LAL++

(Live-a-Log)

A Graphic User Interface builder for C++ programming on X11 systems.

Developed by S. Morel.

1 Introduction (FAQ)

- **What is LAL++ (a bit of history) ?**

  LAL++ is the C++ implementation of LAL. LAL was designed in 1996, originally to make quick tests of image-processing algorithms developed on Apple MacOS (systems 7 to 9). It is a library of functions for creating graphical user interfaces (GUIs). LAL allows to quickly create panels of buttons, text inputs, bar-graphs, plots, and maps. Everything is displayed with colors and 3-D shaded effects (in a "chocolate-bar" style). Compared to the original LAL, LAL++ brought some new features:

  - True-color display. No need to set 256-color mode: Millions-color mode supported.
  - LUT for maps can be either grey-levels or rainbow-like colors.
  - Any type of variable can be displayed as a numerical value, a plot, or a map.
  - Panels of buttons can have any number of buttons for each row.
  - GUI elements can be displayed side-by-side and not only as a stack.

  LAL++ was developed in 2001. In 2018 (17 years after !), it has been ported to the X-WIndow system (X11) to be used on the Linux/Unix environments.

- **What does LAL mean ?**

  It is the acronym of "Live-A-Log", because it was at its time a happy alternative to the standard but grimmy "Die-A-Log" toolbox provided by MacOS. Incidentally, LAL means also "red" in Hindi.

- **What is the license required to use LAL ?**

  LAL++ is freeware and can be used under the general conditions of GNU General Public License. Check out http://www.gnu.org/licenses/gpl.html to know more about these conditions, if you didnt get a license along with your LAL++ distribution.

- **How can I reach the author of LAL++ ?**

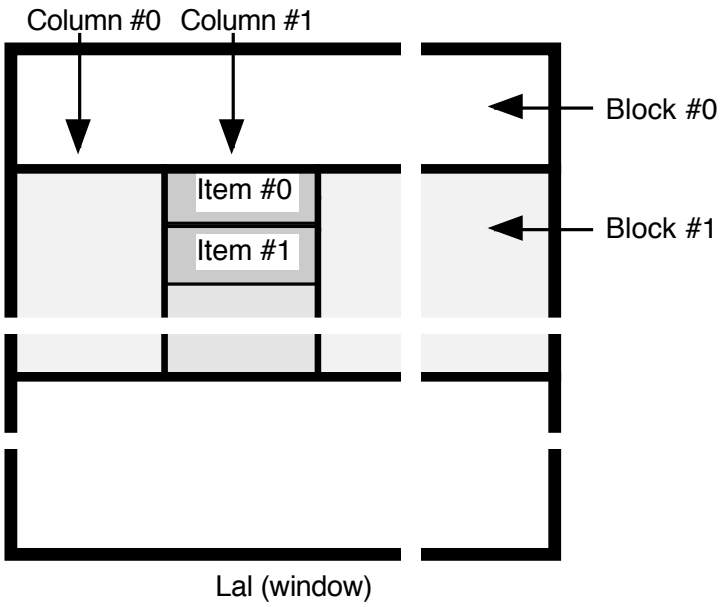
  Simply by e-mail: `sebastien.morel.hugon@gmail.com`

2 Elements

Graphical interfaces made with LAL++ consist in "items". An item may be, for example, a panel of button, a text input, etc... Each item is bound to a variable of the main program (or a global variable) which value is displayed by LAL++ according to its type (integer, string of characters, 1-D array, 2-D array). In some cases, the value of the variable may be changed by action on the interface, therefore by the user. This allows to quickly develop software with graphic user interfaces.

Items are gathered in a window called a "lal" (in lower case). Many lals can be displayed and managed at the same time. The main program uses a "lal environment" global variable which consists in resources used by all the lals.

Compared with the first versions of LAL, LAL++ brings a new way to organise items in the lal. The lal is now divided in horizontal zones called "blocks". Each block is divided in vertical zones called "columns". In each column, items are stacked.



3 Description of the items

3.1 Group of buttons

A "group of buttons" item is a panel of buttons (or widgets) which can be pressed by the user. There are several types of group of buttons (see page x). A long type variable (or argument) reports the current state (which buttons are or have been pressed) of the item.

