AVC, Application View Controller

User Manual

version 0.5.0

Fabrizio Pollastri <pollastri@inrim.it>

AVC outline

Current version is AVC 0.5.0, beta status, released 4-Feb-2008.

Tested on:

Debian GNU/Linux Lenny, FP 4-Feb-2008.

author:

Fabrizio Pollastri, e-mail: pollastri (at) inrim.it.

The AVC web site is hosted at **http://avc.inrim.it**

Logo:

Author note

The author will be happy to ear about any usage of AVC. Please, feel free to send questions, corrections and suggestions to the author. The poor English of this manual requires special indulgence.

Document info

author:	Fabrizio Pollastri
created:	2007-07-10-11:25:23 PM
modified:	2008-02-02-01:22:41
version:	67

Table of Contents

1 Introduction

1.1 What is

AVC, the Application View Controller is a multiplatform, fully automatic, live connection among graphical interface widgets and application variables for the python [1] language.

AVC supports in a uniform way the most popular widget toolkits: GTK [2], Qt3 [3], Qt4 [4], Tk [5], wxWidgets [6].

AVC is a normal python module that can be imported by any python application.

Graphical User Interfaces (GUIs) are the easy way to input data to an application software and to view the data produced by the application. The management of data exchanges between the GUI and the application is a central problem in GUI programming, it absorbs a relevant part of the programming effort. AVC makes the programming of this data exchanges very easy.

AVC is a fully transparent and automatic connection between the values displayed and entered by GUI widgets and the variables of an application using the GUI. The connection is bidirectional. If the application sets a new value into a connected variable, AVC copies the new value into all the widgets connected to the variable. If a new value is entered by a widget, AVC copies the new value into all other widgets connected the variable, into the variable and optionally notifies the change to the application. The connections are autogenerated by looking for matching names between widget names and variable names.

The application is completely unaware of the presence of the connected variables, it reads and writes them as normal variables. Only if the application requires to be immediately notified when a connected variable changes value, a notify handler must be added to the application.

1.2 Features

- Fully transparent widget-variable connections
- Automatic connection by matching widgets and variables names
- No design pattern, no application redesign, no widget toolkit dependent code
- Multiple widget toolkits support: GTK, Qt3, Qt4, Tk, wxWidgets.
- Full compatibility and support for Glade, Qt Designer, Visual Tcl and wxGlade interface design tools.
- Widgets support: button, check button, combo box, entry, label, radio button, slider, spin button, status bar, text view/edit, toggle button.
- Variable types support: boolean, integer, float, string, list, tuple.
- Multiple widgets to one variable connection
- Dual update timing of variable value views: immediate or periodic.
- Testing printout logging activity with selectable verbosity
- Python module written in pure python
- Free software (GNU GPL license version 3 [15])

1.3 Quick start

Essential instructions to get started with AVC. This instructions are for the GTK toolkit, the usage with the other supported toolkits is very similar. The AVC module is supposed already installed. For a simple example, see further along the section “Spinbutton/Spinbox/SpinCtrl Example” of the widget toolkit of interest.

Import the AVC module for GTK.

```
from avc.avcgtk import *
```

Derive the application class from the AVC class. Let suppose that the application class name is "theApp".

```
class theApp(AVC):
```

Design the GUI with Glade [11] or create it statement by statement in the application, naming the widgets with the rule described below.

Define all variables to be connected in the application. Each variable must have a name equal to the matching name of the widgets that are to be connected to the variable. A widget matching name is the widget name itself, if it does not contain a double underscore '__', otherwise is the name part before the double underscore.

In the application, after the creation of the GUI and after the instantiation of all the variables to be connected, call the instance method 'avc_init'. Let suppose that the application instance name is "the_app".

```
the_app.avc_init()
```

All is done for AVC. From this point, AVC takes full control over data exchange between the connected variables and widgets.

1.4 Installation

To run **AVC**, **Python 2.2 or later** must already be installed. The latest release is recommended. Python is available from http://www.python.org/.

The first step is to download the AVC tarball from http://avc.inrim.it/dist/.

Expand the `tar` archive in a temporary directory (**not** directly in Python's `site-packages`). It contains a distutils setup file "setup.py".

Open a shell. Unpack the tarball in a temporary directory (**not** directly in Python's `site-packages`). Commands:

```
tar zxf avc-X.Y.Z.tar.gz
```

X, Y and Z are the major and minor version numbers of the tarball.

