed first. This can be downloaded from ftp://ftp.equation.com/gcc/gcc-8.3.0-64.exe The MinGW toolbox for Scilab can be downloaded from https://atoms.scilab.org/toolboxes/mingw/8.3.0 Read the instructions given on that page.

2. After you successfully install the MinGW toolbox and the recommended gcc, the toolbox should load automatically on restarting Scilab. If you have difficulties, re-installing the mingw toolbox in Scilab should help.
3. On Scilab, **haveacompiler** and **mgw_detectCompiler**, both commands should return True
4. On Scilab, **mgw_getArchBinPath** should print the directory of where Equation Solution installed the gcc compiler.
5. If the open source software/library that you are compiling requires some additional packages such as git, cmake etc. you can use **MSYS2**
   Note: The instructions given below this point are not compulsory and have to be followed only if you understand its need
6. Install MSYS2 from https://www.msys2.org
7. Once MSYS2 is installed, we need to bring it up to date and install the required packages. Open the msys app and launch MINGW64 (mingw64-w64) shell. Do not open an MSYS or MINGW32 shell. This is assuming that you are using a 64-bit OS.
8. Type **echo $MSYSTEM** to check if you are in the MING64 shell.
9. In the shell, do **pacman -Syu**
10. Close the terminal and repeat the last three steps until no more updates are received. Close the terminal after each update.
11. Now add new packages. Open MINGW64 terminal (not others) and do **pacman -S <name of required packages>** For example, **pacman -S git cmake** if you want to install git and cmake

Note: While you compile the external libraries, remember to invoke the compiler provided by the Equation Solution and not any other compiler. The **mgw_getArchBinPath** command when used in Scilab can tell you the location of the compiler provided by the Equation Solution. It is important that the external library and Scilab toolbox, both are compiled using the same compiler. Some libraries do provide configure files. It can be used to set the CC, CXX and F77 flags and hence set the location of the required gcc, g++ and Fortran compiler.

Setting up the build environment for Linux OS

For most cases, Linux already comes with all the build tools that are typically used. Any additional tools, if required, must be installed before proceeding with compilation.

The external-library directory

The **external-library** directory contains the source code: a C file and a header file. This directory contains the “mul.c” file which has the function written in C to multiply two numbers. It also contains the corresponding header file. For your convenience, it has a run.sh file too to compile it on a Linux OS. This is a shell script and can be easily executed on a Linux terminal. If on windows, see the file **windows-compile.txt.** It has a list of commands to be executed on windows command prompt in order to compile the external C library.

Compiling the external C library on Linux

This process is very straight forward. Open the Linux terminal and do the following

1. Execute **./run.sh** This will compile the C library into a shared library file, “libmul.so”
2. This file will also automatically copy the generated shared library and the header file inside a folder named as **thirdparty.** This folder already exists inside the external-library folder
3. This **thirdparty** folder can now be used for building a Scilab Toolbox

Compiling the external C library on Windows

It is important that you first follow the instructions given in **Setting up Build environment for Windows** section before proceeding. Once you have it, do the following

1. Open command prompt
2. Type **<path-to-gcc-provided-by-equation-solution>/gcc.exe -fPIC -c -DBUILDING_EXAMPLE_DLL mul.c** The **mgw_getArchBinPath** command when used in Scilab can tell you the path
3. Type **<path-to-gcc-provided-by-equation-solution>/gcc.exe -shared -o libmul.dll mul.o -Wl,--out-implib,libmul.a**
   This will generate a **libmul.dll** file and **libmul.a** file
4. The **libmul.dll** file and **libmul.a** files then have to be copied inside **thirdparty/windows/lib** directory.
5. The **mul.h** file has to be copied inside the **thirdparty/windows/include** directory
6. This **thirdparty** folder can now be used for building a Scilab Toolbox

By default, the directory named as **thirdparty** already exists and has all the library files and the header files for both Linux and Windows. It has to be copied inside the **Scilab6-Test-Toolbox-master** directory.

Step 1: Building and loading the toolbox

The toolbox that you have downloaded, if it contains the “loader.sce” file inside the “tbx_root” directory, then it means the toolbox is already built. To load the toolbox execute the command “exec loader.sce” on the scilab console. It should load it without any errors.
However, in case if it fails to load or the “loader.sce” file is not available, execute the command “exec builder.sce” on the scilab console. This will initiate the build process for the toolbox. If the process is exited successfully, it will create the “loader.sce” file (or overwrite the existing one).

While you are developing a new toolbox, you will often have to run the build and load process. However, scilab will not allow building a toolbox that is already loaded in the workspace. In such a case you have to unload the toolbox, unlink any linked files, clean the directory and then execute the build process. Following are the commands that will come handy.

