




Programming Environment for Lua

Copyright(C) 2010-2021 Pr. Sidi HAMADY

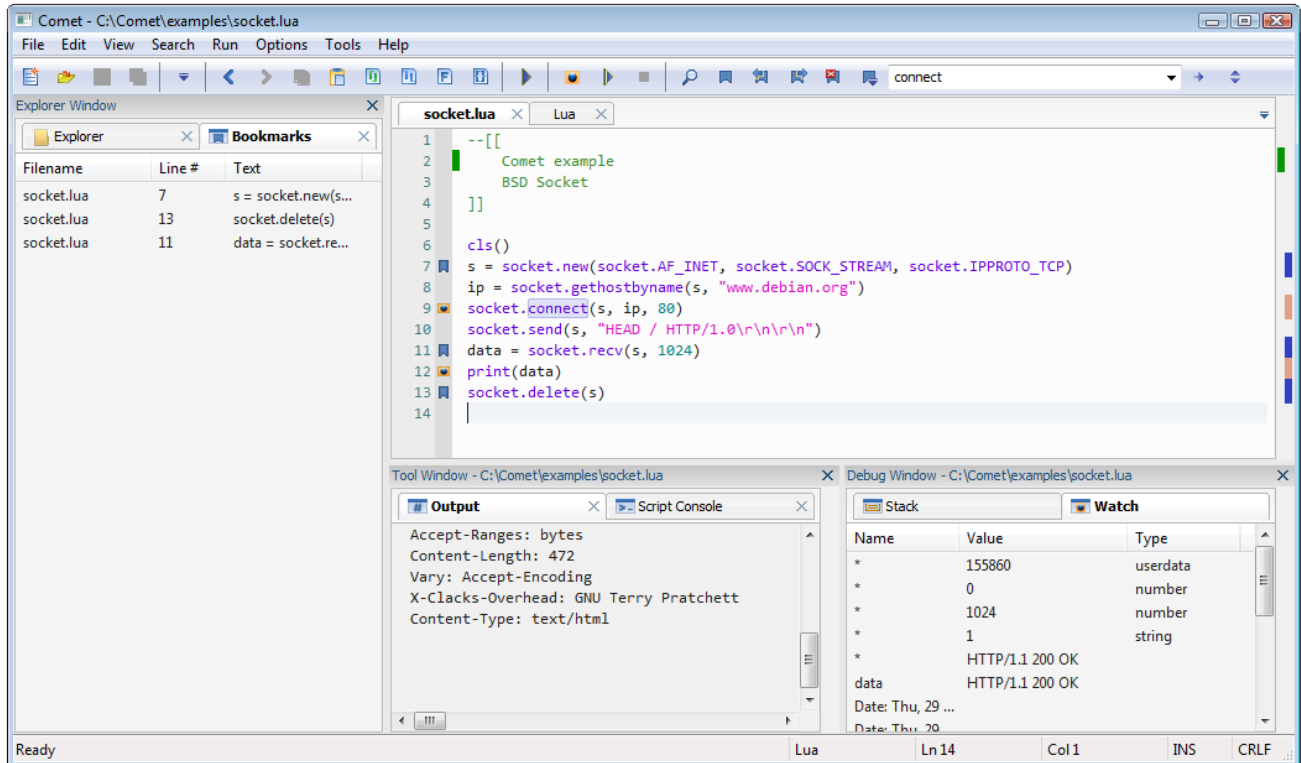
<http://www.hamady.org>

Contents

Comet 	1
1. Introduction.....	3
2. Install	4
3. Quick Start Guide.....	5
4. Comet as a general-purpose editor.....	7
5. GUI Functions	8
6. Time functions	8
7. Math functions	8
8. BSD Socket	10
9. C Modules.....	14
10. Lua IO File Functions.....	17
11. Lua IO Functions	17
12. Debugging.....	17
13. Specifications.....	18
14. Copyright	18

1. INTRODUCTION

Comet is a programming environment for the Lua scripting language. It is available for Android, Linux and Windows with built-in Lua scripting engine, full-featured editor with syntax highlighting, code completion, inline documentation, script console, Lua samples and code snippets, debugging capabilities, etc.