3.2 Input from keyboard

An "input from keyboard" item allows the user to set a value for a variable from the keyboard. The user must click the button containing the name of the variable, and then enter the value, validated by either the "Return" or the "Enter" key. The argument (the variable) bound to such an item may be `long`, `unsigned long`, or `char *` (character string).

3.3 Display-only

A "display-only" looks like an "input from keyboard", but the label corresponding to variable is not written on a clickable button. Therefore, the user cannot change the displayed value of the variable.

3.4 Plus-minus

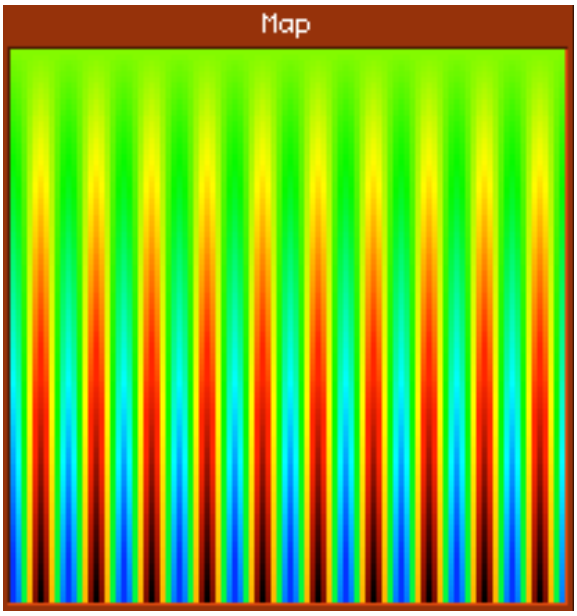
A "plus-minus" item is similar to an "input from keyboard", except that the value of the variable can be easily changed by clicking on two buttons with arrows "Up" and "Down". Two modes are possible. In the first one, clicking the Up arrow adds the defined step to the variable, and clicking the Down arrow subtracts the defined step to the variable. In the second mode, clicking the Up arrow multiplies the variable by the defined step, and clicking the Down arrow divides the variable by the defined step.

3.5 Bar-graph

A "bar-graph" item is a bar with a length proportional to the value of the argument bound to the item. A bar-Graph has two parameters "Min" and "Max". If the argument is equal to Min, the length of the bar will be zero. If the argument is less than Min, a minus sign (meaning "underflow") will be displayed. If the argument is equal to Max, the length of the bar will be maximum. If the argument is greater than Max, a plus sign (meaning "overflow") will be displayed.

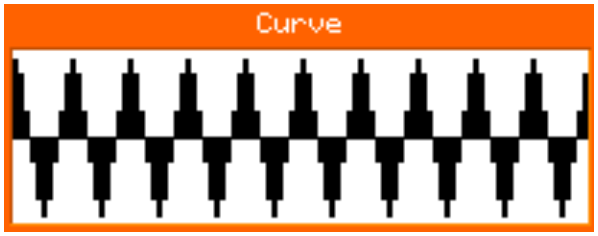
3.6 Map

A "map" item is a graphical representation of a 2-D array. The intensity of each point is proportional to the value of the element of the array represented. A map can be sensitive: clicking a point of the map yields the coordinates of this point in a "point-type" variable.



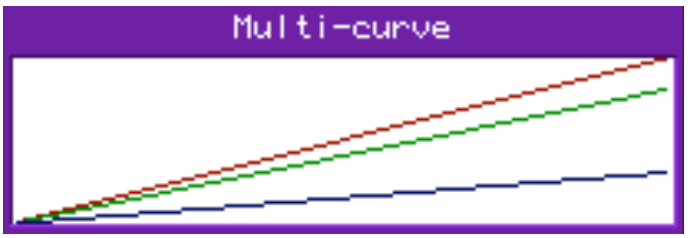
3.7 Graph

A "graph" item is a graphical representation of a 1-D array. The height of each vertical bar of the graph is proportional to the element of the array represented.



3.8 Multi-graph

A "multi-graph" item is a graphical representation of two 1-D arrays. The vertical range is not automatically calculated, as for a curve. One of the curves is dashed for easing identification.



