Documentation for 'Scilab2C' Scilab extension

1 Introduction

'Scilab' is a programming language widely used by researchers and students for scientific computations. Scilab is very easy to use and learn. But with easiness of use comes the problem of porting to different platforms (specially embedded devices) and execution speed where C coding takes the precedence. Wouldn't it be nice if there is an utility to convert scilab code to C code? Here comes the 'Scilab2c' to help. 'Scilab2c' is an extension for Scilab for converting scilab files to C code. This document gives an introduction about 'Scilab2C', how to use it and flowcharts for 'Scilab2C' extension. Contents are arranged as follows:

- Need for code conversion
- Introduction to 'Scilab2C'
- How to use 'Scilab2C'
- Flowcharts for 'Scilab2C'.

2 Need for code conversion

As stated in introduction, Scilab is a high level programming language easily understood by majority of the scientific community. Since syntaxing is very easy to learn, it is preferred over other programming languages like C, Java or Python. But code written in Scilab is not portable to embedded devices as compiler for compiling scilab code to machine languages is not available. On the other hand, code written in C is easily portable to embedded devices because of the avilability of such compilers. But learning C and writing code in C is not that easy as syntaxing rules are quite strigent. So, an extension like 'Scilab2C' is very useful for converting codes written in scilab to C code. Another advantage of C code is that it may consume less space in memory and execute faster than an equivalent scilab code. Hence, a covertor tool like 'Scilab2C' is very useful for the user.

3 Introduction to 'scilab2c'

4 User manual for Scilab2C

This section describes steps to be followed for using Scilab2C. Pre-requisites are mentioned followed by procedure to install Scilab2C.

4.1 Installation

4.1.1 Prerequisites

There are few prerequisites or some packages must be pre installed before we can use Scilab2C. These are:

- scilab >=5.5.1.
- scilab-arduino toolbox (If using Scilab2C to generate code for Arduino)
- Arduino makefile (https://github.com/sudar/Arduino-Makefile). Install using 'sudo apt-get install arduino-mk.
- BCM2835 C library for RasberryPi (http://wiringpi.com/)
- RasberryPi tools (For cross compiling code for RasberryPi)

4.1.2 Installing Scilab2C

Before we can use 'Scilab2C' extension, we need to install latest version of Scilab2C. Follow following procedure to get latest source code from github repo.

- Open terminal window. (Ctrl+Shift+T is shortcut).
- Change current directory to '/path/to/scilab/share/scilab/contrib'
- Clone the git repo using following command:
  `git clone https://github.com/siddhu8990/Scilab2C.git`
- Make sure a directory named 'Scilab2C' is present in 'contrib' folder.
- Open Scilab.
- Run 'builder.sce file present in Scilab2C/2.3-1 using 'exec'. This generates binary files from source files.
  `exec(/path/to/Scilab2C/2.3-1/builder.sce)`

- In 'Home/.scilab/scilabx.x.x' make a new file '.scilab' if it does not exist already. Open .scilab using suitable editor. Add following line in this file:
  `exec(/path/to/Scilab2C/2.3-1/loader.sce)`
  This will load the 'scilab2c' everytime scilab is started.

4.1.3 Installing supporting packages

Most of the supporting packages or libraries which are required are provided with the toolbox. But they were compiled using latest source code available at release of toolbox. If you want to use latest libraries, steps to compile the same are listed below. You can follow these steps and replace old files with newly generated ones.

- **scilab-arduino toolbox**
- **RasberryPi tools**
  - Make a folder named 'RasberryPi tools' somewhere on the harddisk.
  - Open terminal and change directory to 'RasberryPi tools'. Clone 'Tools' repo using 'git clone https://github.com/raspberrypi/tools.git'.
  - Add location of toolchain to your 'PATH' variable.
    `'export PATH=$PATH:/location/of/tools/folder/arm-bcm2708/gcc-linaro-arm-li`
- **BCM2835 C library for RasberryPi**
  - Before going further, make sure that you have installed 'RasberryPi tool' following the instructions given above.
  - Download latest source code from 'http://www.airspayce.com/mikem/bcm2835'.
  - Extract source code at some suitable location on harddrive.
  - Open the terminal window and change current directory to the location where source is extracted.
  - Execute following command
    `./configure -host=arm CC=arm-linux-gnueabihf-gcc ar=arm-linux-gnueabihf-ar`
  - Then execute 'make' to cross compile the library. Don't do 'make install' as it is noramlly next step.
- **Cross compiling Lapack and Blas for RasberryPi**
  - Download latest source code for Lapack from 'http://www.netlib.org/lapack/'. Extract source files at some suitable location.
  - Open file 'make.inc.example' given in Lapack folder using some editor.

- Edit following items as shown:
  - FORTRAN = arm-linux-gnueabihf-gfortran
  - LOADER = arm-linux-gnueabihf-gfortran
  - CC = arm-linux-gnueabihf-gcc
  - ARCH = arm-linux-gnueabihf-ar
  - RANLIB = arm-linux-gnueabihf-ranlib
- Since we are cross compiling for some other platform, normal way compiling will not work.
- Open terminal window and change current directory to laplack directory.
- We will need to compile BLAS, CBLAS and Lapack separately and in same order.
- Change current directory to /path/to/lapack/BLAS/SRC and run 'make'. This will generate 'librefblas.a' in Lapack folder.
- Now change current directory to /path/to/laplack/CBLAS and run 'make'. This will generate 'libcblas.a' in Lapack folder.
- Now change current directory to /path/to/laplack/SRC and run 'make'. This will generate 'liblapack.a' in Lapack folder. Now replace the generated lib files in 'src/c/hardware/rasberrypi/libraries' in 'scilab2c' source folder.

- **GNU Scientific Library (GSL) for RasberryPi**
  - Before going further, make sure that you have installed 'RasberryPi tool' following the instructions given here.
  - Get latest source code for GSL from `ftp://ftp.gnu.org/gnu/gsl/`.
  - Extract source code at some suitable location on harddrive.
  - Open the terminal window and change current directory to the location where source is extracted.
  - Execute following command
    `./configure -host=arm CC=arm-linux-gnueabihf-gcc ar=arm-linux-gnueabihf-ar --enable-static`
  - Then execute 'make libgsl.la' to cross compile the library. Don't do 'make install' as it is noramlly next step.
  - Library 'libgsl.a' is created in folder '.libs'. By default this folder is hidden.
  - Now replace the generated lib file in 'src/c/hardware/rasberrypi/libraries' in 'scilab2c' source folder.

4.2 Using Scilab2C for C code generation

Scilab2C extension in scilab can be used for generating C code from a scilab script. Currently it supports three types of target platforms:

- **Standalone C code**: General C code which can be compiled using any compiler.
- **Arduino** : Generated code with few additions can be run on an Arduino board. (A scilab-arduino extension is required)
- **AVR** : Code can be generated for using hardware peripherals of AVR microcontroller

You can follow following steps for generating C code using scilab2c extension for required target platforms.

