FOSSEE Scilab-ROS Interface

Developer's Manual

toolbox@scilab.in

July 4, 2017

Contents

1 Introduction

FOSSEE Scilab-ROS Interface is a toolbox in Scilab maintained and developed by FOSSEE[1](Free and Open Source Software in Education), IIT Bombay[2]. It provides the following functionalities:

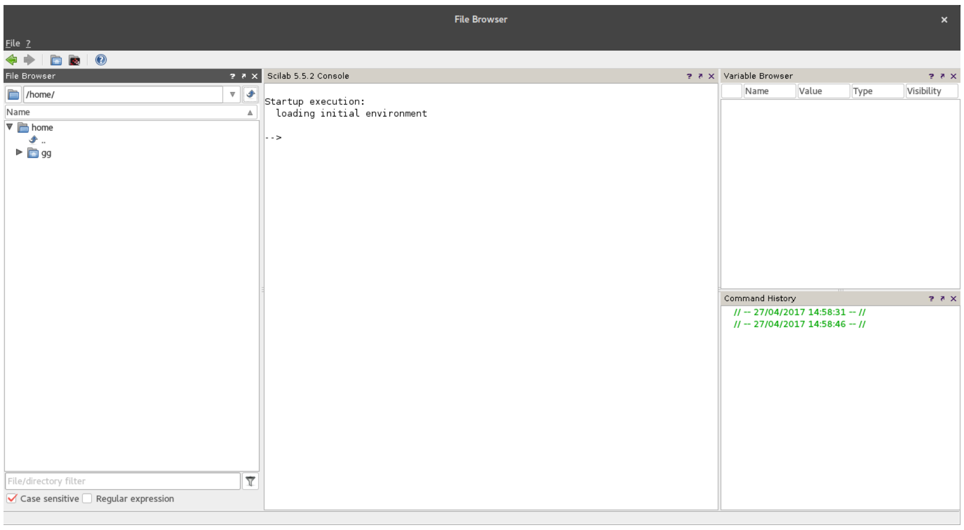
- Execute a ROS command from Scilab.
- Get the parameter of a ROS Subscriber/Publisher from Scilab.
- Get value of a rostopic from Scilab.

It also serves as a data transfer medium between ROS and Scilab.
Scilab is a open-source numerical computational package licensed under GPLv2 license which has broad applications in educational and engineering domains. It is a competitive alternative to Matlab and Octave. Scilab was initially maintained and developed by INRIA[3](French Institute for Research in Computer Science and Automation). Scilab consortium was formed in June 2010 which now handles Scilab development. FOSSEE Scilab-ROS Interface uses Scilab-C API to communicate to a ROS server running in the background. More information about the API could be found here. Scilab-C API[4]. Scilab also provides API functions to call libraries from C++ and FORTRAN. However, the scope of the programming of this toolbox is limited to C.

1.1 Scilab

Scilab, as mentioned above is a numerical computational software. It can be downloaded from the Scilab website[5] or from the respective repositories in case of Linux or macOS. The standard IDE of Scilab is given in figure 1.1.It contains a file browser,variable browser, command history and the console.

[1]https://fossee.in/
[2]http://iitb.ac.in/
[3]https://www.inria.fr/en/
[4]https://help.scilab.org/docs/5.4.1/ru_RU/api_scilab_getting_started.htm
[5]http://www.scilab.org/download/latest



```
Startup execution:
  loading initial environment

-->
```

```
// -- 27/04/2017 14:58:31 -- //
// -- 27/04/2017 14:58:46 -- //
```

Figure 1.1: Scilab GUI

Scilab also provides a command line interface which can be run by `scilab-cli` from the terminal.
Native scilab provides only limited functions. For advanced and specific functions, the user has to download toolboxes which are scilab packages. The scilab package manager is called ATOMS(AuTomatic mOdules Management for Scilab).The available scilab toolboxes on ATOMS can either accessed through the ATOMS website[6] or by clicking Applications»Module manager in the Scilab menu bar.

