1 Developer Manual for Scilab2C

Note: This section is for developers who seek to contribute to this toolbox. Developers who want to use this toolbox, please see 'User manual for Scilab2C'.
This section explains the structure of the 'Scilab2c' toolbox, lists out coding guidelines and instructions for contributing to code.

1.1 Structure of the toolbox

As a developer you will mostly be doing modifications in 'macros', 'src' and 'tests' folder. 'macros' folder contains all scilab scripts used for conversion. These macros are grouped into different folders according to their functionality. 'src' folder contains all c files which implement scilab functions in c.

1.2 Coding guidelines

Please follow following guidelines while doing any modifications in the toolbox.

1. Make new folder in 'macros' and 'src/c' only when new scilab or c file do not fit into any of the existing folders. If new folder is added in 'macros' make sure you add file 'buildmacros.sce' and add that folder in 'etc/scilab2c.start' also.

2. Add appropriate comments while writing scilab and c functions.

3. Naming convention to be followed for c functions is '<data_type><function_name><dimension>' where dimension can be 's' or 'a' for scalar and vector/matrix respectively and data type can be one of mentioned in table 1.
   For example, c function names for 'sin' function for data type double will be 'dsins.c' and 'dsina.c' for scalar and matrix respectively.

4. Generated c function names follow following name convention
   <inputs_type_&_dimensions><function_name><outputs_type_&_dimensions> where dimensions can be '0' for scalar types and '2' for vector/matrix types of data. Input and output types can be any one of the specified in table 1
   For example, scilab function 'sin (30)', generated c function name will be 'd0sind0' as input is of type double and scalar (d0) and output is also of same type and dimensions (d0).

5. Interface files provide an interface link to generated c name to a corresponding c function in 'src' folder. For e.g., '#define s0coss0(in) scoss(in)' links 's0coss0' with c function 'scoss'. 's0coss0' is function name generated for cos function using real single precision data as an input and output is of same type. While 'scoss' is implementation of cos in c for scalar inputs of type real single precision.

1.3 Instructions for adding new functions

1. Decide new scilab function for which support for C conversion is to be added.

2. Run that function with arguements different data types if function supports different data types. Note down the data types supported.

3. Go to help file of the same function in Scilab and understand its functionality, cases for input/output arguements (like just one input or variable number of inputs etc), dependance of functionality on input arguements etc.

4. Write C code implementing same functionality as provided by selected scilab function. You may require to write multiple functions (in separate files) as with some scilab function functionality is different for different number or types of input arguements.

5. Name of C function and corresponding c file should correspond to types and number of input/output arguements and name of the function. For naming convention, refer 'Coding guidelines'. For example, a function with name 'functionname' accepting one input arguement of type 'double' scalar and giving output of type 'double' scalar, corresponding C function name will be 'dfunctionnames'.

6. In 'src' folder in toolbox, find suitable folder which contains other functions providing similar functionality. If you think new function does not fit into any of the current folders than only make new folder. Now in selected folder make a new folder with name as that of scilab function name. For example, 'functionname' in above case. Store c files corresponding to this function in this folder. Make sure you have covered all possiblities of input/output arguements.

7. Write header and interface files for new function. A header file with name 'functionname.h' contains definition of all functions defined in that folder. An interface file contains definition

Table 1: Data types

Input/Output data type	Representation
unsigned 8 bit	u8
signed 8 bit	i8
unsigned 16 bit	u16
signed 16 bit	u16
real single precision	s
real double precision	d
complex single precision	c
complex double precision	z
string	g

linking function name generated during conversion from scilab to c and functions written in c. For more details about interface files, refer 'Coding guidelines'.

8. Store header file and interface file in folders named 'includes' and 'interfaces' respectively.

9. Open '/path/to/toolbox/includes/sci2clib.h' Add name of the header files here also at appropriate position.

10. Goto folder 'macros/findDeps'. Update files 'getAllHeaders.sci', 'getAllInterfaces.sci', 'getAllSources.sci'. You need to add paths of corresponding files in these files.