- **Generating standalone C code**

  1. Write the scilab script first which is to be converted to C. Scilab code can contain single file or many files, but each file must be a scilab function. There must be one main scilab file in case project contains many files, from which execution of code starts. All scilab files must be in a single folder.
  2. Before a scilab can be translated to c file, some function annotations should be added manually. Function annotations gives information about no. of inputs/outputs, their types etc. Refer 'Function annotations' for more details.
  3. Type 'sci2c_gui' or 'scilab2c' in scilab console. This will prompt the GUI of Scilab2C extension as shown in figure 1
  4. Click 'Browse' next to 'Main file name' textbox, browse to location of main scilab file and select it. (Refer figure 2)
  5. If scilab code contains many files, select folder containing these file by clicking 'Browse' next to 'Sub-functions' textbox.
  6. Create a new folder somewhere on the disk, preferably in same folder containing scilab files. Select this newly created folder by clicking 'Browse' next to 'Directory name' textbox. (Refer figure 3)
  7. Choose appropriate options from 'Options' box. Different options are explained below:

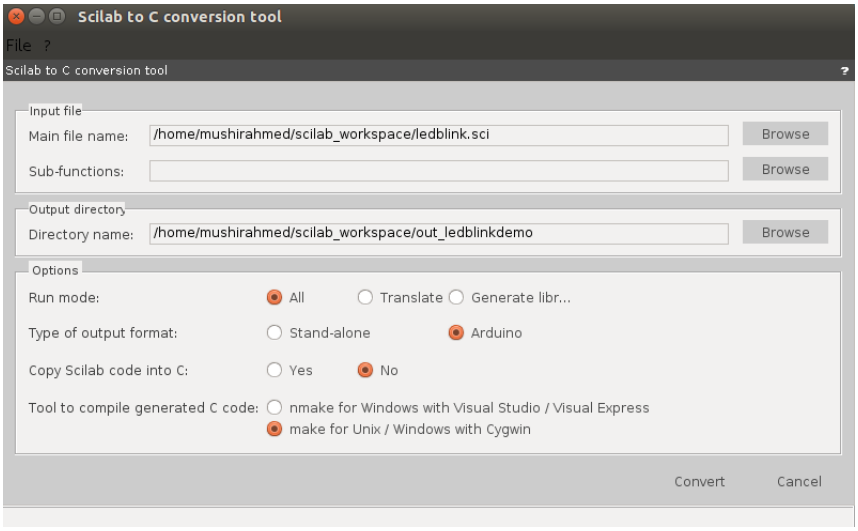


Figure 1: GUI for 'Scilab2C'

(a) Run mode : If only directory structure is to generated in output directory, select 'Generate library'. If only conversion of scilab files is to be done, select 'Translate'. In case both are to be done, select 'All'.

(b) Target platform : To generate standalone C code, select 'Standalone C' from dropdown. (Refer figure 4)

(c) Copy scilab code into C: Select 'Yes' or 'No' accordingly.

(d) Tool to complie generated C code: Select appropriate option depending upon platform on which generated code will be complied.

8. Confirm everything again and then press 'Convert' button. (Refer figure 5)

9. After clicking 'Convert', scilab code will be run in scilab, to check for any errors. If code runs successfully, a prompt will occur asking if you want to continue to code conversion or not. Select 'Yes'. If scilab code doesnt run correctly then code conversion is stopped there itself. Correct the scilab code and follow the steps again.

10. After selecting 'Yes' for code conversion, code conversion starts. If code conversion is done successfully, you will see the message in command window.

11. Generated code can be seen in output folder. By default a makefile is generated which uses 'GCC' compiler to compile the C code. You can compile this code using 'make'. Open output folder in terminal and