NOTE
It is good practise to run `atomsSystemUpdate()` in scilab console to update ATOMS upon fresh install. In case, you are using Scilab 5.5, execute `atomsRepositoryAdd('http://atoms.scilab.org/5.5')` instead of the previous command

1.2 Interface Libraries

FOSSEE Scilab-ROS Interface mainly uses Scilab, C and ROS. The Scilab-C API would work automatically if Scilab is installed.

1.3 Downloading the Files

SRI can be downloaded onto your system in the following ways:

[6]https://atoms.scilab.org/

- From github : clone from the github repository, `git clone https://github.com/saumyadoshi/Scilab-ROS-Toolbox.git`

You can load the Interface in Scilab by going to the interface root directory and running `exec builder.sce` on the Scilab console. These commands compiles and loads the files onto Scilab memory. `exec cleaner.sce` will delete all the compiled binary files.

1.4 Prerequisites

The user is expected to have a working knowledge of Scilab, C and ROS. You should atleast know the basic commands of ROS so that the same could be implemented in Scilab as well. In addition to this, you should have good command over Linux and familiar with its workflows.

1.5 Purpose of document

This manual is made with the idea of making users familiar with FOSSEE Scilab-ROS Interface and also give them a preliminary idea about its working so that they can contribute to its development as well.

This Developer's Manual consists of 7 chapters. chapter 2 deals with the associated files briefly. It is followed by chapter 3 which explains thoroughly how to install ROS Indigo. chapter 4 explains in details the Scilab-C API file which loads the ROS interface into Scilab. chapter 5 explains on how to load and execute the interface into Scilab console. chapter 6 enlists the most likely errors and how to avoid them. chapter 7 lists down the bugs of the current version and gives the developers a brief idea on how they can further develop the interface.

2

Interface Files

For this interface, currently we are trying to create a junction between the rosmaster and Scilab. Hence, the number of functions are very few, therefore, the developers have used `ilib\_build` function to build the required files rather than create a toolbox. For more details about `ilib\_build`, check the help[1] page.

FOSSEE Scilab-ROS Interface has 2 files currently. One is the builder.sce file which builds another file ros_toolbox.c using `ilib\_build` and also executes the loader.sce file generated by the previous build command. The ros_toolbox.c file would be explained in the upcoming chapters.

[1]https://help.scilab.org/docs/5.4.0/fr_FR/ilib_build.html

3

Installing ROS

3.1 Introduction

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

3.2 Ubuntu install of ROS Indigo

If you need to install from source (**not recommended**), please see *source*[1] *(download-and-compile) installation instructions* .

3.2.1 Installation

ROS Indigo **ONLY** supports Saucy (13.10) and Trusty (14.04) for debian packages.

Configure your Ubuntu repositories

Configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse." You can follow the Ubuntu guide[2] for instructions on doing this.

Setup your sources.list

Setup your computer to accept software from packages.ros.org.

- sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

[1]http://wiki.ros.org/indigo/Installation/Source [2]https://help.ubuntu.com/community/Repositories/Ubuntu

Set up your keys

- sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116

If you experience issues connecting to the keyserver, you can try substituting hkp://pgp.mit.edu:80 or hkp://keyserver.ubuntu.com:80 in the previous command.

Installation

First, make sure your Debian package index is up-to-date:

- sudo apt-get update
  - If you are using Ubuntu Trusty **14.04.2** and experience dependency issues during the ROS installation, you may have to install some additional system dependencies.
    - **Do not install these packages if you are using 14.04, it will destroy your X server**:

      sudo apt-get install xserver-xorg-dev-lts-utopic mesa-common-dev-lts-utopic libxatracker-dev-lts-utopic libopenvg1-mesa-dev-lts-utopic libgles2-mesa-dev-lts-utopic libgles1-mesa-dev-lts-utopic libgl1-mesa-dev-lts-utopic libgbm-dev-lts-utopic libegl1-mesa-dev-lts-utopic

      Do not install the above packages if you are using 14.04, it will destroy your X server

    Alternatively, try installing *just* this to fix dependency issues:

    - sudo apt-get install libgl1-mesa-dev-lts-utopic

  For more information on this issue see this answers.ros.org thread[3] or this launchpad issue[4]

There are many different libraries and tools in ROS. We provided four default configurations to get you started. You can also install ROS packages individually.