The main goal of Comet is to provide an **easy-to-use development environment for the Lua programming language on Android, Linux and Windows**. It integrates the Lua scripting engine with all Lua functionalities. Within the Comet environment, you can develop algorithms for science and engineering with one of the most elegant and fast scripting languages.

To learn the Lua programming language, you can visit the Lua official website: <http://www.lua.org>

Excellent tutorials, covering all Lua aspect from basics to advanced programming techniques, can be found here: <http://lua-users.org/wiki/TutorialDirectory>

Comet can also be used as a **general purpose full-featured editor supporting C/C++, Bash/Text, Python, Octave, Fortran, LaTeX and Makefile** with configurable tools (e.g. to run a compiler or a bash script).

2. INSTALL

- For **portable version** on **Windows** and **Linux**: Download **comet_windows_64bit.zip** (Windows 64bit) from <http://www.hamady.org> or **comet_windows_32bit.zip** (Windows 32bit) or **comet_linux_64bit.tgz** (Linux 64bit) or **comet_linux_32bit.tgz** (Linux 32bit), unzip/untar in any location (USB key or memory stick for example) and run **comet.exe** (Windows) or **comet** (Linux) in the bin directory.
- For **setup version** (on **Windows 64bit**): Download **comet_windows_64bit.msi**, double-click on the setup file and it will install Comet.

If you need only the Lua editor, without the numerical and visualization modules, you can download Comet N, the Comet stripped version:

For portable version (64bit): Download:

cometn_windows_64bit.zip (Windows 64bit)

cometn_linux_64bit.tgz (Linux 64bit)

unzip/untar in any location and run **comet.exe** or **comet** in the bin directory.

For 32bit portable version: Download:

cometn_windows_32bit.zip (for Windows 32bit)

cometn_linux_32bit.tgz (for Linux 32bit)

Under Ubuntu/Debian 64bit, if you encounter error such as:

./comet: error while loading shared libraries: libpng12.so.0

Install libpng12 as follows:

cd /tmp/

wget http://mirrors.kernel.org/ubuntu/pool/main/libp/libpng/libpng12-0_1.2.54-1ubuntu1_amd64.deb

sudo dpkg -i ./libpng12.deb

sudo rm ./libpng12.deb

If you encounter (in Ubuntu/Debian 64bit) error such as:

Failed to load module "canberra-gtk-module"

Install libcanberra-gtk-module as follows:

sudo apt install libcanberra-gtk-module libcanberra-gtk3-module

- For **Android**: Install from Google Play.

To know if a new version is available, click *Menu/Help/Check for Update...*

3. QUICK START GUIDE

With the Comet toolbar, you can open, save, navigate in the document with bookmarks, run/debug script.

You can perform the same actions, and more, with the menu.

Comet includes code examples and snippets (*Help/Examples* and *Edit/Insert Snippet* menu). With code snippets, you can easily insert some pieces of code in your script for better productivity.

For Android: If you want to change the font size, select menu/Settings. You can create a new script or save the current script to another location, by selecting menu. If you want to automatically run a script on startup (AutoRun), select menu/Settings/AutoRun and choose your script in the file explorer. If you long click the AutoRun button in menu/Settings, the AutoRun feature will be disabled. You can enable or disable the soft keyboard by selecting menu/Keyboard.

You can print text on the output area by using the Lua standard `print` or `io.write` function. To clear the output area, use the `cls()` function or select *Clear* in the context menu.

For Android: You can set the output area height by selecting S (small), M (median) or L (large) in Menu / Settings (Output).
As for Windows and Linux, Comet for Android has two graphical themes: light and dark, and syntax highlighting functionality. You can switch between themes, enable or disable syntax highlighting in menu/Settings.