Go to the directory created by expanding the tarball:

```
cd avc-X.Y.Z
```

Get root privileges and install the package:

```
su
(enter root password)
python setup.py install
```

If the python executable isn't on your path, you'll have to specify the complete path, such as /usr/local/bin/python.

2 Common reference

This is the part of the user manual common to all supported widget toolkits: GTK, Qt3, Qt4, Tk and wxWidgets.

2.1 Supported widgets

The following table shows the correspondences between the AVC abstract widget types and the names of the real widgets in the supported toolkits.

Table 1: Map of supported widget

AVC abstract widget type	real widgets by supported toolkits				
	GTK	**Qt3**	**Qt4**	**Tk**	**wxWidgets**
Button	Button	QPushButton(1)	QPushButton(1)	Button	Button BitmapButton
Check Button	CheckButton	QCheckBox	QCheckBox	Checkbutton	CheckBox
Combo Box	Combo Box	QComboBox(2)	QComboBox(2)	-	Choice ComboBox
Entry	Entry	QLineEdit	QLineEdit	Entry	TextCtrl
Label	Label	QLabel	QLabel	Label	StaticText
Radio Button	RadioButton	QRadioButton	QRadioButton	Radiobutton	RadioBox
Slider	Hscale VScale	QSlider(3)	QSlider(3)	Scale	Slider
Spin Button	SpinButton	QSpinBox(4)	QSpinBox(4) QdoubleSpinbox	Spinbox	SpinCtrl
Status Bar	StatusBar(5)	-	-	-	StatusBar(5)
Text View	TextView	QTextEdit	QTextEdit	Text	TextCtrl
Toggle Button	ToggleButton	QPushButton(6)	QPushButton(6)	Togglebutton	ToggleButton

Notes
(1) QPushButton with "toggleButton" property set to "False" (the default).
(2) QComboBox with "editable" property set to "False" (the default).
(3) QSlider manages interger values only.
(4) QSpinBox manages interger values only.
(5) StatusBar is used as a simple output label.
(6) QPushButton with "toggleButton" property set to "True". Set it with QPushButton method `setToggleButton(True)`.

2.2 Widgets-variables names matching

AVC connects widgets and variables using a names matching procedure with the following rules.

The matching name for a variable is the variable name itself.

The matching name for a widget is the widget name itself, if the name does not contain a double underscore ('__'), otherwise the matching name is the part of the widget name before

the double underscore. This allow to differentiate widget names for widgets that are to be connected to the same variable.
Each widget having a matching name equal to a variable matching name is connected to that variable.

A widget can be connected to one variable. A variable can be connected to one or more widgets.

widget name	**matching name**
button_ok	button_ok
toggle__button	toggle
check_button_1	check_button_1
radio_button__2	radio_button

Table 2: Examples of matching names

2.3 Application class

The application that uses AVC must be instantiated from an application class that is derived from the `AVC` class. Let suppose that the application class name is "theApp", the application class statement will be

```
class theApp(AVC):
```

The `AVC` class is derived from the builtin `object` class that is the base of all new style classes introduced with python 2.2. So, also the application becomes a new style class.

2.4 AVC initialization

AVC start its job just after it is initialized. AVC initialization can take place in the application after the creation of the GUI and after the instantiation of all variables to be connected. AVC initialization is done by calling the instance method `avc_init`. Let suppose that the application instance name is "the_app", the AVC init statement will be

```
the_app.avc_init()
```

When the value of a connected variable is changed, the values displayed by the widgets connected to it are updated by AVC in one of two allowed modes: immediate or periodic. Mode selection is done at AVC initialization specifying the "view_period" argument. If the argument is omitted, like in `the_app.avc_init()`, it is assigned a default value of 0.1 seconds, selecting a periodic views update with that period. If the argument is assigned a value, like in `the_app.avc_init(view_period=0.2)`, views will be updated every "view_period" seconds. If the argument is assigned to zero or to "None" value, like in `the_app.avc_init(view_period=0)`, views will be updated immediately after each change of the variable value.

2.5 Abstract widget collection

Button

The memoryless press button, its connected variable must be a boolean. In normal state (button not pressed) the variable is "False", in pressed state (mouse pointer over button and mouse button 1 pressed) the variable is "True". Names for button widget in supported toolkits: GTK "Button", Qt3 and Qt4 "QPushButton" with toggle attribute off, Tk “Button”, wxWidgets “Button”.

Check button

The behavior of the check button widget is the same of the toggle button widget. See toggle button. Names for check button widget in supported toolkits: GTK "CheckButton", Qt3 and Qt4 "QCheckBox", Tk “Checkbutton”, wxWidgets “CheckBox”.