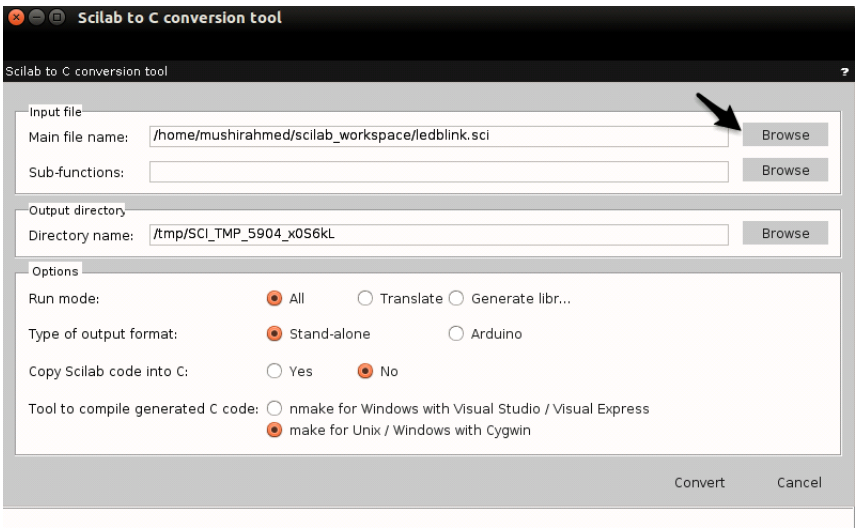


Figure 2: Select 'main' scilab file for conversion

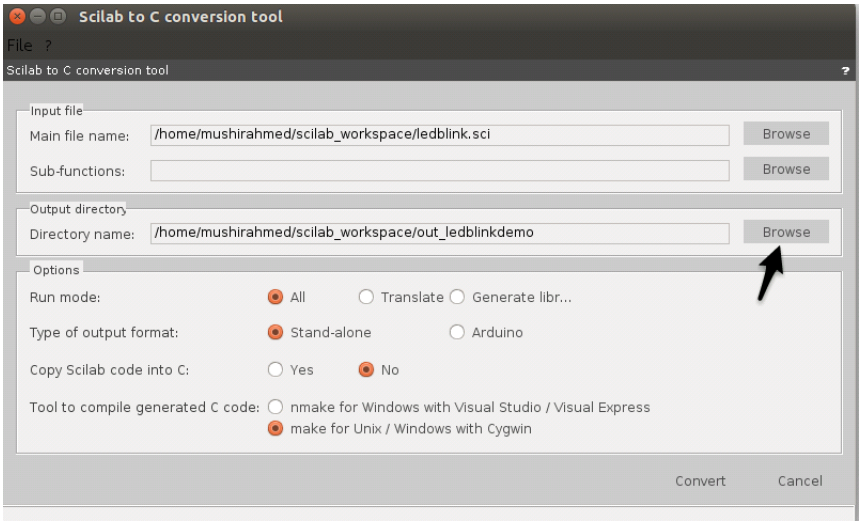


Figure 3: Select output folder

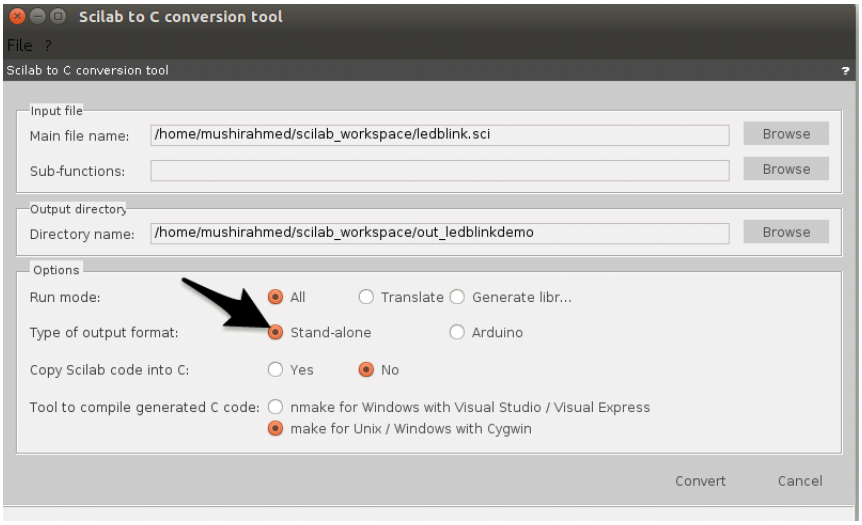


Figure 4: Select 'Standalone C' from dropdown

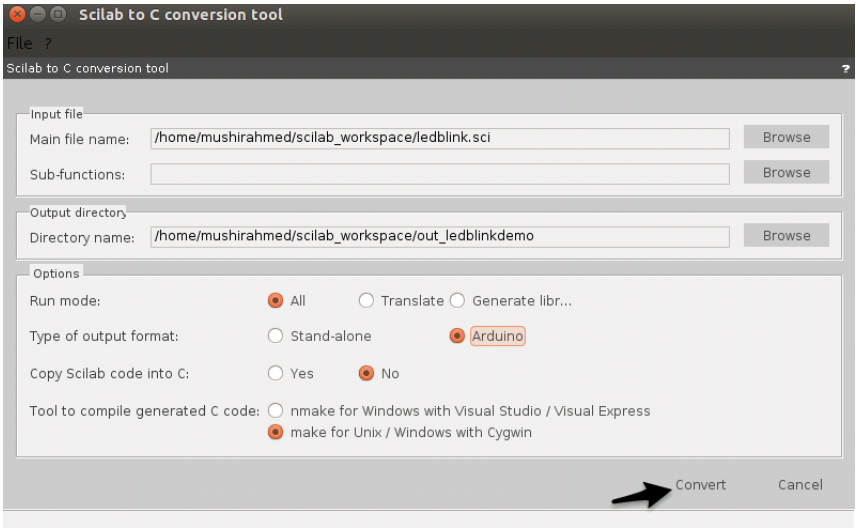


Figure 5: Select output folder

type ‘make’ and press Enter. Once code is compiled successfully, it is run in terminal and output can be seen in terminal window. Check the output for correctness. If code did not behave as expected, correct the scilab code and follow the process again.

- **Generating code for Arduino**

  1. Write the scilab script first which is to be converted to C. Scilab code can contain single file or many files, but each file must be a scilab function. There must be one main scilab file in case project contains many files, from which execution of code starts. All scilab files must be in a single folder. You can verify working of scilab script by runnig it on an Arduino board. Modify the script untill code behaves as expected. Once script is finalised, remove the commands ‘open_serial’ and ‘close_serial’.
  2. Type ‘sci2c_gui’ or ‘scilab2c’ in scilab console. This will prompt the GUI of Scilab2C extension as shown in figure 1
  3. Click ‘Browse’ next to ‘Main file name’ textbox, browse to location of main scilab file and select it. (Refer figure 2)
  4. If scilab code contains many files, select folder containing these file by clicking ‘Browse’ next to ‘Sub-functions’ textbox.
  5. Create a new folder somewhere on the disk, preferably in same folder containing scilab files. Select this newly created folder by clicking ‘Browse’ next to ‘Directory name’ textbox. (Refer figure 3)
  6. Choose appropriate options from ‘Options’ box. Different options are explained below:
     (a) Run mode : If only directory structure is to generated in output directory, select ‘Generate library’. If only conversion of scilab files is to be done, select ‘Translate’. In case both are to be done, select ‘All’.
     (b) Target platform : To generate C code for arduino, select ‘Arduino’ from dropdown. (Refer figure 6)
     (c) Copy scilab code into C: Select ‘Yes’ or ‘No’ accordingly.
     (d) Tool to complie generated C code: Select appropriate option depending upon platform on which generated code will be complied.
  7. Confirm everything again and then press ‘Convert button. (Refer figure 5)
  8. Code conversion will start, promting different messages in command window. If conversion completes successfully, prompt will occur in command window indicationg the same.

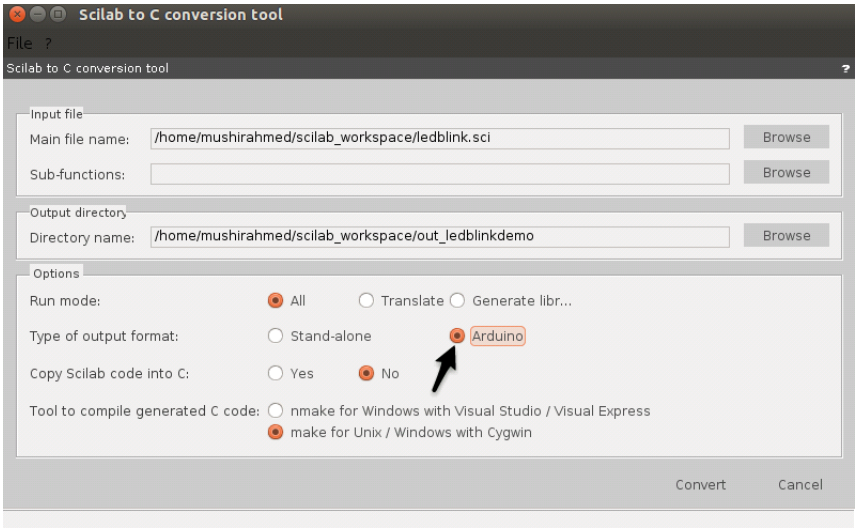


Figure 6: Select 'Standalone C' from dropdown

9. Generated code can be seen in output folder. A separate folder named 'Arduino' is created, which contains a makefile and an arduino sketch file – sci2c_arduino.ino.
10. Open 'Makefile' using suitable text editor. Change following parameters according to board and connection:
    (a) BOARD_TAG
    (b) ARDUINO_PORT
11. Open the terminal and change current directory to the directory containing modified Arduino sketch and then compile by typing 'make' in terminal.
12. If code is compiled successfully, you can upload it to arduino using 'make upload' command.
13. If code doesnot behave as expected, modify sclab code and follow the steps again.

- **Generating code for AVR**

4.3 Function Annotations

Each scilab function/file should start with the function annotation section having following structure:

5 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.

5.1 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.

5.2 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. 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 **??**

4. 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'

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

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.

5.3 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 slected folder make a new folder with name as that of c 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 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 'includes/sci2clib.h' Add name of the header file here also.
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 fo 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/output for differnet types and numbers. Add as much of these lines as required. On line 17, inser 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 scilab2 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.