- ✓ You can create a new script or save the current script to another location, by selecting **File/New** (*Menu/New* on Android) or *File/Save As* (*Menu/Save As* on Android) menu or the toolbar corresponding buttons.
- ✓ You can **print text** on the output area by using the Lua standard **print** function (or the **io.write** function). To clear the output area, use the **cls()** function.
- ✓ You can set **bookmark** in you code and then navigate easily by clicking click Add/Remove Bookmark, Next or Previous Bookmark (either Menu/Search or the correspond toolbar buttons). Bookmarks are saved with your document and are restored next time you open it. The list of all active bookmarks can be displayed by selecting *View/Bookmarks* in the menu. The bookmarks, modification and search markers can be saved (or not) alongside the document by checking (or unchecking) the corresponding option: *Options/Save Navigation Info* menu.
- ✓ The **editor options** (Font, Syntax highlighting, AutoCompletion, Call Tips, Colors, Indentation and long lines indicator, Folding ...) can be easily modified in the *Options* menu.
- ✓ You can **view** end of lines, white spaces, file explorer, output window, script console, tool bar, status bar ... in the *View* menu.
- ✓ The **editor toolbar** includes, in addition to the commonly used commands, an integrated edit zone to (i) navigate through the current document functions (or sections for LaTeX): just drop down the list and select a function; (ii) to search in the document: just type a text and click the toolbar button to the right (or press ENTER); (iii) go to a line: just type the line number and click the toolbar button

to the right (or press ENTER). Note that the document functions list can be manually updated by clicking the 'Update List' toolbar button (this list is created when the document is loaded or executed). To reset the list, right click in it and choose 'Reset' in the popup menu.

- ✓ The **editor infobar** on the right border includes the main navigation markers (modification, find results, bookmarks ...), useful for long documents. Click on a marker in the infobar to go to the corresponding line. The infobar can be enabled or disabled by clicking *Options/View Infobar Markers* menu.
- ✓ Comet's **execution mode** is designed to perform long calculations, without user interaction. This feature makes it suitable for math and algorithms development.

For Android: you can activate the long calculations mode, which set the device processor to stay awake during script execution, by selecting Menu/Settings.

Comet can be executed from the command line:

```
$ comet -run input [-out outfile] [-show]
```

input may be a filename or a double-quoted expression and **outfile** is the output filename:

```
$ comet -run "print(sin(math.pi/4))"
```

```
$ comet -run cometin.lua -out cometout.txt
```

Note that the plot functions and other GUI functions are only available in the Comet GUI.

- ✓ **Comet** integrates a **Lua script console** (Menu View / Script Console). You can use it as an interactive programming console to perform quick tests for example. To use the script console, simply write a Lua valid statements (example: `x = math.pi/3; y = sin(x); print(y);`), and press ENTER. You can enter multi-line statements: just press SHIFT+ENTER after each line and ENTER to run. You can use any standard Lua function or Comet module.

First example:

Type the following script in the editor...

```
-- Lua first script
cls()
a = math.cos(math.pi/4)
print("a = ", a)
```

... and click Run (or press F12 on Windows and Linux).

4. COMET AS A GENERAL-PURPOSE EDITOR

Comet can be used as a general purpose **full-featured editor supporting C/C++, Bash/Text, Python, Octave, Fortran, LaTeX and Makefile** with configurable tools (e.g. to run a compiler or a bash script). The document language is automatically selected using the file extension. For unknown extension, you can manually select the document language in *File/Current File/File Type* menu. To configure the tools used, for example, to compile and execute code, select *Options/External Tools* menu. Comet supports **configurable tools for C/C++, Python, LaTeX and Makefile**.