Combo box
The combo box, an item selector. The connected variable must be of type integer, its value is the index of the selected item. When no item is selected index is -1. Names for combo box widget in supported toolkits: GTK “ComboBox”, Qt3 and Qt4 "QComboBox", not available in Tk, wxWidgets “Choice” “ComboBox”.

Entry
The text entry, its connected variable can be integer, float or string. Text input must conform to the type of the connected variable. If the connected variable is of type string, its value is copied to the entry widget "as is", if type is integer or float, the value is converted to string before copy. Names for text entry widget in supported toolkits: GTK "Entry", Qt3 and Qt4 "QLineEdit", Tk “Entry”, wxWidgets “TextCtrl”.

Label
The text label, its connected variable can be boolean, integer, float, string, list, tuple or object. If the label is created with a default text, AVC tests it against the connected variable to be a valid python formatting string. If the test is successful, the default text is saved by AVC and used to format the label text updates when the connected variable value changes. If the connected variable is a generic python object, the formatting string is applied to the dictionary of the object. If the test is not successful, the label text updates are rendered by the standard python string representation applying the str function to the connected variable. For further details, see the “label example”. Names for text entry widget in supported toolkits: GTK “Label”, Qt3 and Qt4 “QLabel”, Tk “Label”, wxWidgets “StaticText”.

Radio button
The radio buttons come always in groups of two or more radio buttons. Each radio button behaves like a check button, but only one radio button at a time can be checked in each group. A variable of type integer can be connected to each group of radio buttons, its value is the index of the checked radio button in the group. Names for text entry widget in supported toolkits: GTK “RadioButton”, Qt3 and Qt4 “QRadioButton”, Tk “Radiobutton”, wxWidgets “RadioBox”.

Slider
The slider, its connected variable can be integer or float. The GTK "HScale" and "VScale" support both types. On the contrary, Qt3 and Qt4 support only integer with "QSlider" widget. Remember that in python floats are always doubles. Names for text entry widget in supported toolkits: GTK “Hscale” and “Vscale”, Qt3 and Qt4 “QSlider”, Tk “Slider”, wxWidgets “Slider”.

Spin button
The spin button, its connected variable can be integer or float. The GTK "SpinButton" support both types. On the contrary, Qt3 and Qt4 differentiate integer or float support with two widgets: "SpinBox" and "DoubleSpinBox". Remember that in python floats are always doubles. Names for spin button widget in supported toolkits: GTK "SpinButton", Qt3 and Qt4 "QSpinBox" for integer and "QDoubleSpinBox" for float, Tk “Spinbox”, wxWidgets “SpinCtrl”.

Status bar
The status bar, its connected variable is a string. Names for text view/edit widget in supported toolkits: GTK "StatusBar", Qt3, Qt4 and Tk not supported, wxWidgets “StatusBar”.

Text view/edit
The text view/edit, its connected variable is a string. Names for text view/edit widget in supported toolkits: GTK "TextView", Qt3 and Qt4 "QtextEdit", Tk “Text”, wxWidgets “TextCtrl”.

Toggle button
The toggle button, a button with memory, its connected variable must be a boolean. Each time the button is pressed, it changes its state: from on to off or viceversa. In off state the variable

is "False", in on state the variable is "True". Names for toggle button widget in supported toolkits: GTK "ToggleButton", Qt3 and Qt4 "PushButton" with toggle attribute on, Tk “Togglebutton”, wxWidgets “ToggleButton”.

2.6 Testing and debugging

AVC can produce a printout of its activity that can be useful for testing and debug purposes. The verbosity level of the printout can be selected from 0 (minimum) to 4 (maximum). Let suppose that the program to test is “myprogram.py”, then to produce the printout with the maximum verbosity the following command is required.

```
myprogram.py --avc-verbosity 4
```

The content of the each verbosity level is the following.

- **level 0**: nothing printed, the default.
- **level 1**: header with AVC version, widget toolkit type, program name, verbosity level, connection update mode; connection list with name, variable type, initial value.
- **level 2**: as level 1 plus the widgets and the change handlers list of each connection.
- **level 3**: as level 2 plus the details of widgets in connections lists.
- **level 4**: as level 3 plus full widget tree scansion at init time.