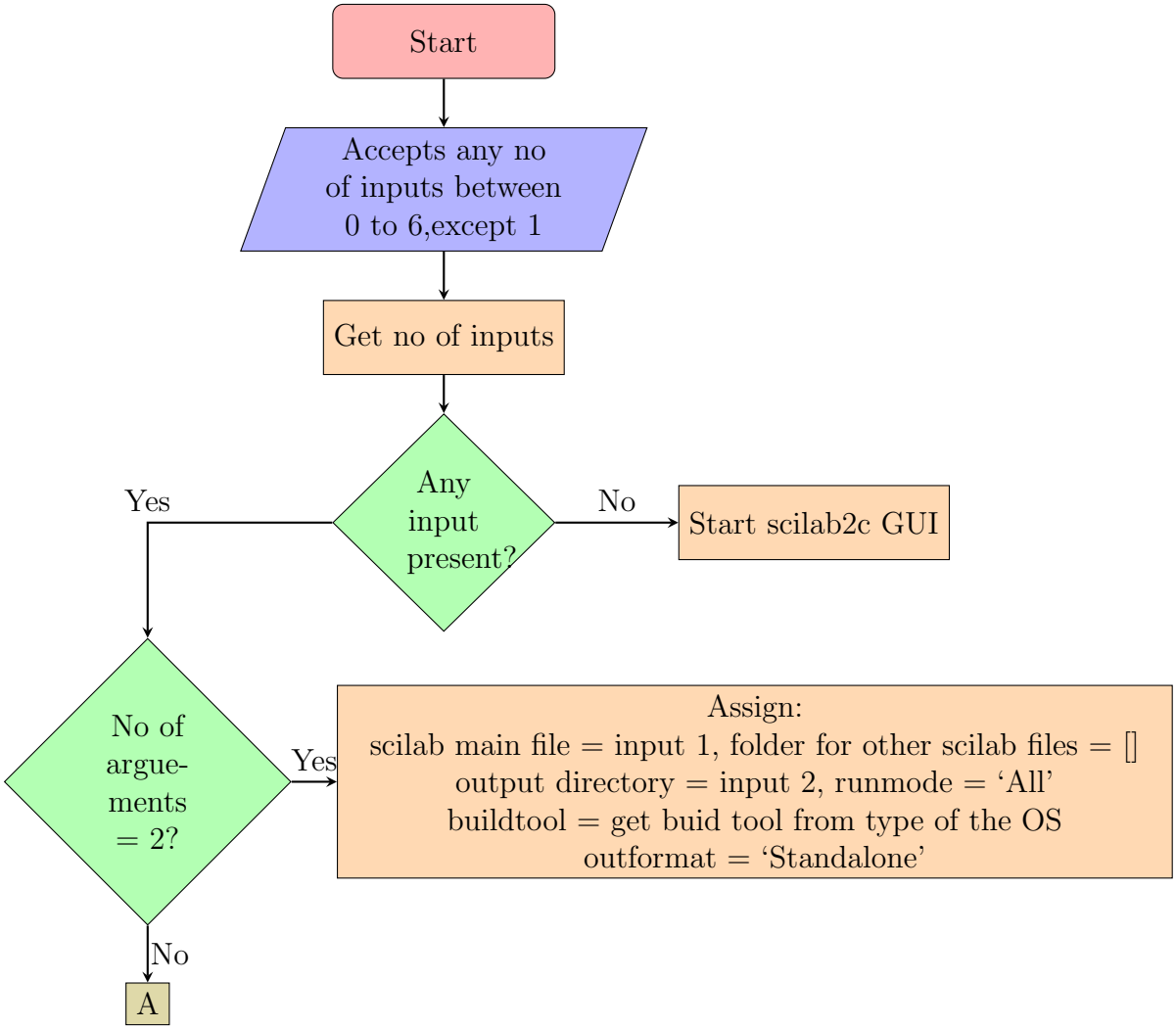
6 Flowcharts

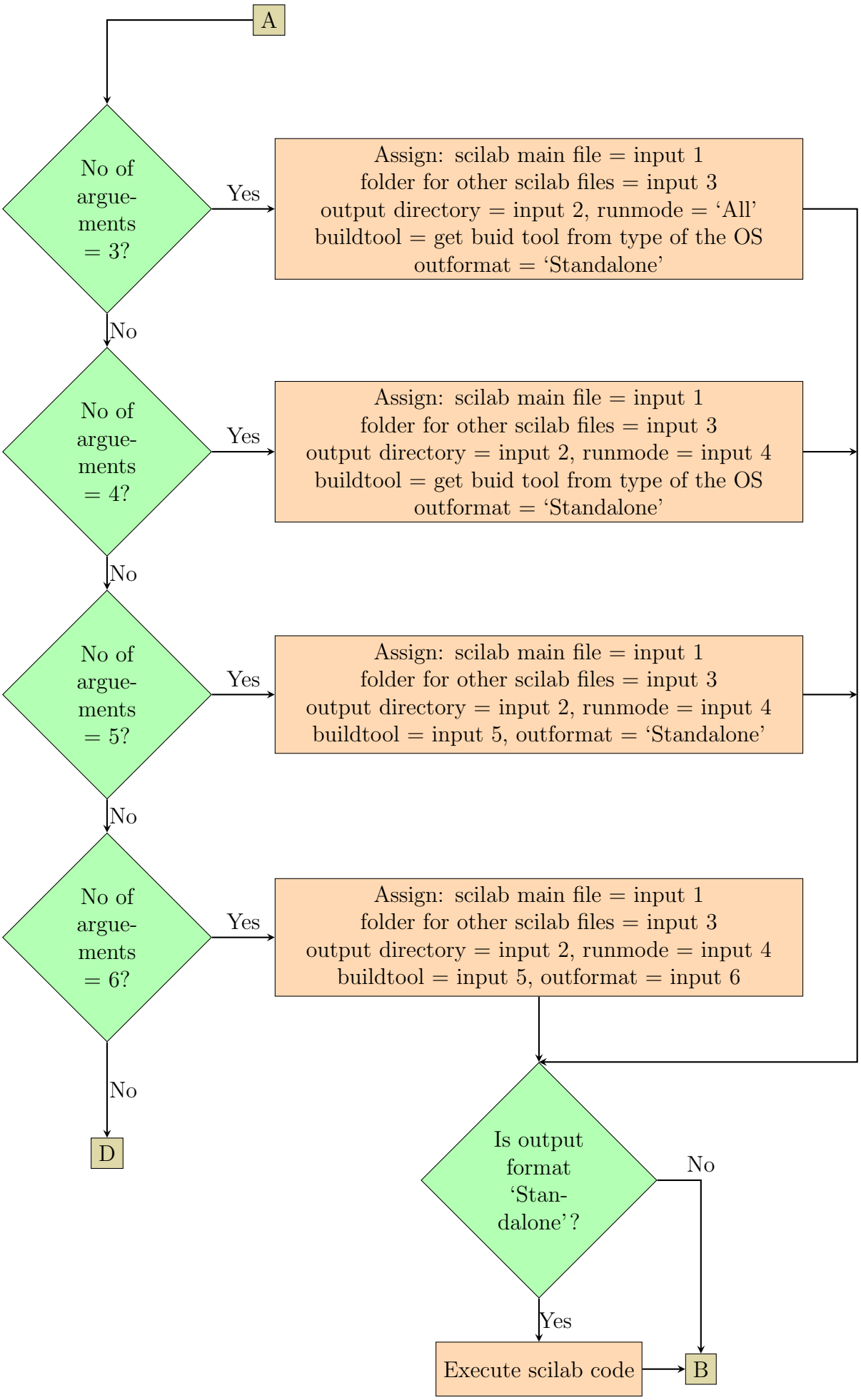
scilab2c.sci

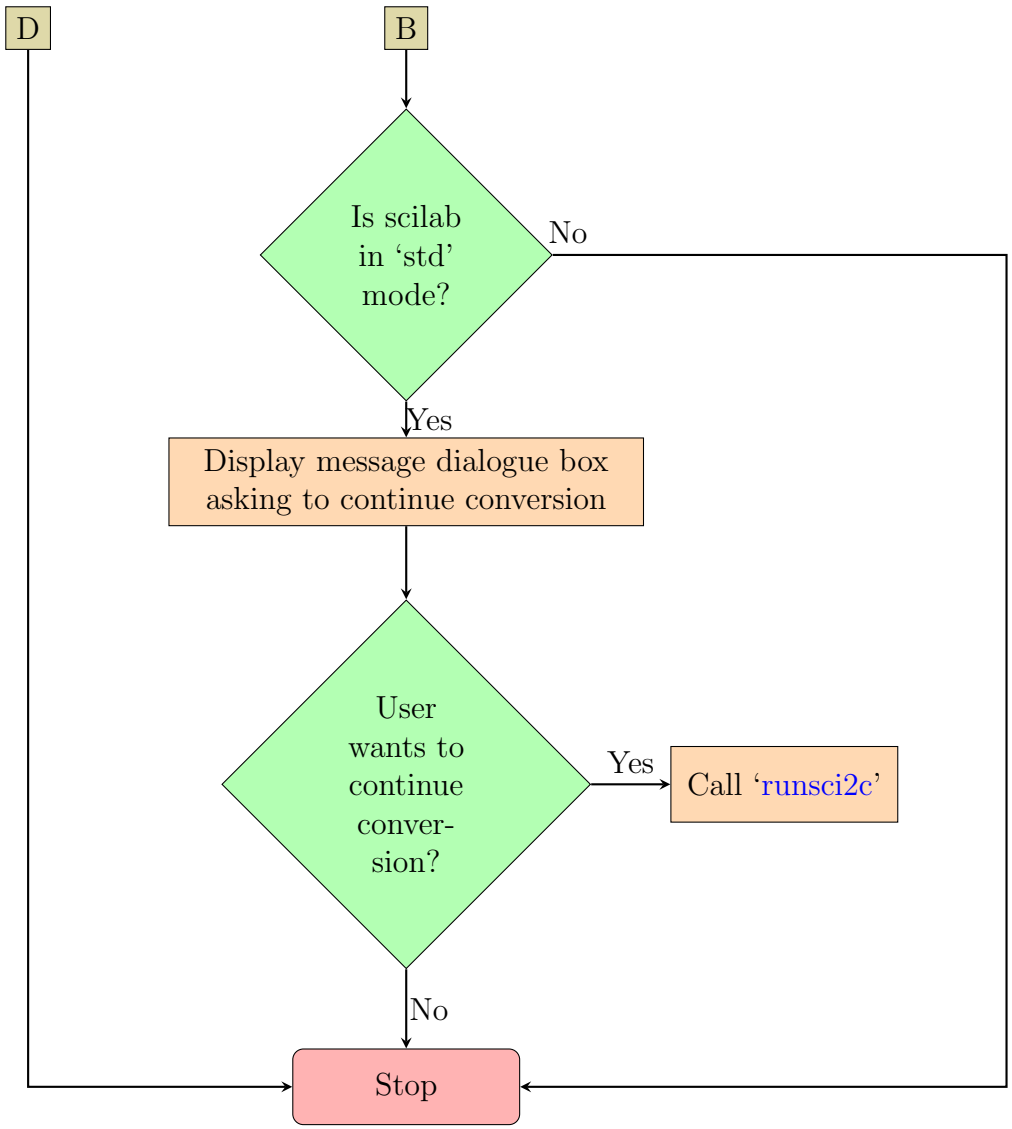
Siddhesh Wani

Introduction

'scilab2c' is the main function for scilab2c module. Code execution for scilab2c starts from this function. 'scilab2c' function can be used from command line. It accepts variable number of arguements and depending upon no of arguements and their values it calls 'runsci2c'.




A
No of argue-ments = 3?
Yes
Assign: scilab main file = input 1
folder for other scilab files = input 3
output directory = input 2, runmode = 'All'
buildtool = get buid tool from type of the OS
outformat = 'Standalone'
No
No of argue-ments = 4?
Yes
Assign: scilab main file = input 1
folder for other scilab files = input 3