To configure a specific tool, select menu *Options/External Tools*:

- (i) For Windows, you can create a batch file (named **build.bat** for example), put the command in this file and add the batch command **call build.bat** in the *External Tools* dialog after selecting the corresponding file type (example: **LaTeX**). The following example gives a typical **build.bat** content for building LaTeX document under Windows using MiKTeX and the Sumatra PDF viewer:

```
@echo off
@del /f /q myreport.pdf >nul 2>&1
latexmk -pdf -pvc- -halt-on-error myreport.tex
if %errorlevel% equ 0 (start "" "SumatraPDF.exe" myreport.pdf) else (exit /b 1)
```

- (ii) For Linux, similarly you can create a bash file (named **build.sh** for example), put the command in this file (example: `pdflatex -halt-on-error -file-line-error myreport.tex`) and add **./build.sh** in the *External Tools* dialog after selecting the corresponding file type (example: **LaTeX**).
- (iii) For Python (for both Windows or Linux), just put a command like this (replace with your installed Python interpreter): **C:\Python\python.exe -u %s.py** (Windows) or **python -u %s.py** (Linux).

Comet includes a **user-defined syntax highlighting template** that can be customized for a specific language or file type: click *Options/User-defined Syntax* menu, enter the corresponding file extension (without dot) and the space-separated list of keywords to highlight. You can save the user-defined syntax highlighting template to a file (button *Save...*) or load the template from a file (button *Load...*).

5. GUI FUNCTIONS

To interact graphically with the user, one can use the following two GUI functions:

```
gui.alert(msg)           -- show an alert dialog box
answer = gui.ask(msg)    -- display Yes-No dialog.
                           -- Returns 1 if Yes selected, 0 otherwise
```

Example:

```
-- GUI
answer = gui.ask("Save calculation results?")
if answer == 1 then
    -- ...
    gui.alert("Calculation results saved.")
end
```

6. TIME FUNCTIONS

```
time.tic()               -- sets the wall-clock timer
time.toc()               -- returns the number of milliseconds elapsed
                           -- since the last tic() call
time.sleep(n)            -- sleeps for n milliseconds
time.format(n)           -- formats a duration (integer in milliseconds)
                           -- into string
```

Example:

```
-- Time
time.tic()
time.sleep(200)
dt = time.toc()
print(time.format(dt))
```

7. MATH FUNCTIONS

With Comet, the math functions are mapped to global functions (e.g.: you can use `cos` or `math.cos`).

Example:

```
-- Math
cls()
a = cos(pi/4)
print("result = ", a)
```


Below is a summary of the Lua math functions:

<code>math.abs</code>	<code>math.acos</code>	<code>math.asin</code>
<code>math.atan</code>	<code>math.atan2</code>	<code>math.ceil</code>
<code>math.cos</code>	<code>math.cosh</code>	<code>math.deg</code>
<code>math.exp</code>	<code>math.floor</code>	<code>math.fmod</code>
<code>math.frexp</code>	<code>math.huge</code>	<code>math.ldexp</code>
<code>math.log</code>	<code>math.log10</code>	<code>math.max</code>
<code>math.min</code>	<code>math.modf</code>	<code>math.pi</code>
<code>math.pow</code>	<code>math.rad</code>	<code>math.random</code>
<code>math.randomseed</code>	<code>math.sin</code>	<code>math.sinh</code>
<code>math.sqrt</code>	<code>math.tanh</code>	<code>math.tan</code>

And a summary of the Comet global math functions:

<code>abs</code>	<code>acos</code>	<code>asin</code>
<code>atan</code>	<code>atan2</code>	<code>ceil</code>
<code>cos</code>	<code>cosh</code>	<code>deg</code>
<code>exp</code>	<code>floor</code>	<code>fmod</code>
<code>frexp</code>	<code>huge</code>	<code>ldexp</code>
<code>log</code>	<code>log10</code>	<code>max</code>
<code>min</code>	<code>modf</code>	<code>pi</code>
<code>pow</code>	<code>rad</code>	<code>random</code>
<code>randomseed</code>	<code>sin</code>	<code>sinh</code>
<code>sqrt</code>	<code>tanh</code>	<code>tan</code>

NB: Lua gives the Napierian logarithm the name `log` and the decimal logarithm is named `log10`, as in C language.