11. Open 'macros/ToolInitialisation/INIT_FillSCI2LibCDirs.sci'. This file basically lists out the functions supported by scilab2c. Whenever a new function is added to scilab2c toolbox, this file must be updated. Each function atleast contains the following description. Actual description can contain more lines depending on the function.

```
1     ClassName = 'Sin';
2
3     // --- Class Annotation. ---
4     PrintStringInfo('Adding Class: '+ClassName+'.',GeneralReport,'
          file','y');
5     ClassFileName = fullfile(SCI2CLibCAnnClsDir,ClassName+
          ExtensionCAnnCls);
6     PrintStringInfo('NIN= 1',ClassFileName,'file','y');
7     PrintStringInfo('NOUT= 1',ClassFileName,'file','y');
8     PrintStringInfo('OUT(1).TP= FA_TP_USER',ClassFileName,'file','y'
          );
9     PrintStringInfo('OUT(1).SZ(1)= IN(1).SZ(1)',ClassFileName,'file'
          ,'y');
10    PrintStringInfo('OUT(1).SZ(2)= IN(1).SZ(2)',ClassFileName,'file'
          ,'y');
11
12    // --- Function List Class. ---
13    ClassFileName = fullfile(SCI2CLibCFLClsDir,ClassName+
          ExtensionCFuncListCls);
14    PrintStringInfo('s0'+ArgSeparator+'s0',ClassFileName,'file','y')
          ;
15
16    // --- Annotation Function And Function List Function. ---
17    FunctionName = 'sin'; //BJ : Done AS : Float_Done
18    PrintStringInfo('       Adding Function: '+FunctionName+'.',
          GeneralReport,'file','y');
```

```
19      INIT_GenAnnFLFunctions(FunctionName,SCI2CLibCAnnFunDir,ClassName
            ,GeneralReport,ExtensionCAnnFun);
20      INIT_GenAnnFLFunctions(FunctionName,SCI2CLibCFLFunDir,ClassName,
            GeneralReport,ExtensionCFuncListFun);
```

Line 1 specifies the name of class. Single class can describe multiple functions if input/output arguemnts types/numbers are same for multiple functions.
Line 3,4 and 5 are same for all functions.
Line 6,7 specify number of inputs and outputs respectively. Line 8 specify type of the output. Line 9 and 10 combinely specify the size of output. Refer to other descriptions for filling out these lines. If function supports variable number of inputs, add multiple of these 5 lines for each possibility of number of inputs. For e.g., if function can take 1, 2 or 3 input arguments, there will three sets of these lines. Line 14 specifies all possible combinations of inputs/outputs for differnet types and numbers. Add as much of these lines as required. On line 17, insert correct function name. This is same as scilab function name. Add remaining lines as it is.

12. Run 'builder.sce' and 'loader.sce' files in main toolbox folder.

13. Now write a scilab script which makes use of newly added function and convert it using scilab2c toolbox. Check the results obtained. If results are not as desired, modify c files as required and check again. Do it iteratively untill correct results are obtained.

1.4 Instructions for adding new functions using LAPACK Subroutine

1. Lapack subroutine supports many functions, LAPACK (Linear Algebra Package) is a standard software library for numerical linear algebra.Find a suitable sub-routine which is required in your function. Lets suppose you wish to write a program for SVD (Singular Value Decomposition), then DGESVD subroutine is used for real values.

2. Go to Netlib website or google it, type your subroutine name in search query, than youll find decription of subroutine and the input variables required.

3. Include lapack.h in your C program and include the name of function in lapack.h also.
   #include lapack.h

4. As the function call is external, so before wrting the function name declare the function using extern keyword. As lapack library takes all the variable as address, keep this point in mind and declare variables accordingly.
   for SVD function:
   extern double dgesvd_(char*,char*,int*,int*,double*,int*,double*,double*,int*, double*,int*,double *,int*,int*);

5. Read the description of input variables and according to your needs choose variables values.
   For example SVD function:
   char JOBU = A // all M columns of U are returned in array U
   char JOBVT = A // all N rows of V**T are returned in the array VT;
   Similarly for other variables.

6. After assigning all the variables appropiate values and memory call the function.
   For example SVD function:

   dgesvd_(&JOBU,&JOBVT,&M,&N,buf,&LDA,S,U,&LDU,VT,&LDVT,WORK,&LWORK,&INFO);