- **Desktop-Full Install** : ROS, rqt[5], rviz[6], robot-generic libraries, 2D/3D simulators and 2D/3D perception

[3]http://answers.ros.org/question/203610/ubuntu-14042-unmet-dependencies/

[4]https://bugs.launchpad.net/ubuntu/+source/mesa-lts-utopic/+bug/1424059

[5]http://wiki.ros.org/rqt

[6]http://wiki.ros.org/rviz

Indigo uses Gazebo **2** which is the default version of Gazebo on Trusty and is recommended. If you would like to instead use a newer version of Gazebo (5, 6 or 7), refer to these instructions[7] on the Gazebo site. Note that installing a newer version of Gazebo will require you to build dependent packages (such as turtlebot_gazebo) to be built from source. See also Using a specific Gazebo version with ROS[8].

– sudo apt-get install ros-indigo-desktop-full

  or click here

Initialize rosdep

Before you can use ROS, you will need to initialize rosdep. rosdep enables you to easily install system dependencies for source you want to compile and is required to run some core components in ROS.

sudo rosdep init
rosdep update

Environment setup

It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched:

echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
source ~/.bashrc

If you have more than one ROS distribution installed, ~/.bashrc must only source the setup.bash for the version you are currently using.

If you just want to change the environment of your current shell, you can type:

source /opt/ros/indigo/setup.bash

If you use zsh instead of bash you need to run the following commands to set up your shell:

echo "source /opt/ros/indigo/setup.zsh" >> ~/.zshrc
source ~/.zshrc

[7]http://gazebosim.org/tutorials?tut=ros_wrapper_versions

[8]http://gazebosim.org/tutorials?tut=ros_wrapper_versions#Usingaspecific

4

ros_toolbox.c file

4.1 Introduction

This file is the file developed using the Scilab-C API. A header file is included for the same.