2.6.1 Testing printout for example gtk_counter.py

The following example shows the output produced by running the example “gtk_counter.py” (see “GTK examples”) with maximum verbosity.

```
./gtk_counter.py --avc-verbosity 4
++++
AVC 0.5.0 - Activity Report
widget toolkit binding: GTK
program: ./gtk_counter.py
verbosity: 4
connection update mode: periodic, period=0.1 sec
widget tree scansion at init ...
  skip unsupported widget Window,"GtkWindow"
  skip unsupported widget Window,"counter"
  skip unmatched widget Label,"GtkLabel"
  skip unsupported widget HBox,"hbox1"
  add widget Label,"counter" to connection "counter"
  add widget CheckButton,"high_speed" to connection "high_speed"
  skip unmatched widget Label,"GtkLabel"
creating connection "counter" ...
  type: <type 'int'>
  initial value: 0
  widget: <gtk.Label object at 0xb6f0dbe4 (GtkLabel at 0x8295098)>,"counter"
    valid format string: "<b>%d</b>"
creating connection "high_speed" ...
  type: <type 'bool'>
  initial value: False
  widget: <gtk.CheckButton object at 0xb6f2c9dc (GtkCheckButton at
0x829b000)>,"high_speed"
```

```
  connected handler "high_speed_changed "
----
```

In the “widget tree scansion at init” all the widgets of the GUI are analyzed. Each widget can be skipped (ignored) or added to a connection. A widget is skipped because it is of type not supported AVC or it has a name not matching any variable of the application. A widget is added to a connection because it name matches some application variable. For each widget, its class type and its name are printed.

Things to be noticed. The connection “counter” has a label widget that was preloaded with a valid formatting string ("%d"). The connection “high_speed” has check button widget and it has the change handler “high_speed_changed”

3 GTK Reference

This is the part of the user manual specific to the GTK widgets toolkit.

3.1 Module dependencies

AVC GTK depends on PyGTK [7] the python wrapper for GTK libraries. AVC GTK imports the following modules from PyGTK.

```
import gtk
import gobject
```

3.2 Widget naming

Both Glade, the interface designer, and GTK allow duplicated naming of widgets.

3.3 Status bar widget

AVC uses the GTK status bar widget as a simple output label. Only context #1 with one or none message on status bar stack is used.

3.4 Interface designer

AVC is fully compatible with Glade, the design tool for GTK. Glade produces an interface description that is saved as a specific xml format (.glade).

4 Qt3 Reference

This is the part of the user manual specific to Qt3 [3] widgets toolkit.

4.1 Module dependencies

AVC Qt3 depends on PyQt v3 [8] the python bindings for Qt v3 application framework. AVC Qt3 imports the following modules from PyQt.

```
import qt
```

4.2 Widget naming

Qt3 Designer and Qt3 **do not** allow duplicated naming of widgets. So use the 'double underscore' mechanism to differentiate widgets names.

4.3 Application class

The application that uses AVC must be instantiated from an application class that is derived from the `QApplication` class and from the `AVC` class. Let suppose that the application class name is "theApp", the application class statement will be

```
class theApp(QApplication,AVC):
```

4.4 Interface designer

AVC is fully compatible with Qt3 Designer, the design tool for Qt3. Qt3 Designer produces an interface description that is saved as a specific xml format (.ui).

5 Qt4 reference

This is the part of the user manual specific to Qt4 [4] widgets toolkit.

5.1 Module dependencies

AVC Qt4 depends on PyQt v4 [8] the python bindings for Qt v4 application framework. AVC Qt4 imports the following modules from PyQt.

```
import PyQt4.Qt as qt
```

5.2 Widget naming

Qt4 Designer and Qt4 **do not** allow duplicated naming of widgets. So use the 'double underscore' mechanism to differentiate widgets names.

5.3 Application class

The application that uses AVC must be instantiated from an application class that is derived from the `QApplication` class and from the `AVC` class. Let suppose that the application class name is "theApp", the application class statement will be

```
class theApp(QApplication,AVC):
```

5.4 Interface designer

AVC is fully compatible with Qt4 Designer, the design tool for Qt4. Qt4 Designer produces an interface description that is saved as a specific xml format (.ui).

6 Tk reference

This is the part of the user manual specific to Tk [5] widgets toolkit.

6.1 Module dependencies

AVC Tk depends on Tkinter [9] the python bindings for Tk application framework. Tkinter is part of the standard python library. AVC Tk imports the following module from python standard library.

```
import Tkinter
```

6.2 Widget naming

The Tk toolkit has a specific naming scheme for its widgets. Widget name is generally the concatenation of its parent's name followed by a period (unless the parent is the root window .) and a string containing no periods, e. g. ".baseframe.button1". For this reason, the complete name of each widget is unique. AVC takes as widget name not the complete Tk name but only the part after the rightmost dot. For example a widget with the complete Tk name ".baseframe.button1" has the AVC name "button1".