4 Start programming with LAL++

4.1 Installation

LAL++ consists of following files:

- `lal.C` : source file of LAL++. The code inside is independent from the graphic-system of the computer on which LAL is used.
- `lalPort_X11.C` : source file of the "port" (interfacing) between LAL and the X11 graphic-system.
- `lal.h` : header file of LAL++.
- `lalPort_X11.h` : header of the X11 port.
- `lalPort_switch.h` : header to select the port to be used to communicate the graphic-system. By default, the X11 port is selected by this file.

The more convenient way to use LAL++ in C++ programs is to build a static library, by doing the following (from the directory where the .C and .h files have been copied):

```
g++ -lm -lX11 -c lalPort_X11.C
g++ -lm -lX11 -c lal.C
ar -rcs liblalpp.a lalPort_X11.o lal.o
ranlib libllalpp.a
mv liblalpp.a <place_of_user_defined_libraries>
cp lalPort_X11.h <place_of_user_defined_libraries>
cp lal.h <place_of_user_defined_libraries>
cp lal_port_switch.h <place_of_user_defined_libraries>
```

Another way to use LAL++ is to copy the .C and .h files above into the directory of the source code of the program.

4.2 Compilation of programs using LAL++

If the LAL++ library has been created (see previous sub-section), the source file(s) of a program using LAL++ shall contain at the beginning:

```
#include <lal.h>
```

Otherwise, if the source files of LAL++ are directly used in the directory containing the source files of the program under development, these source files shall contain at the beginning:

```
#include "lal.h"
```

To compile a program which uses LAL++, use the following command (if the LAL++ library has been created):

```
g++ -I<place_of_user_defined_libraries> -L<place_of_user_defined_libraries>
    <source file 1> ... <source file n> -o <executable file> -lm lX11 -llalpp
```

Note that the order of the arguments is important for many g++ compilers-linkers.

If no libraries are created (source files of LAL++ copied into the directory of the source files of the program under development), use the following command:

```
g++ -lm lX11 -Im <source file 1> ... <source file n> lalPort_X11.C lal.C
    -o <executable file>
```

Note that we suppose that the X11 and Maths libraries (along with their header files) are already accesible by the system variables which define the default paths to libraries and headers.

4.3 LAL++ environment and objects

In the `main` function of the program, it is first necessary to define a LAL++ "environment" object. This object contains resources that are common to the lals. Hence, the `main` function of the program shall contain the following lines:

```
LalEnv *theLalEnv
...
theLalEnv = new LalEnv(argc,argv)
```

where `argc` and `argv` are the arguments of the `main` function.

A lal is an object of class `Lal`. The syntax of the constructor is:

```
Lal(char *theTitle,long xLal,long yLal)
```

`theTitle` is a character string containing the name of the window to which corresponds the Lal. `xLal` and `yLal` are the coordinates of the left-top corner of the window ((0,0) corresponds to the left-top corner of the screen).

Once a lal has been created, it is necessary to attach it to its environment by using the `Attachlal` method of the `LalEnv` class:

```
LalEnv::AttachLal)(Lal *theLal)
```

5 Creating items

The new lal can now be filled with its items.

5.1 Group of buttons

To create a group of buttons, use the following method of class Lal:

```
long Lal::NewSelector(char *theTitle,long theColor,long theType,char *theNames,
                      long *theArg)
```

`theTitle` is the title of the group of buttons, to be displayed at the top. `theColor` is the color of the group of button (see list of colors on in Sec. 10). `theType` can be one of the following constants:

- `ONE` : Only one button can be pressed. Clicking on a new button releases the previous pressed button. The argument contains the number (starting from 0) of the pressed button.

- `CLICK` : Same as ONE, but the button appears released after a click. The argument contains therefore the number of the last button clicked.
- `MANY` : Several buttons can be pressed at the same time. Clicking on a button toggles its binary state (0=pressed/1=not pressed). The argument contains the value $\sum_{i=0} state(i) \times 2^i$, where $state(i)$ is the state of button number i (i starts from zero).

It is possible to choose the number of buttons for each row in the group. This number must be contained in the 6 lower bits of `theType`. For example, the value of theType for a group of button where only one button can be pressed, and with 4 buttons per row, should be: `ONE | 4`. If no value for the number of buttons per row is specified, the group will have 2 or 3 buttons per row, depending on the total number of buttons. `theNames` is the names of the buttons. They must be separated by the character — (see example in Sec. 13). `theArg` is the argument to which the state of the group of buttons must be reported.