This code receives a command as an input from Scilab (GUI/CLI), runs it at the terminal, and the output from the terminal is stored in a string, which in turn, is transferred to Scilab as a string matrix and is stored in the output variable.

```
#include "api_scilab.h"
#include "BOOL.h"
#include <localization.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
/*
    ===================================================================
     */
int ros_toolbox(char *fname, unsigned long fname_len)
{


/*
 */
   //==========================INPUT
       ===================================//
   SciErr sciErr;
   int i,j;
   int iLen    = 0;
   //variable info
   int iRows      = 0;
   int iCols      = 0;
```

```
int* piAddr     = NULL;
int* piLen      = NULL;
int iRowsOut    = 1;
int iColsOut    = 1;
char* pstOut    = NULL;
char** pstData = NULL;
//check input and output arguments

   CheckInputArgument(pvApiCtx, 1, 1);
   CheckOutputArgument(pvApiCtx, 1, 1);

//get variable address
sciErr = getVarAddressFromPosition(pvApiCtx, 1, &piAddr);
if(sciErr.iErr)
{
   printError(&sciErr, 0);
   return 0;
}

//first call to retrieve dimensions
sciErr = getMatrixOfString(pvApiCtx, piAddr, &iRows, &iCols,
     NULL, NULL);
if(sciErr.iErr)
{
   printError(&sciErr, 0);
   return 0;
}

piLen = (int*)malloc(sizeof(int) * iRows * iCols);

//second call to retrieve length of each string
sciErr = getMatrixOfString(pvApiCtx, piAddr, &iRows, &iCols,
     piLen, NULL);
if(sciErr.iErr)
{
   printError(&sciErr, 0);
   return 0;
}

pstData = (char**)malloc(sizeof(char*) * iRows * iCols);
for(i = 0 ; i < iRows * iCols ; i++)
{
   pstData[i] = (char*)malloc(sizeof(char) * (piLen[i] + 1))
       ;//+ 1 for null termination
}

//third call to retrieve data
sciErr = getMatrixOfString(pvApiCtx, piAddr, &iRows, &iCols,
     piLen, pstData);
if(sciErr.iErr)
{
   printError(&sciErr, 0);
   return 0;
}
```

```
    //==========================================================================//

char *command = pstData[0];
//=======================TERMINAL OPERATION
    ============================//

char a[100][10000];
int size = 0;
FILE *fpipe = (FILE*)popen(command,"r");
char line[10000];
while(fgets(line, sizeof(line), fpipe)){
   strcpy(a[size],line);
   size++;
   if(strncmp(line,"---",3)==0){
      break;
   }
   //sciprint("%s", line);

}

pclose(fpipe);



//==========================================================================//

//================OUTPUT STRING WRITING
    ====================================
int iRows2     = size;
int iCols2     = 1;
char** pstData2   = NULL;
pstData2       = (char**)malloc(sizeof(char*) * iRows2 *
    iCols2);

//printf("%d", size);
int k;
for(k=0;k<size;k++){
   pstData2[k] = a[k];
}

sciErr = createMatrixOfString(pvApiCtx, nbInputArgument(
    pvApiCtx) + 1, iRows2, iCols2, pstData2);
if(sciErr.iErr)
{
   printError(&sciErr, 0);
   return 0;
}

//free container
free(pstData2);
free(pstData);
//assign allocated variables to Lhs position
   AssignOutputVariable(pvApiCtx, 1) = nbInputArgument(
                pvApiCtx) + 1;

        return 0;
}
/*
    ==========================================================================
     */
```

4.2 Basic Explanation

As one can see in the code above, the input arguments are taken from the Scilab console. Only one argument is acceptable which should be passed in double inverted quotes. Once the user has done so, it checks for any possible error, else creates a string matrix, accepts the input and finally passes and stored the command in the variable 'command'.

This command is then passed on to the terminal using a piped file pointer. Once it is executed, it is stored in a 2-D character array i.e. a string array till one reaches the end of line. One thing to notice here is that we break the loop to copy data if we encounter "—" string. The reason being that in case the data is real time and keeps adding after a certain period of time, then the command would not return. Eventually, as the data keeps piling up, it would through a segmentation fault due to stack overflow crashing the Scilab console as well.

In order to avoid this, we break the loop. Real-time data output of a rostopic appends the string "—" after each instant. So, we break the loop and return the data. Thus, as far as the scope of this interface is concerned, the real-time data of the last instant could only be obtained.

After this operation, we create an output matrix, copy the data stored in the string array and return the matrix back to the scilab output argument variable.

5

Implementation

5.1 Pre-requisites

Since it is assumed that the user has a working knowledge of ROS, it is required that the user starts a rosmaster by executing the command 'roscore' in the terminal. Then, the user can communicate with this rosmaster using Scilab via this interface. Also, it is necessary that the user **starts Scilab via terminal only**.

5.2 Building files

The user has to ensure that both the files 'builder.sce' and 'ros_toolbox.c' are in the same directory. After that on the Scilab Console, the user has to navigate to that directory and exceute the 'builder.sce' file with the command 'exec builder.sce'. This command would build the API file 'ros_toolbox.c' using ilib_build and also execute the generated 'loader.sce' file.

5.3 Entering commands

Once the roscore is working and the builder file is executed, the user can start executing the ROS commands. For this, the user has to enter the required command in this way:

ros('<command>')

If the user is expecting some output, then in this way:

OutPutVar = ros('<command>')

The result would be stored in OutPutVar variable.