8. BSD SOCKET

The socket namespace includes BSD-like network functions:

```
s = socket.new(af, type, proto)    -- creates socket where:  
                                     -- af: family (socket.AF_INET by default)  
                                     -- type: type (socket.SOCK_STREAM by default)  
                                     -- proto: protocol (def: socket.IPPROTO_TCP)  
                                     -- returns the socket identifier
```

```
ok = socket.bind(s, addr, port)    -- binds socket s, where:  
                                     -- s: socket identifier  
                                     -- addr: address (ex: socket.INADDR_ANY)  
                                     -- port: port to bind to  
                                     -- returns true on success
```

```
ok = socket.listen(s, backlog)    -- listens on socket, where:  
                                     -- s: socket identifier  
                                     -- backlog: maximum queue length  
                                     -- returns true on success
```

```
ok = socket.connect(s, addr, port)  
                                     -- connects socket, where:  
                                     -- s: socket identifier  
                                     -- addr: address  
                                     -- port: port to connect to  
                                     -- returns true on success
```

```
sa = socket.accept(s, addr, port)  
                                     -- accepts connection and create new socket:  
                                     -- s: socket identifier  
                                     -- returns the new socket identifier sa
```

```
ok = socket.timeout(s, to)  
                                     -- sets the recv and send timeout, where:  
                                     -- s: socket identifier  
                                     -- timeout in milliseconds  
                                     -- returns true on success
```

ok = socket.setsockopt(s, opt, val)

```
-- sets socket option, where:  
-- s: socket identifier  
-- option to set (ex: socket.SO_SNDTIMEO)  
-- val: option value  
-- returns true on success
```

ip = socket.getpeername(s)

```
-- gets the socket s peer ip address  
-- s: socket identifier  
-- returns peer ip address
```

hn = socket.gethostbyaddr(s, addr)

```
-- gets the host name for ip address  
-- s: socket identifier  
-- addr: ip address  
-- returns host name
```

ha = socket.gethostbyname(s, name)

```
-- gets the ip address for host name  
-- s: socket identifier  
-- name: host name  
-- returns ip address
```

ha = socket.getsockname(s)

```
-- gets the socket name, where:  
-- s: socket identifier  
-- name: host name  
-- returns socket name
```

ok = socket.send(s, data, flags)

```
-- sends data, where:  
-- s: socket identifier  
-- data: data to send  
-- flags: optional flags  
-- returns true on success
```

ok = socket.sendto(s, addr, port, data, flags)

- sends data, where:
- s: socket identifier
- addr: address
- port: port to send to
- data: data to send
- flags: optional flags
- returns true on success

data = socket.recv(s, datasize, flags)

- receives data, where:
- s: socket identifier
- datasize: data size to be received
- flags: optional flags
- returns received data

data = socket.recvfrom(s, addr, port, datasize, flags)

- receives data from, where:
- s: socket identifier
- addr: address
- port: port to receive from
- datasize: data size to be received
- flags: optional flags
- returns received data

errf = socket.iserr(s)

- gets the error flag:
- s: socket identifier
- returns true if error flag set

errm = socket.geterr(s)

- gets the error message:
- s: socket identifier
- Returns the error message, if any

socket.shutdown(s)

- shutdowns socket, where:
- s: socket identifier

socket.delete(s)

- delete socket and free resources, where:
- s: socket identifier

Example:

```
-- BSD Socket
cls()
s = socket.new(socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_TCP)
ip = socket.gethostbyname(s, "www.debian.org")
socket.connect(s, ip, 80)
socket.send(s, "HEAD / HTTP/1.0\r\n\r\n")
data = socket.recv(s, 1024)
print(data)
socket.delete(s)
```

9. C MODULES

In addition of the built-in modules, **Comet includes two useful C modules:**

- (i) **lpeg** pattern matching module: documentation: <http://www.inf.puc-rio.br/~roberto/lpeg/>

```
-- http://www.inf.puc-rio.br/~roberto/lpeg/
local lpeg = require("lpeg")

-- matches a word followed by end-of-string
p = lpeg.R"az"^1 * -1

print(p:match("hello"))      --> 6
print(lpeg.match(p, "hello")) --> 6
print(p:match("1 hello"))    --> nil
```