output directory = input 2, runmode = input 4
buildtool = get buid tool from type of the OS
outformat = 'Standalone'
No
No of argue-ments = 5?
Yes
Assign: scilab main file = input 1
folder for other scilab files = input 3
output directory = input 2, runmode = input 4
buildtool = input 5, outformat = 'Standalone'
No
No of argue-ments = 6?
Yes
Assign: scilab main file = input 1
folder for other scilab files = input 3
output directory = input 2, runmode = input 4
buildtool = input 5, outformat = input 6
No
D
Is output format 'Stan-dalone'?
No
Yes
Execute scilab code
B


D
B
Is scilab in 'std' mode?
No
Yes
Display message dialogue box asking to continue conversion
User wants to continue conver- sion?
Yes
Call 'runsci2c'
No
Stop

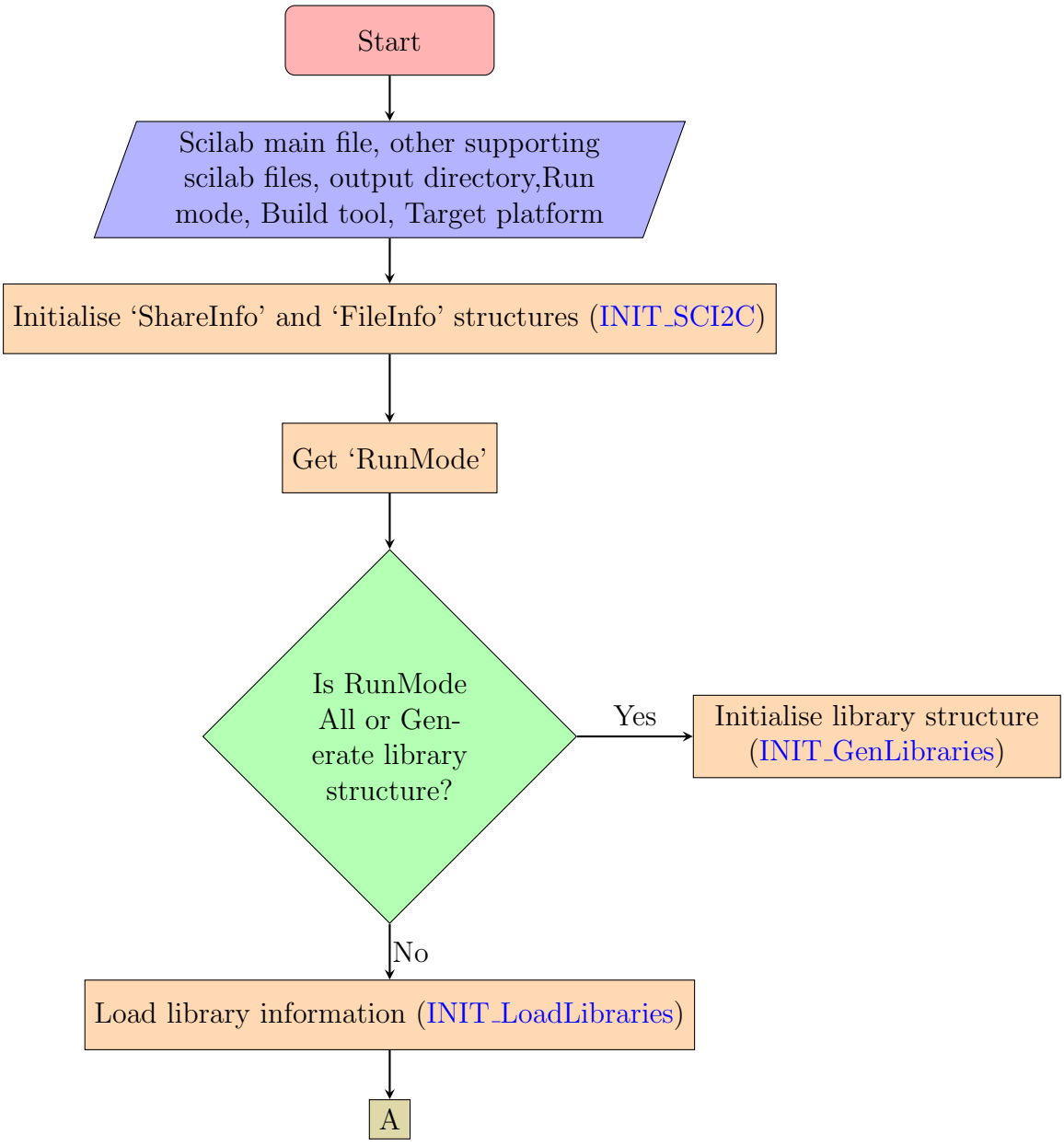
;