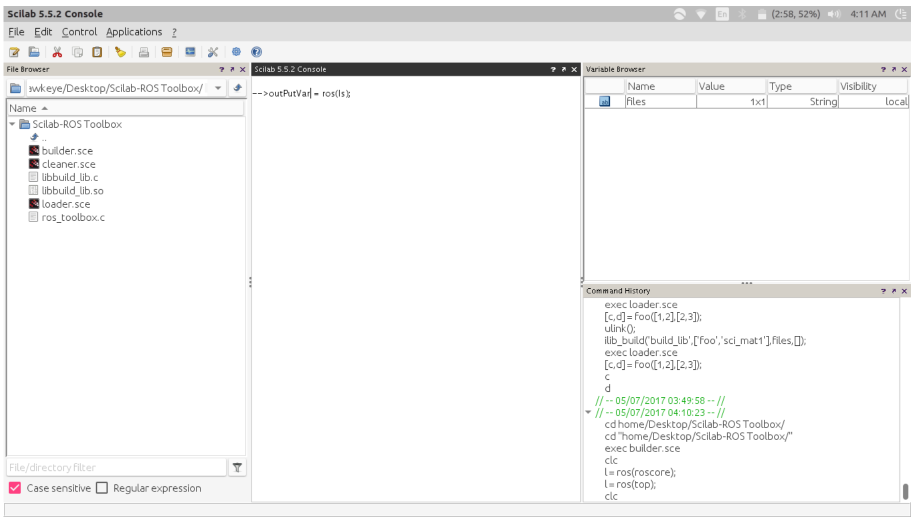
NOTE: If any of the steps are not implemented correctly, for example, inverted quotes are not inserted, then there is a possibility that Scilab would crash.

6

Error Handling

6.1 Possible Errors and how to avoid them

1. Segmentation fault (core dumped):



```
-->outPutVar = ros(ls);
```

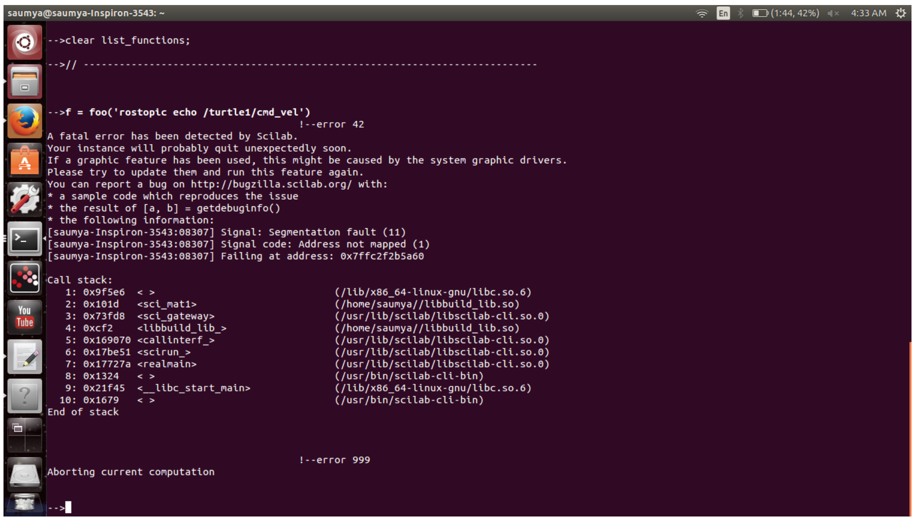
Name	Value	Type	Visibility
Files	1x1	String	local

```
    exec loader.sce
    [c,d] = foo([1,2],[2,3]);
    ulink();
    ilib_build('build_lib',['foo','sci_mat1'],files,[]);
    exec loader.sce
    [c,d] = foo([1,2],[2,3]);
    c
    d
  // -- 05/07/2017 03:49:58 -- //
  // -- 05/07/2017 04:10:23 -- //
    cd home/Desktop/Scilab-ROS Toolbox/
    cd "home/Desktop/Scilab-ROS Toolbox/"
    exec builder.sce
    clc
    l = ros(roscore);
    l = ros(top);
    clc
```

Figure 6.1: Not using quotes

This would throw a segmentation fault(core dumped) error which would crash Scilab.