- (ii) **lmapm** library for Arbitrary Precision Math (MAPM) by Michael C. Ring, modified Lua interface from lmapm by Luiz Henrique de Figueiredo:

```
-- Library for Arbitrary Precision Math (MAPM) by Michael C. Ring
-- Lua interface by Luiz Henrique de Figueiredo

lmapm = require ("lmapm")

-- lmapm library functions:
-- __add(x,y)      cbrt(x)      mod(x,y)
-- __div(x,y)      ceil(x)      mul(x,y)
-- __eq(x,y)       compare(x,y) neg(x)
-- __lt(x,y)       cos(x)       number(x)
-- __mod(x,y)      cosh(x)      pow(x,y)
-- __mul(x,y)      digits([n])  round(x)
-- __pow(x,y)      digitisin(x) sign(x)
-- __sub(x,y)      div(x,y)     sin(x)
-- __tostring(x)   exp(x)       sincos(x)
-- __unm(x)        exponent(x)  sinh(x)
-- abs(x)          factorial(x)  sqrt(x)
-- acos(x)         floor(x)      sub(x,y)
-- acosh(x)        idiv(x,y)     tan(x)
-- add(x,y)        inv(x)        tanh(x)
-- asin(x)         iseven(x)     tonumber(x)
-- asinh(x)        isint(x)      tostring(x,[n,exp])
-- atan(x)         isodd(x)      version
-- atan2(y,x)      log(x)
-- atanh(x)        log10(x)

print("\nSquare root of 2")
print("math.sqrt(2) ", math.sqrt(2))
print("lmapm.sqrt(2) ", lmapm.sqrt(2))
print(lmapm.version)
```

User C modules: you can write your own C modules to use in Comet.

First, write your C module, as in the following example:

```
/* put the Comet include dir in your include path */
#include <lua.h>
#include <lualib.h>
#include <lauxlib.h>

#ifdef WIN32
#define CMODULE_API __declspec(dllexport)
#else
#define CMODULE_API
#endif

/* compatibility with Lua 5.1 */
#if (LUA_VERSION_NUM == 501)
#define luaL_newlib(L, f) luaL_register(L, "Cmodule", f)
#endif

static int Cmodule_version(lua_State *pLua)
{
    lua_pushstring(pLua, "Cmodule v0.1");
    return 1;
}

static const luaL_Reg Cmodule[] = {
    { "version", Cmodule_version },
    { NULL, NULL }
};

CMODULE_API int luaopen_Cmodule(lua_State *pLua)
{
    luaL_newlib(pLua, Cmodule);
    return 1;
}
```

Second, build your C module with your favorite C compiler, Visual C++ on Windows or gcc or Clang on Linux, for example. Before compiling, add the Comet include directory (Comet Dir/include) in the compiler include path and set the linker to link against the Comet Lua Core library (luacore.dll under Windows and luacore.so under Linux). Under Visual C++, you can generate, for your specific VC version, the lib from luacore.dll and luacore.def by using the dumpbin and lib tools included in Visual C++ as follows:

```
cd C:\Comet\bin\
dumpbin /exports luacore.dll > ..\lib\luacore.txt
cd ..\lib\
echo LIBRARY luacore > luacore.def
echo EXPORTS >> luacore.def
for /f "skip=19 tokens=4" %A in (luacore.txt) do @echo %A >> luacore.def
lib /def:"luacore.def" /out:"luacore.lib" /machine:x64
```