6.3 Interface designer

AVC supports the 'Visual Tcl' interface design tool for Tk. Visual Tcl produces an interface description that is saved as tcl script.

7 wxWidgets reference

This is the part of the user manual specific to wxWidgets [6] widgets toolkit.

7.1 Module dependencies

AVC wxWidgets depends on wxPython [10] the python bindings for wxWidgets application framework. AVC wxWidgets imports the following module from python standard library.

```
import wx
```

7.2 Widget naming

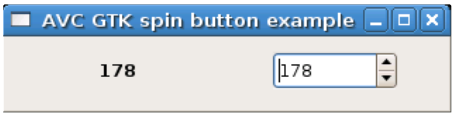
Both wxGlade, the interface designer, and wxWidgets allow duplicated naming of widgets.

7.3 Interface designer

AVC supports the 'wxGlade' interface design tool for wxWidgets and all other design tools producing an interface description that is saved in the native xml format ('xrc') of wxWidgets.

8 GTK examples

8.1 Spin button example



This simple example shows how **AVC** can manage data exchange from widget to widget without any specific code in the application. The program creates a window with two widgets: a spin button and a label. When the value in the spin button is changed by clicking on up or down arrows or by entering it with the keyboard, the new value is displayed into the label.

8.1.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2006 Fabrizio Pollastri
# .license    : GNU General Public License v3

import gtk                              # gimp tool kit bindings
import gtk.glade                        # glade bindings

from avc.avcgtk import *                # AVC for GTK


GLADE_XML = 'gtk_spinbutton.glade'      # GUI glade descriptor


class Example(AVC):
  """
  A spin button whose value is replicated into a label
  """

  def __init__(self):

    # create GUI
    self.glade = gtk.glade.XML(GLADE_XML)

    # autoconnect GUI signal handlers
    self.glade.signal_autoconnect(self)

    # the variable holding the spin button value
    self.spin_value = 0


  def on_destroy(self,window):
    "Terminate program at window destroy"
    gtk.main_quit()


#### MAIN

example = Example()                     # instantiate the application
example.avc_init()                      # connect widgets with variables
gtk.main()                              # run GTK event loop until quit
```

```
#### END
```

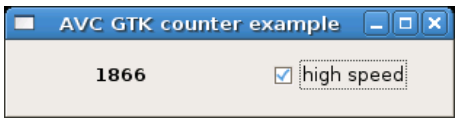
The GUI layout was previously edited with Glade and saved to the file ‘gtk_spinbutton.glade’.

The key points of the example regarding **AVC** are the following.

- During Glade editing, the same name '**spin_value**' was given to the spin button and to the label.
- The specific **AVC** module for GTK is imported at program begin (`from avc.avcgtk import *`).
- The application class is derived from the **AVC** class (`class Example(AVC):`).
- A integer variable with an initial value of 0 and name '**spin_value**' is declared in the application (`self.spin_value = 0`).
- The avc_init method is called after the instantation of the application class, to realize the connections of the two widgets through the '**spin_value**' variable and to initialize the widgets values with the initial value of the variable (`example.avc_init()`).

Example files in directory 'examples' of distribution: program 'gtk_spinbutton.py' , Glade descriptor 'gtk_spinbutton.glade'.

8.2 Counter example



This example shows how **AVC** can manage data input from a check button widget to the application and from the application to a label widget without any specific code in the application. The program creates a window with two widgets: a check button and a label. The label displays the value of an integer counter. The check button controls the increment speed of the counter. Initially, it is unchecked meaning that the increment speed of the counter is 2 units per second. When the user checks the check button the increment speed grows to 10 units per seconds and returns to the initial value (2) when the check button is unchecked again.

8.2.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2006 Fabrizio Pollastri
# .license    : GNU General Public License v3

import gobject                          #--
import gtk                              #- gimp tool kit bindings
import gtk.glade                        # glade bindings

from avc.avcgtk import *                # AVC for GTK


GLADE_XML = 'gtk_counter.glade'         # GUI glade descriptor
LOW_SPEED = 500                         #--
HIGH_SPEED = 100                        #- low and high speed period (ms)


class Example(AVC):
  """
  A counter displayed in a Label widget whose count speed can be
  accelerated by checking a check button.
```

```
  """

  def __init__(self):

    # create GUI
    self.glade = gtk.glade.XML(GLADE_XML)

    # autoconnect GUI signal handlers
    self.glade.signal_autoconnect(self)

    # the counter variable and its speed status
    self.counter = 0
    self.high_speed = False

    # start counter incrementer at low speed
    gobject.timeout_add(LOW_SPEED,self.incrementer)


  def incrementer(self):
    """
    Counter incrementer: increment period = LOW_SPEED, if high speed is False,
    increment period = HIGH_SPEED otherwise. Return False to destroy previous
    timer.
    """
    self.counter += 1
    if self.high_speed:
      period = HIGH_SPEED
    else:
      period = LOW_SPEED
    gobject.timeout_add(period,self.incrementer)
    return False


  def on_destroy(self,window):
    "Terminate program at window destroy"
    gtk.main_quit()


#### MAIN

example = Example()                 # instantiate the application
example.avc_init()                  # connect widgets with variables
gtk.main()                          # run GTK event loop until quit

#### END
```