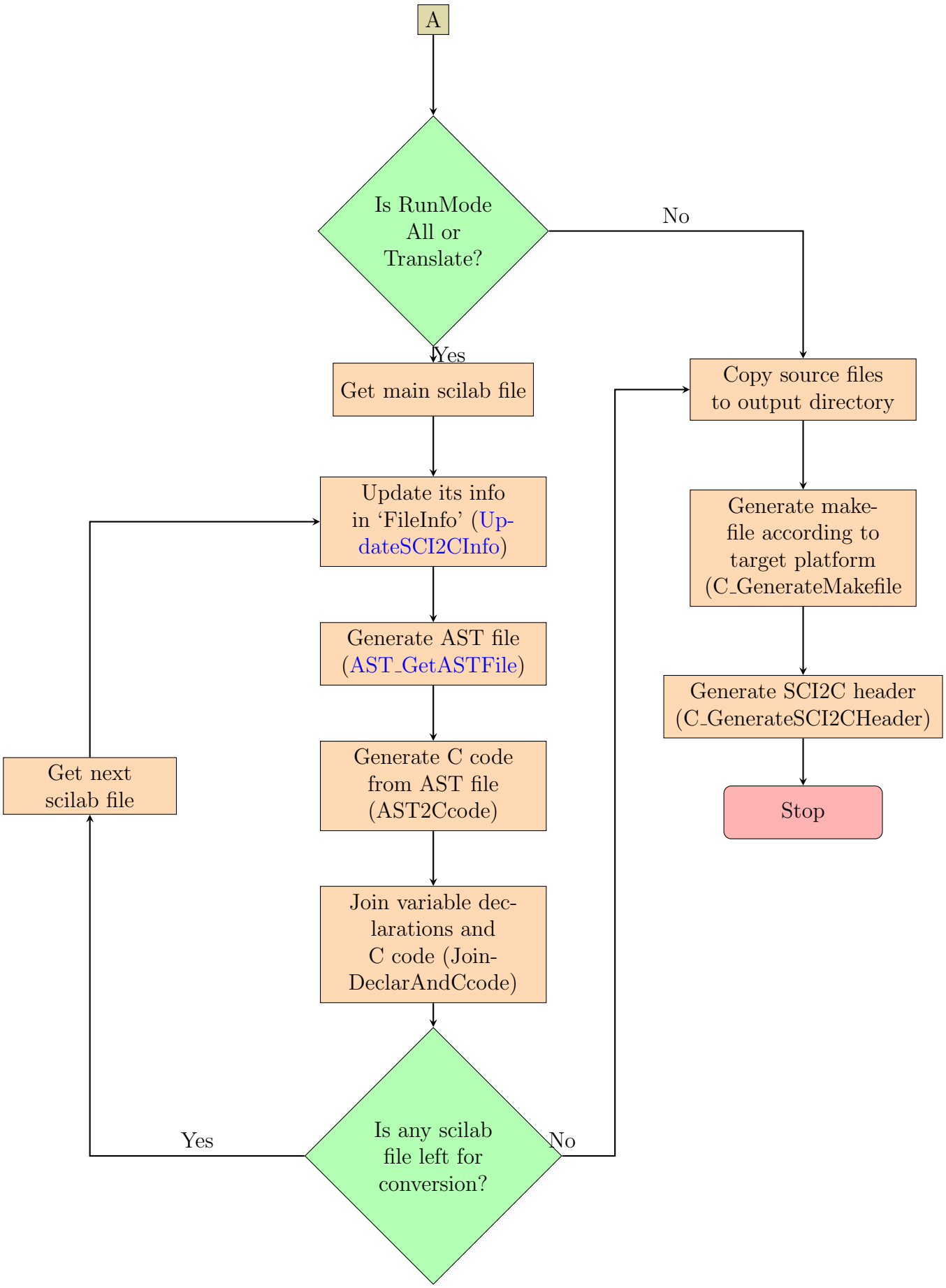
runsci2c.sci

Siddhesh Wani

Introduction

'runsci2c' is called by 'scilab2c'. It calls some initialisation modules and then does the conversion. After conversion is complete, it copies the source files in output directory and then calls other module to generate makefile for compilation.




A
Is RunMode All or Translate?
No
Yes
Get main scilab file
Update its info in 'FileInfo' (UpdateSCI2CInfo)
Generate AST file (AST_GetASTFile)
Generate C code from AST file (AST2Ccode)
Join variable declarations and C code (JoinDeclarAndCcode)
Is any scilab file left for conversion?
Yes
Get next scilab file
No
Copy source files to output directory
Generate makefile according to target platform (C_GenerateMakefile
Generate SCI2C header (C_GenerateSCI2CHeader)
Stop