5.2 Input from keyboard

To create an "input from keyboard" item, use the following method of class Lal:

```
long Lal::NewInputKbrd(char *theTitle,long theColor,long theSize,<type> *theArg)
```

`theTitle` contains the description of the variable to be modified by the user. This description will be displayed. `theColor` is the color of the item (see Sec. 10). `theSize` is the maximum number of characters that can be typed by the user. `theArg` is the variable to be modified and can be of any of the types given in Sec. 11 The returned value is either the item-number in the lal (starting from 0), or -1 if an error occurred (e.g., out of memory).

Note: if the type of `theArg` is `char *`, then the argument is treated as a character string which length is `theSize`. What is entered and displayed is what the character string contains, not the numerical value (in decimal) of the character pointed by `theArg`. Nevertheless, if the type of the argument is `unsigned char *`, then the entered/displayed decimal value will be the one of the byte pointed by `theArg`.

5.3 Display-only

To create a "display-only" item, use the following method of class Lal:

```
long Lal::NewDisplayOnly(char *theTitle,long theColor,long theSize,<type> *theArg)
```

The syntax is the one used for creating an "input from keyboard". `theSize` is the maximum number of characters to be displayed. The returned value is either the item- number in the lal (starting from 0), or -1 if an error occurred (e.g., out of memory).

As for "input from keyboard" items, the same remark applies to arguments having a `char *` type.

5.4 Plus-minus

To create a "plus-minus" item, use the following method of class Lal:

```
long Lal::NewPlusMinus(char *theTitle,long theColor, long theSize,double min,
                       double max,double step,long progrs,<type> *theArg)
```

`theTitle` contains the description of the variable to be modified by the user. This description will be displayed. `theColor` is the color of the item (see Sec. 10). `theSize` is the maximum number of characters

that can be typed by the user. min and max are respectively the minimum and the maximum value that can be set by the user, using either the up and down arrows or the keyboard. step is the progression step used when up-and down arrows are clicked on. progrs is the type of progression. It can be:

- ARI : Arithmetical progression (e.g.,1,2,3,4,etc...).
- GEO : Geometrical progression (e.g.,1,2,4,8,etc...).

theArg is the variable to be modified and can be of any of the types given in Sec. 11. The returned value is either the item-number in the lal (starting from 0), or -1 if an error occurred (e.g., out of memory).

5.5 Bar-graph

To create a "bar-graph" item, use the following method of class Lal:

```
long Lal::NewBarGraph(char *theTitle,long theColor,double min,double max,
                      <type> *theArg)
```

theTitle contains the description of the variable which value will be displayed as a bar- graph. This description will be displayed. theColor is the color of the item (see Sec. 10). min and max are the limits defining the range of the bar-graph. An empty bar with a minus sign (meaning underflow) will be displayed if the value of variable is smaller than min, A full bar with a plus sign (meaning overflow) will be displayed if the value of variable is greater than max. theArg is the variable to be modified and can be of any of the types given in Sec. 11. The returned value is either the item-number in the lal (starting from 0), or -1 if an error occurred (e.g., out of memory).

5.6 Map

To create a "map" item, use the following method of class Lal:

```
long Lal::NewMap(char *theTitle,long theColor,long xo,long yo, long xl,long yl,
          long z,<type> **theArg,Point *CursLoc)
```

theTitle contains the description of the 2-D array which will be displayed as a map. theColor is the color of the item (see Sec. 10). xo and yo are the coordinates of the top-left corner of the zone of the map to display, while xl and yl are respectively the width and the height of this zone. z is the zoom factor used for displaying. If z is positive, the zone is displayed with a magnification value equal to z. If z is negative, the zone size is (-xl*z,-yl*z), and is displayed with a reduction factor equal to -z.

theArg is the array to be displayed and can be of any of the types given in Sec. 11. CursLoc is a point-type variable reporting the coordinates of the point of the map that has been clicked on with the mouse. The returned value is either the item-number in the lal (starting from 0), or -1 if an error occurred (e.g., out of memory).