The GUI layout was previously edited with Glade and saved to the file ‘gtk_counter.glade’.

The key points of the example regarding **AVC** are the following.

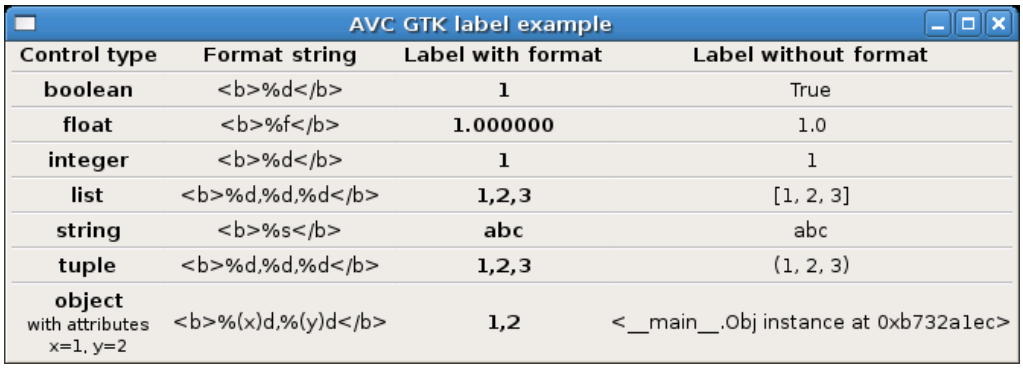
- During Glade editing, the name '**counter**' was given to the label and the name '**high_speed**' was given to the check button.
- The specific **AVC** module for GTK is imported at program begin (`from avc.avcgtk import *`).
- The application class is derived from the **AVC** class (`class Example(AVC):`).
- A integer variable with an initial value of 0 and name '**counter**' is declared in the application to hold the counter value (`self.counter = 0`).
- A boolean variable with an initial value of False and name '**high_speed**' is declared in the application to hold the speed status of the counter increment speed (`self.high_speed = False`).

- The avc_init method is called after the instantation of the application class (`example.avc_init()`) to realize the connections between the '**counter**' variable and the label widget and between the the '**high_speed**' variable and the check button, the label widget is initialized with the initial value of the '**counter**' variable.

Example files in directory 'examples' of distribution: program 'gtk_counter.py' , Glade descriptor 'gtk_counter.glade'.

8.3 Label example

This example shows the formatting capabilities of the label widget. For each supported type of the connected variable, a formatting string is defined and a sample value of the connected variable is displayed into two label widgets: one with formatting and the other with the standard python string representation.



8.3.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2008 Fabrizio Pollastri
# .license    : GNU General Public License v3


import gtk                      # gimp tool kit bindings
import gtk.glade                # glade bindings

from avc.avcgtk import *        # AVC for GTK

GLADE_XML = 'gtk_label.glade'   # GUI glade descriptor


class Example(AVC):
  """
  Showcase of formatting capabilities for the label widget
  """

  def __init__(self):

    # create GUI
    self.glade = gtk.glade.XML(GLADE_XML)
```

```
    # autoconnect GUI signal handlers
    self.glade.signal_autoconnect(self)

    # all types of connected variables
    self.bool_value = True
    self.float_value = 1.0
    self.int_value = 1
    self.list_value = [1,2,3]
    self.str_value = 'abc'
    self.tuple_value = (1,2,3)
    class Obj:
      "A generic object with 2 attributes x,y"
      def __init__(self):
        self.x = 1
        self.y = 2
    self.obj_value = Obj()


  def on_destroy(self,window):
    "Terminate program at window destroy"
    gtk.main_quit()