2. Stack Overflow
 This error is caused when the data size is too large or the real-time data keeps on coming but does not satisfy the break condition pre-defined in the code, then eventually, the memory stack buffer allocated to Scilab overflows, and Scilab crashes giving this error.
(In GUI mode, it would exit throwing a Segmentation Fault. But, if



```
saumya@saumya-Inspiron-3543: ~

-->clear list_functions;

-->// ---------------------------------------------------------------------------



-->f = foo('rostopic echo /turtle1/cmd_vel')
                                          !--error 42
A fatal error has been detected by Scilab.
Your instance will probably quit unexpectedly soon.
If a graphic feature has been used, this might be caused by the system graphic drivers.
Please try to update them and run this feature again.
You can report a bug on http://bugzilla.scilab.org/ with:
* a sample code which reproduces the issue
* the result of [a, b] = getdebuginfo()
* the following information:
[saumya-Inspiron-3543:08307] Signal: Segmentation fault (11)
[saumya-Inspiron-3543:08307] Signal code: Address not mapped (1)
[saumya-Inspiron-3543:08307] Failing at address: 0x7ffc2f2b5a60

Call stack:
   1: 0x9f5e6  < >                            (/lib/x86_64-linux-gnu/libc.so.6)
   2: 0x101d   <sci_mat1>                     (/home/saumya//libbuild_lib.so)
   3: 0x73fd8  <sci_gateway>                  (/usr/lib/scilab/libscilab-cli.so.0)
   4: 0xcf2    <libbuild_lib_>                (/home/saumya//libbuild_lib.so)
   5: 0x169070 <callinterf_>                  (/usr/lib/scilab/libscilab-cli.so.0)
   6: 0x17be51 <scirun_>                      (/usr/lib/scilab/libscilab-cli.so.0)
   7: 0x17727a <realmain>                     (/usr/lib/scilab/libscilab-cli.so.0)
   8: 0x1324   < >                            (/usr/bin/scilab-cli-bin)
   9: 0x21f45  <__libc_start_main>            (/lib/x86_64-linux-gnu/libc.so.6)
  10: 0x1679   < >                            (/usr/bin/scilab-cli-bin)
End of stack



                                          !--error 999
Aborting current computation


-->
```

Figure 6.2: Stack Overflow

the exact error is to be known, execute the same operation in scilab-cli mode.)

3. Java Error
   Sometimes, the user might encounter a Java error, for example, an Abstract Window toolkit (java.awt error), in this case, it is advised to implement the same operation in scilab-cli mode.

7

Further Work

1. Data Parsing
   As explained in the code above, currently the data is returned in the form of a String Matrix to the Scilab console. If computations are to be carried out upon this data, then it is necessary to parse the data to the required data type.

2. Real-Time Data
   As mentioned in the chapter Error Handling, there are several complications when the data returned is real time. First of all, real-time data is not returned back to Scilab. The instance when ROS gives a string "—" to separate two instances, the loop breaks and the data is returned. In possible scenarios, when ROS does not print that string in real-time execution, the interface would crash once the data size surpasses stack memory allocated to Scilab. Hence, much development is possible with respect to real time data analysis.
   Probably using some ROS libraries, the data could first be extracted (for example : rosbridge_suite[1] library provides a JSON API to ROS functionality for non-ROS programs.) After this step is comlpeted, further analysis of the data can carried out using Scilab.

[1]http://wiki.ros.org/rosbridge_suite