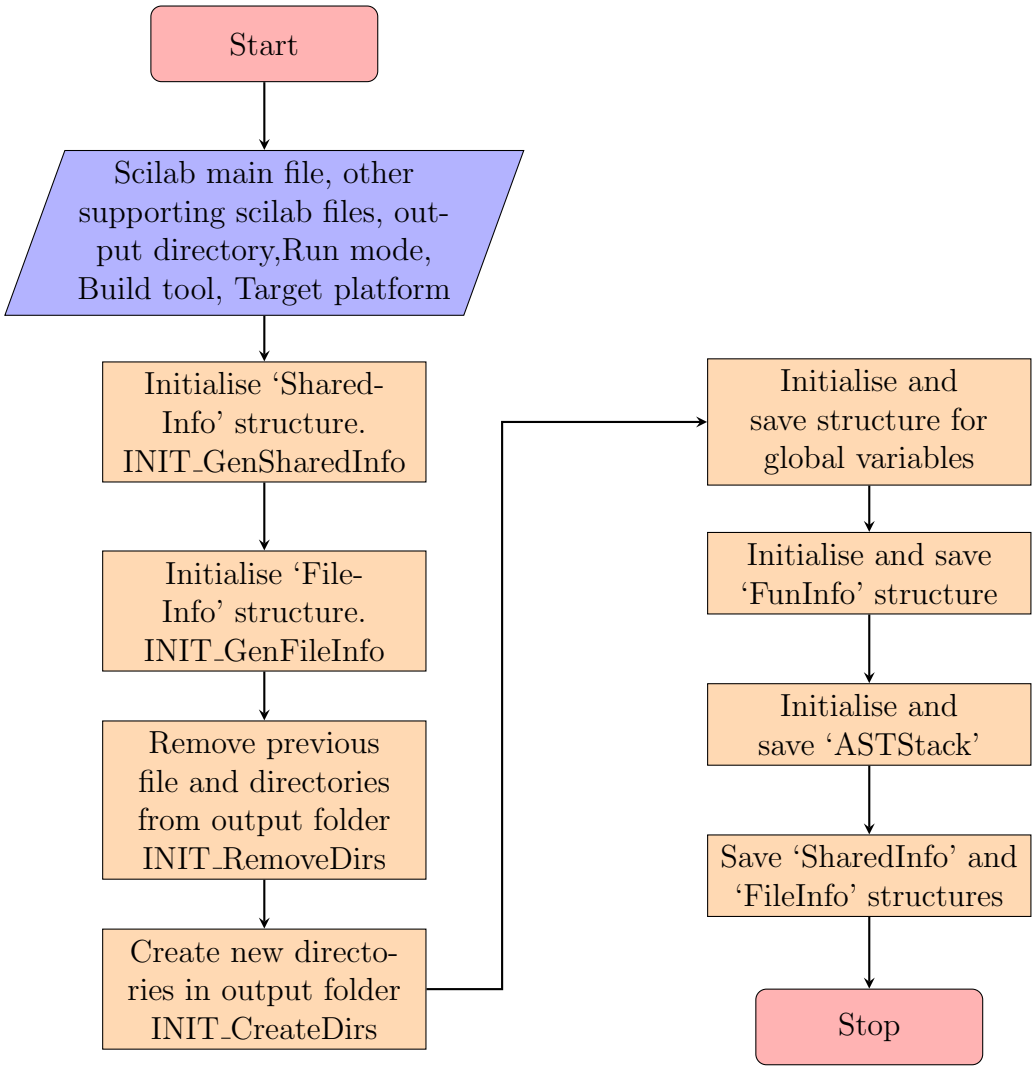
;

INIT_SCI2C.sci

Siddhesh Wani

Introduction

'INIT_SCI2C' initialises sscilab2c extension using the given input parameters. 'FileInfo' and 'SharedInfo' structures are initialised and stored in .dat files. Directory structure is created. Other few .dat files required by extension are initialised and saved on disk.



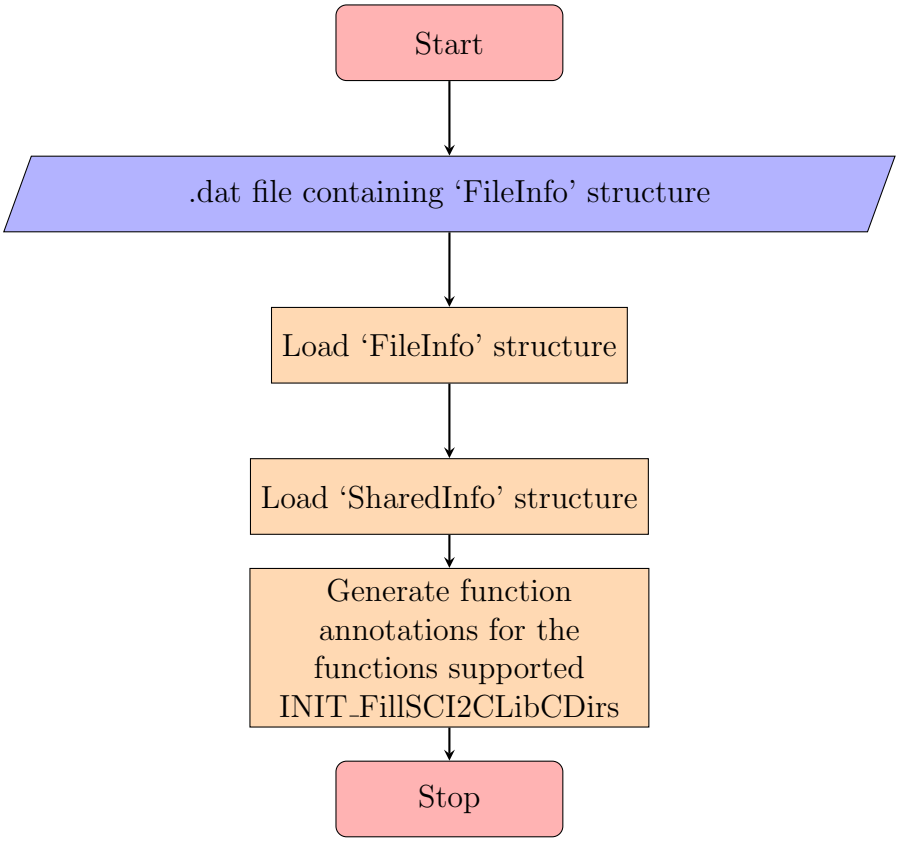
;

INIT_GenLibraries.sci

Siddhesh Wani

Introduction

'INIT_GenLibraries' call 'INIT_FillSCI2CLibCDirs' which generates function annotations for the functions supported by scilab2c extension.



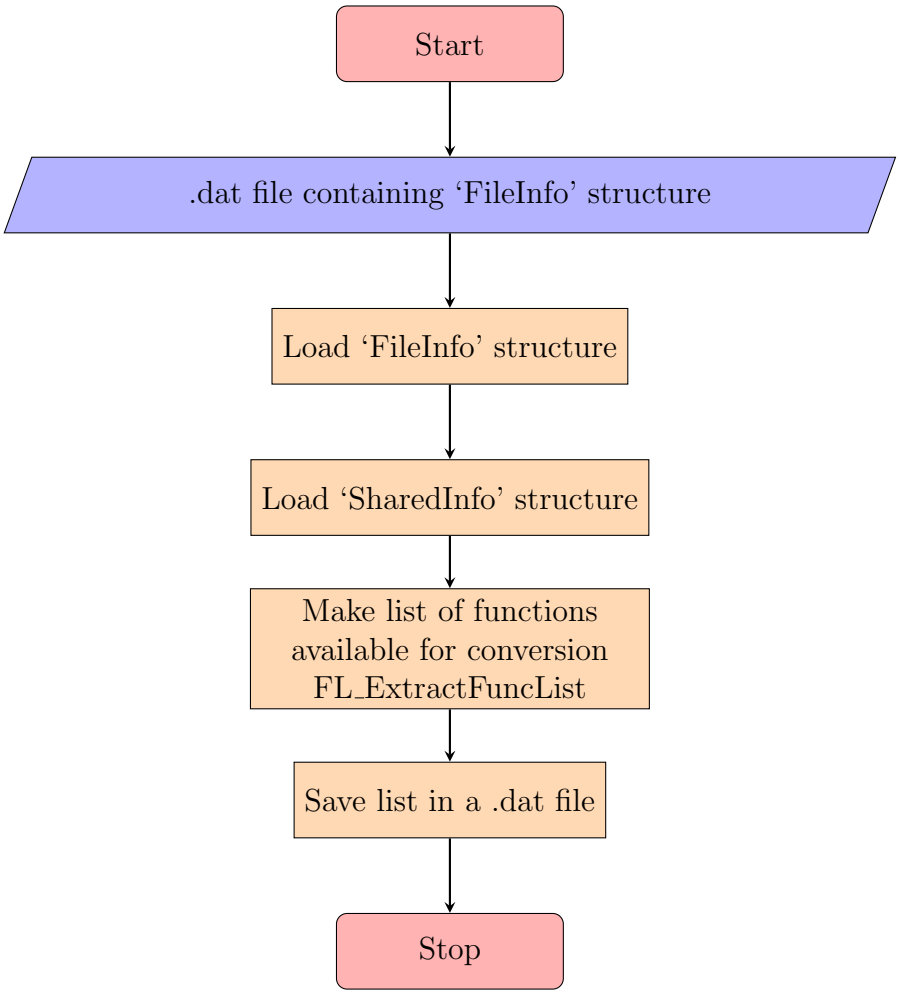
;