#### MAIN

example = Example()             # instantiate the application
example.avc_init()              # connect widgets with variables
gtk.main()                      # run GTK event loop until quit

#### END
```

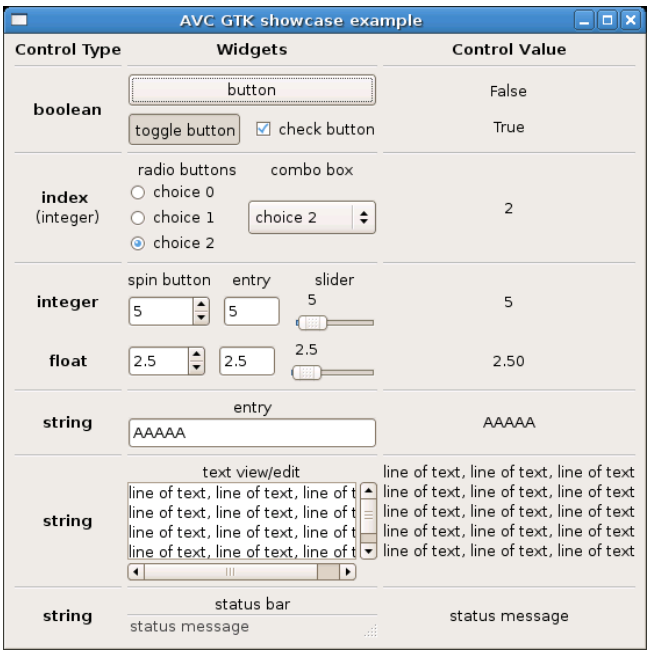
The GUI layout was previously edited with Glade and saved to the file ‘gtk_label.glade’.

Apart the general requirements of AVC, already pointed out in the other examples, the relevant points of the label example about AVC are the following.

- For each control type (for each row) the two label widgets, one in the column “Label with format” and one in the column “Label without format”, are connected to the variable of the corresponding type. For example, in row “boolean”, both label widgets are called “bool_value”, so they connect to the variable `self.bool_value` .
- When the GTK event loop is entered both columns are set to display the initial values of the connected variables. For example, in row “integer”, both labels are set to display the integer value 1.
- The differences of representation between the column “Label with format” and the column “Label without format” reflect the different printout results coming from the formatting capabilities of the label widget and from `str`, the generic textual rendering function of python.

Example files in directory 'examples' of distribution: program 'gtk_label.py' , Glade descriptor 'gtk_label.glade'.

8.4 Showcase example



This example shows a table of all widget/variable type combinations supported by **AVC**. The program creates a window with three columns: the first shows the type of the connected variable, the second shows all the widgets that can be connected to that type of variable, the third shows the current value of each variable. Each row of the window represent a widgets/variable combination as follows.

- Row 1: memoryless button with boolean variable, pressed = True, unpressed = False.
- Row 2: buttons with memory, toggle and check buttons, pressed = True, unpressed = False.
- Row 3: mutually exclusive choices widgets, radio buttons numbered from 0 to 2 and a combo box with 3 items, index variable = number of checked radio button and selected item of combo box.
- Row 4: integer input/output widgets, spin button, entry and slider.
- Row 5: float input/output widgets, spin button, entry and slider.
- Row 6: string input/output widget, entry.
- Row 7: string input/output widget, text view/edit.
- Row 8: status messages, status bar.

The text label widget is used in all output modes for the column of the connected variable values. The program increment the value of each connected variable looping top-bottom at three rows per seconds. The user can also change the values in the connected variables interacting with the widgets.

8.4.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2006 Fabrizio Pollastri
# .license    : GNU General Public License v3
```

```
import gobject                          #--
import gtk                              #- gimp tool kit bindings
import gtk.glade                        # glade bindings

from avc.avcgtk import *                # AVC for GTK


GLADE_XML = 'gtk_showcase.glade'        # GUI glade descriptor
INCREMENTER_PERIOD = 333                # ms


class Example(AVC):
  "A table of all supported widget/control type combinations"

  def __init__(self):

    # create GUI
    self.glade = gtk.glade.XML(GLADE_XML)

    # autoconnect GUI signal handlers
    self.glade.signal_autoconnect(self)

    # the control variables
    self.boolean1 = False
    self.boolean2 = False
    self.radio = 0
    self.integer = 0
    self.float = 0.0
    self.string = ''
    self.textview = ''
    self.status = ''

    # start variables incrementer
    increment = self.incrementer()
    gobject.timeout_add(INCREMENTER_PERIOD,increment.next)


  def incrementer(self):
    """
    Booleans are toggled, radio button index is rotated from first to last,
    integer is incremented by 1, float by 0.5, string is appended a char
    until maxlen when string is cleared, text view/edit is appended a line
    of text until maxlen when it is cleared. Status bar message is toggled.
    Return True to keep timer alive.
    """
    while True:

      self.boolean1 = not self.boolean1
      yield True

      self.boolean2 = not self.boolean2
      yield True

      if self.radio >= 2:
        self.radio = 0
      else:
        self.radio += 1
      yield True

      self.integer += 1
      yield True
```

```
      self.float += 0.5
      yield True

      if len(self.string) >= 10:
        self.string = ''
      else:
        self.string += 'A'
      yield True

      if len(self.textview) >= 200:
        self.textview = ''
      else:
        self.textview += 'line of text, line of text, line of text\n'
      yield True

      if not self.status:
        self.status = 'status message'
      else:
        self.status = ''
      yield True


  def on_destroy(self,window):
    "Terminate program at window destroy"
    gtk.main_quit()


#### MAIN

example = Example()                    # instantiate the application
example.avc_init()                     # connect widgets with variables
gtk.main()                             # run GTK event loop until quit

#### END
```