1. exec unloader.sce
2. exec cleaner.sce
3. ulink

4. clear
5. exec builder.sce
6. exec loader.sce

You can make a scilab script out of these commands and run it every time you want. You may have to put the “exec unloader.sce” inside a try-catch statement because the “unloader.sce” file will not be available if the previous build process has failed.

Step 2: Giving a name to your toolbox

The toolbox that you have downloaded from the above mentioned link has the name “test_toolbox”. This configuration of naming your toolbox is made in certain files. To change the name of this toolbox, you need to make changes in the following files. You essentially have to replace the text “test_toolbox” appearing in these files with the name you want to give to the toolbox.

1. Inside “tbx_root/etc” directory, change the .start and .stop files names to the name of your toolbox
2. Inside “tbx_root/etc” directory, change the content of the .start file
3. Inside “tbx_root/etc” directory, change the content of the .stop file
4. Inside “tbx_root/sci_gateway/cpp” directory, change the content of the “builder_gateway_cpp.sce” file
5. Inside the “tbx_root” directory, change the content of “builder.sce” file

Step 3: Writing the toolbox function definitions using scilab (This step can be skipped if you are only interested in adding an external library)

In this step you are writing your own function definitions purely in scilab and adding it to the toolbox. Take a look at the “add.sci” file inside the “tbx_root/macros” directory. It defines a function “add”. Apart from all the comments, the only content of this function is an **addition** expression between the two inputs and returning the result to the output. We will discuss the comments in the “help” document section. Note that the function definitions are written in a .sci file. Each function that you want to add must be defined in a separate .sci file. There is no limit to how many .sci files you can add. There is also a “buildmacros.sce” file in the macros directory. The content of this file must not be changed. Also, “buildmacros.sce” must exist. It is possible for a toolbox to be entirely made up of function definitions written in scilab alone.

Step 4: Implementing the toolbox function definitions using C/C++

This step is required only when you have any of the following reasons

1. The function definitions that you want to add are fairly complicated and can be best expressed in C/C++ language.
2. The C/C++ functions are already existing and you only want to add them to the toolbox.

We have to write a C program making use of the Scilab API to connect to the function that we want to add. We call these programs “scilab gateway functions”. Take a look at the “sci_multiply.cpp” file inside the “tbx_root/sci_gateway/cpp” directory . To understand the content of this file, refer to
https://help.scilab.org/docs/6.0.2/en_US/api_scilab_getting_started.html
Also see the other references it points to. Notice that the variable “fname[]” defines the name for the function as "multiply" and this name will be available for the user after the toolbox is loaded. Notice that the function definition itself has the name “sci_multiply” which is a convention to follow. Another thing to note is that it includes the “mul.h” header file and also makes a call to the function “mul”. Note that the thirdparty directory is required to be present containing the compiled shared library.

Next we have to make changes to the “builder_gateway_cpp.sce” file inside the “tbx_root/sci_gateway/cpp” directory. Here we make three major edits very carefully.

First is making an entry for our function name in the variable “Function_Names”. This variable is of Matrix type and contains information such as “function name”, “gateway function name”, “format”. If there are more files to include they should come in the next row.

Second is making an entry for the function file name in the variable “Files”. This variable is of Matrix type and each file name has to be entered in double quotes and separated by a comma.

Third is configuring the “C_Flags” and “Linker_Flag” variables as per need inside the “if getos()” loop. For Linux, notice the inclusion of “-lmul” flag which directs it to link to our library “mul”. For Windows, see the **libs** variable. This part has to be carefully configured while including external C files that depend on a third party program.

Step 5: Editing the .start file

A file named as <toolbox-name>.start can be found inside the **tbx_root/etc/** directory. In our case, it is **test_toolbox.start.** This file is called whenever a Scilab toolbox is loaded. Hence, this file can be used to write all instructions to be executed before the toolbox is loaded. Instructions to load the required library can be written in this file using a **link** command. This is an important step in order to successfully load your toolbox.

Step 5: Writing the help entry for the toolbox

Writing help for every function that we add to the toolbox is important. It is one of the steps that you have to execute if you wish to officially publish the toolbox. Toolbox help is something that will appear inside the scilab help window when you execute the command “help” on the scilab console. It will appear as one of the category of help on the left hand side. Expanding the option will list down all of the functions for which you have added and written help for.

Writing help is very straight forward. Help files are actually auto generated based on the content you write following a particular structure. This help content can only be written in the functions defined in the .sci files inside the “tbx_root/macros” directory. These are usually comments but they follow certain keywords which helps in auto-generation of the function help. Comparing the help for a function and the associated content of the .sci file will give you a fair idea. See https://help.scilab.org/docs/6.0.1/en_US/help_from_sci.html for more information on how to write help. One important thing to remember is to not skip a line without commenting it out. Also, if you happen to notice that the modifications that you make are not getting updated, delete the xml files inside the “tbx_root/help/en_US” directory and also the content of the “tbx_root/help/en_US” directory