INIT_SciLibraries.sci

Siddhesh Wani

Introduction

'INIT_LoadLibraries' generates list of available function annotations.



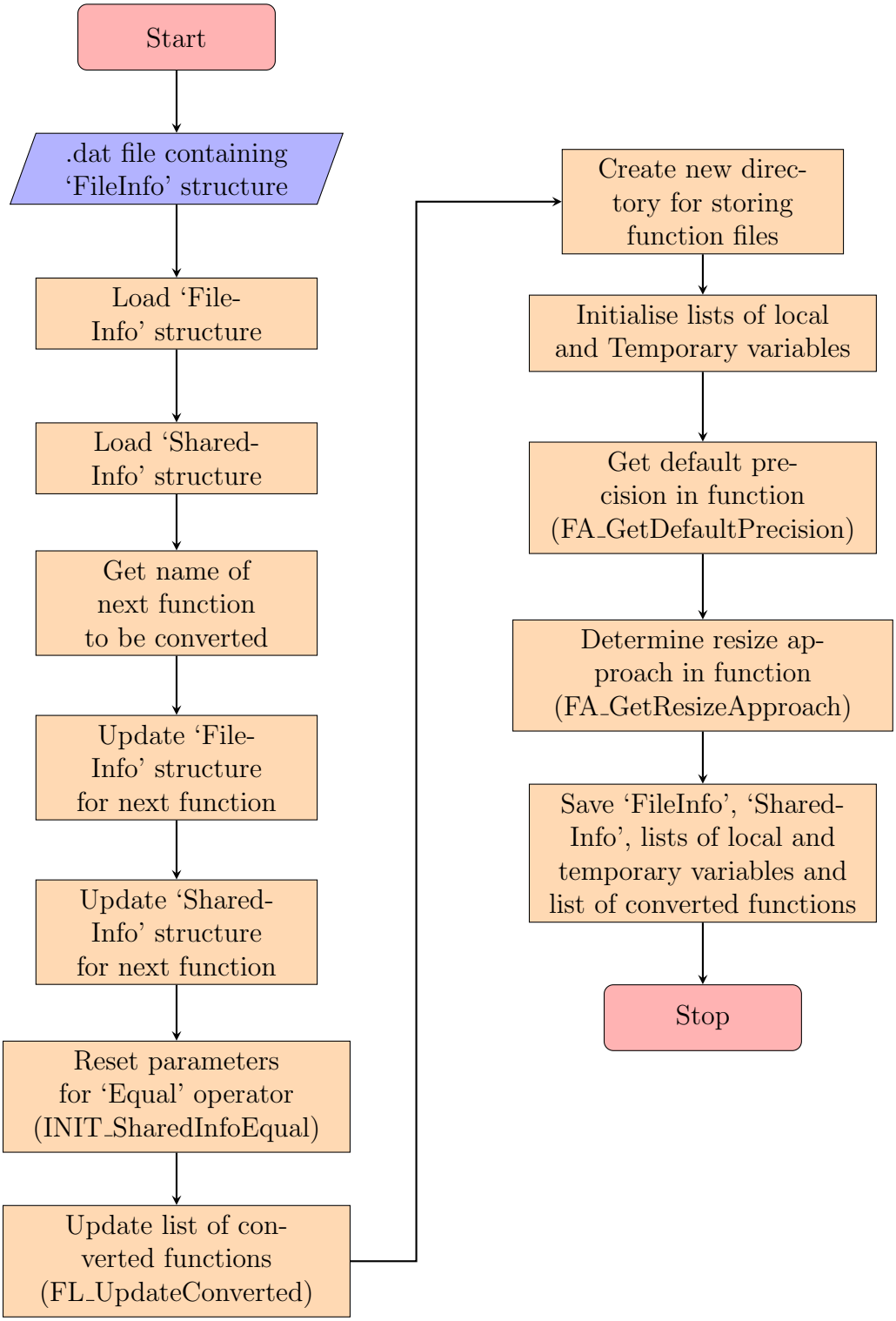
;

UpdateSCI2CInfo.sci

Siddhesh Wani

Introduction

This fucntion gets next scilab function to be converted and updates its information in 'Fileinfo' structure.



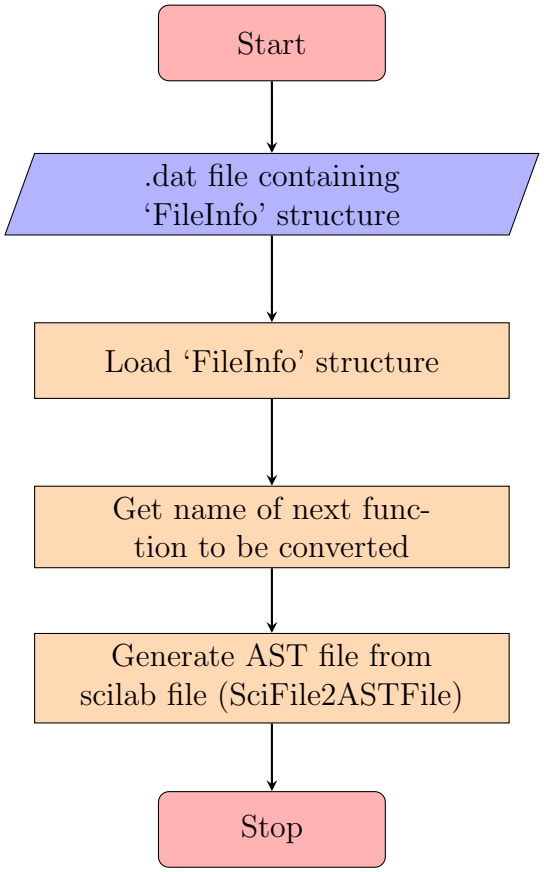
;

AST_GetASTFile.sci

Siddhesh Wani

Introduction

Gets next function from 'FileInfo' structure and calls 'SciFile2ASTFile' with appropriate arguments to generate AST file file from scilab file.



;