A Visual C++ project example is given in Comet/examples/Cmodule. Under Linux, you can use a Makefile such as (file located in Comet/examples/Cmodule):

```

WORKDIR = `pwd`
CC = gcc
INC =
CFLAGS = -Wall -funwind-tables

INC_RELEASE = -I../..../include $(INC)
CFLAGS_RELEASE = -O2 -fPIC -DUSE_LUAJIT -std=c99 $(CFLAGS)
LDFLAGS_RELEASE = -s -L../..../include -lluacore $(LDFLAGS)
OBJDIR_RELEASE = .
OUT_RELEASE = ./Cmodule.so
OBJ_RELEASE = ./Cmodule.o

all: release

clean: clean_release

before_release:
    test -d . || mkdir -p .

after_release:

release: before_release out_release after_release

out_release: before_release $(OBJ_RELEASE)
    $(LD) -shared $(OBJ_RELEASE) -o $(OUT_RELEASE) $(LDFLAGS_RELEASE)

$(OBJDIR_RELEASE)/Cmodule.o: Cmodule.c
    $(CC) $(CFLAGS_RELEASE) $(INC_RELEASE) -c Cmodule.c -o
$(OBJDIR_RELEASE)/Cmodule.o

clean_release:
    rm -f $(OBJ_RELEASE) $(OUT_RELEASE)

.PHONY: before_release after_release clean_release

```

and build with:

```
LD_RUN_PATH='$ORIGIN' make all
```

Of course, if your module is already built or purchased you can use it as usual with the Lua ad hoc functions.

Lastly, when your module was built as dll (under Windows) or so (under Linux), you can load it and use it under Solis by using the standard Lua functions:

```

local Cmodule = require("Cmodule")
print(Cmodule.version())

```


10. LUA IO FILE FUNCTIONS

The Lua **io** file functions are available in Comet (open, read, write, close and so on.).

Example:

```
-- IO script
cls()
local f = io.open("/mnt/sdcard/Comet/test.txt", "r")
local t = f:read("*all")
f:close()
```

11. LUA IO FUNCTIONS

The input function **io.read** is completely implemented for Windows, Linux and Android.

Example:

```
-- IO read
cls()
io.write("Enter your name: ")
name = io.read("*a")
io.write("Name: ", name)
```

12. DEBUGGING

Comet provides debugging capabilities on Windows and Linux: you can set breakpoints in the code (Menu *Run/Add Breakpoint* or the toolbar corresponding button), view the stack status and current variables set in the Watch window. To start debugging, click Debug (or Menu *Run/Debug* or Ctrl+F12). Debugger can 'Step Into' or 'Step Over' on a call depending on the option set by user in Menu *Run/Step Into* (unchecked for 'Step Over').

13. SPECIFICATIONS

SYSTEM REQUIREMENTS

Comet runs on:

- (i) PC with Windows™ XP, Vista, Windows 7/8/10 32bit or 64bit installed.
- (ii) PC with Linux 32bit or 64bit installed.
- (iii) Smartphone or Tablet with Android 2.2 or later installed.

INSTALL

- For **portable version** on **Windows** and **Linux**: Download *comet_windows_64bit.zip* (Windows 64bit) or *comet_windows_32bit.zip* (Windows 32bit) or *comet_linux_64bit.tgz* (Linux 64bit) or *comet_linux_32bit.tgz* (Linux 32bit), unzip/untar in any location (USB key or memory stick for example) and run *comet.exe* (Windows) or *comet* (Linux) in the bin directory.
- For **setup version** (on **Windows 64bit**): Download *comet_windows_64bit.msi*, double-click on the setup file and it will install Comet.
- For **Android**: Install from Google Play.

14. COPYRIGHT

Comet:

Copyright(C) 2010-2021 Pr. Sidi HAMADY

<http://www.hamady.org>

sidi@hamady.org

Comet is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. Comet is free of charge only for non-commercial use.

Sidi Ould Saad Hamady expressly disclaims any warranty for Comet. Comet is provided 'As Is' without any express or implied warranty of any kind, including but not limited to any warranties of merchantability, noninfringement, or fitness of a particular purpose.

Comet may not be redistributed without authorization of the author.

Comet uses:

The Lua programming language, (C) 1994–2013 Lua.org, PUC-Rio.

The Lua Just-In-Time Compiler, (C) 2005-2015 Mike Pall.

The Scintilla Component, (C) 1998-2004 Neil Hodgson.

The wxWidgets GUI toolkit, (C) 1992-2013 the wxWidgets team.