The LUT (look-up table) that is used for the map must have been indicated before the NewMap instruction by using the SetLut instruction (which is not a method):

```
void SetLut(long lut)
```

lut can be either the GREY constant for a grey-level LUT, or COLOR for a "rainbow" LUT, starting from dark red for the lowest value displayed, going through orange, yellow then green tones as the displayed value increases, and ending at bright blue for the highest value displayed

5.7 Graph

To create a graph, use the following method of class Lal:

```
long Lal::NewGraph(char *theTitle,long theColor,long xo, long xl,long yl,long z,
                   <type> *theArg,double *min,double *max)
```

theTitle contains the description of the 1-D array which will be displayed as a graph. theColor is the color of the item (see Sec. 10). xo is the starting point of the zone of the 1-D array to be displayed. xl is the number of displayed points. If z is positive, the zone is displayed with a magnification value equal to z. If z is negative, the zone size is -xl*z, and is displayed with a reduction factor equal to -z. theArg is the array to be displayed and can be of any of the types given in Sec. 11. Minimum and maximum values of the array that are displayed are reported in min and max.

The returned value is either the item-number in the lal (starting from 0), or -1 if an error occurred (e.g., out of memory).

5.8 Multi-graph

To create a "multi-graph" item, use the following method of class Lal:

```
long Lal::NewMultiGraph(char *theTitle,long theColor,long xo, long xl,long yl,
                        long z,long n,<type> **theArg,double *min,double *max)
```

theTitle contains the description of the 1-D array which will be displayed as a graph. theColor is the color of the item (see Sec. 10). xo is the starting point of the zone of the 1- D array to be displayed. xl is the number of displayed points. yl is the number of graphs to be displayed. If z is positive, the zone is displayed with a magnification value equal to z. If z is negative, the zone size is -xl*z, and is displayed with a reduction factor equal to -z. theArg is the array to be displayed and can be of the types given in Sec. 11. The indexing order in the array must be:

```
theArg[{graph-number}][{x}]
```

Minimum and maximum values of the array that are displayed are reported in min and max. The returned value is either the item-number in the lal (starting from 0), or -1 if an error occurred (e.g., out of memory).

5.9 Dealing with complex numbers

Any items (except groups of buttons) can use complex for the type of their argument. In this case, the part of the complex number must be specified as an extra argument, placed right after theArg argument. Type of this argument long, and can be one of the following constants:

Lreal	Real part.
Limaginary	Imaginary part.
Lmodulus	Modulus.
Lsquaremod	Squared modulus.
LphaseDeg	Phase, expressed in degrees.
LphaseRad	Phase expressed in radians.

6 Arranging items in the lal

By default, a created Lal has one block with one column. Items will be displayed in the column from top to bottom in the order they are created. Creating an item implicitly fills a column. To create a new column at the right of the current one has been filled with the desired items, use the method of class Lal:

```
void Lal::NextColumn(void)
```

To create a new block, after the last column of the current block has been filled, and below the current block, use the method of class Lal:

```
void Lal::NextBlock(void)
```

7 Managing lals

Once all the lals have been filled with their items, they must be "computed" by using the following method:

`long Lal::Prepare(void)`

The returned value is 0 if an error occurred (e.g., out of memory). To display a lal, use the following method:

`void Lal::Show(void)`

The lals should be managed in the main loop of the program, by using the instruction:

`void Lal::WatchLal(void)`

This instruction checks the state of the mouse, and accordingly updates the variables bound to group of buttons. Values of variables bound to "input from Keyboard" or "plus-minus" items are also updated by `WatchLal`.
Still in the main loop, if variables (which values are bound to Lal items) have been updated, the display of each Lal must be updated accordingly with the following method of class `Lal`:

`void Lal::Update(void)`

It is possible to change "on fly" some features of items. Thus, an "input from Keyboard" item can be turned into a "Display Only" by using the method of class `Lal`:

`long Lal::LockInput(long n)`

Where `n` is the item-number (returned after its creation) of the lal. The returned value is 1 if the item has been locked, or 0 if an error has occurred (item type mismatch). Reversely, a "display-only" item can be turned into an "input from keyboard" by using the method of class Lal:

`long Lal::UnlockInput(long n)`

Where `n` is the item-number (returned after its creation) of the lal. The returned value is 1 if the item has been locked, or 0 if an error has occurred (item-type mismatch). To change the displayed zone of a "map" item, use the following method of class Lal:

`long Lal::ModifyMap(long n,long xo,long yo,long z)`

Where `n` is the item-number (returned after its creation), `xo` and `yo` are the new coordinates of the displayed zone, and `z` is the new zoom factor. The returned value is 0 if an error has occurred.
To change the displayed zone of a "graph" item, use the following method of class Lal:

`long Lal::ModifyGraph(long n,long xo,long z)`

Where n is the item-number (returned after its creation), xo is the new origin of the displayed zone, and z is the new zoom factor. The returned value is 0 if an error has occurred.
Removing lals To just remove a lal from screen, use the following method of class `Lal`:

`long Lal::Hide(void)`

Use the `Show` method to display it back. To delete a lal with no possibility to display it back, use the destructor of class `Lal`:

`void Lal::~Lal(void)`

8 Colors

The possible values for theColor argument in item creation are the following explicitly- named constants:

- `Lred`
- `Lgreen`
- `Lblue`
- `Lbrown`
- `Lgrey`
- `Lamber`
- `Lpink`
- `Lpurple`

9 Possible types of argument

The following types are supported for the argument of all items, except groups of buttons:

- `long`
- `unsigned long`
- `short`
- `unsigned short`
- `float`
- `double`
- `char`
- `unsigned char`
- `complex`

10 Example

The following listing is a demonstration program of LAL++:

```
#include "lal.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>


int main(int argc, char **argv)
```

```
{
  LalEnv *testLalEnv;
  Lal *testLal;
  Lal *testLal2;
  long x=0;
  long y=0;
  float v=123.45;
  double **a;
  float *b;
  long i,j;
  long running;
  char s[40];
  char dum[80];
  char tvChans[40];
  Point pt;

  pt.h=0;
  pt.v=0;
  strcpy(tvChans,"BBC|CBS|NBC|NHK");

  a=(double **)malloc(100*sizeof(double *));
  b=(float *)malloc(100*sizeof(float));
  for (i=0;i<100;i++)
  {
    b[i]=cos(2*(float)i*PI/10);

    a[i]=(double *)malloc(100*sizeof(double));
    for (j=0;j<100;j++)
    {
      a[i][j]=b[i]*sin(2*(float)j*PI/50);
    }
  }
  strcpy(s,"bubu");

  testLalEnv=new LalEnv(argc,argv);

  testLal=new Lal("Test",50,50);
  testLalEnv->AttachLal(testLal);

  testLal->SetLut(COLOR);
  testLal->NewSelector("TV Channel",Lgreen,CLICK,tvChans,&x);
  testLal->NextColumn();
  testLal->NewInputKbrd("Enter X",Lred,8,s);
  testLal->NextBlock();
  testLal->NewPlusMinus("Threshold",Lblue,8,0,1000,2,ARI,&v);
  testLal->NewBarGraph("Fuel",Lpink,10.,250.,&v);
  testLal->NewSelector("Fruits",Lpurple,MANY,"Apple|Banana|Grape|Orange|Peach|Pear",&y);
  testLal->NewInputKbrd("Fruits",Lpurple,4,&y);
  testLal->NextColumn();
  testLal->NewGraph("Curve",Lamber,0,100,60,2,b,NULL,NULL);
```

```
  testLal->NewMap("Map",Lbrown,0,0,100,100,2,a,&pt);
  testLal->NewMultiGraph("Multi-curve",Lpurple,0,100,50,2,1,3,a,NULL,NULL);

  testLal->Prepare();
  testLal->Show();

  testLal2=new Lal("Test2",300,50);
  testLalEnv->AttachLal(testLal2);

  testLal2->SetLut(COLOR);
  testLal2->NewSelector("TV Channel",Lpink,ONE,tvChans,&x);
  testLal2->NewMap("Map",Lbrown,0,0,100,100,2,a,&pt);
  testLal2->Prepare();
  testLal2->Show();

  running=1;
  while (running)
  {
    testLalEnv->WatchLal();
    printf ("Clicked point on map: x = %d ; y = %d\n",pt.h,pt.v);
    if (x==3)
    {
      running=0;
    }
    testLal2->Update();
  }
  delete testLal;
  delete testLalEnv;
  exit(0);
}
```