The GUI layout was previously edited with Glade and saved to the file ‘gtk_showcase.glade’.

The key points of the example regarding **AVC** are the following.

- During Glade editing, the following names were given to the widgets.

Row	widget	name
1	button	**boolean1__button**
	output value label	**boolean1__var**
2	togglebutton	**boolean2__togglebutton**
	checkbutton	**boolean2__checkbutton**
	output value label	**boolean2__var**
3	radiobutton0	**radio__radiobutton0**
	radiobutton1	**radio__radiobutton1**
	radiobutton2	**radio__radiobutton2**
	combobox	**radio__combobox**
	output value label	**radio__var**
4	spinbutton	**integer__spinbutton**
	entry	**integer__entry**
	slider	**integer__slider**
	output value label	**integer__var**
5	spinbutton	**float__spinbutton**
	entry	**float__entry**
	slider	**float__slider**
	output value label	**float__var**

6	entry	**string__entry**
	output value label	**string__var**
7	textview	**textview__textview**
	output value label	**textview__var**
8	statusbar	**status__statusbar**
	output value label	**status__var**

- The specific **AVC** module for GTK is imported at program begin (`from avc.avcgtk import *`).
- The application class is derived from the **AVC** class (`class Example(AVC):`).
- The following variables are declared in the application.

```
self.boolean1 = False
self.boolean2 = False
self.radio = 0
self.integer = 0
self.float = 0.0
self.string = ''
self.textview = ''
self.status = ''
```

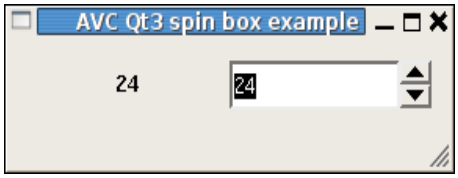
- The avc_init method is called after the instantation of the application class (`example.avc_init()`) to realize the connections of all widegts/variable combinations and to initialize the widgets values with the initial value of the connected variable .

Example files in directory 'examples' of distribution: program 'gtk_showcase.py' , Glade descriptor 'gtk_showcase.glade'.

9 Qt3 examples

9.1 Spin box example

For a functional description of the graphic interface see the GTK “Spin button example“ at page 17.



9.1.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2006 Fabrizio Pollastri
# .license    : GNU General Public License v3

from qt import *                        # Qt interface
from qtui import *                      # ui files realizer
import sys                              # system support

from avc.avcqt3 import *                # AVC for Qt3


UI_FILE = 'qt3_spinbox.ui'


class Example(QApplication,AVC):
  "A spin box whose value is replicated into a text label"

  def __init__(self):

    # create GUI
    QApplication.__init__(self,sys.argv)
    self.root = QWidgetFactory.create(UI_FILE)
    self.setMainWidget(self.root)
    self.root.show()

    # the variable holding the spinbox value
    self.spin_value = 0


#### MAIN

example = Example()                     # instantiate the application
example.avc_init()                      # connect widgets with variables
example.exec_loop()                     # run Qt event loop until quit

#### END
```

The GUI layout was previously edited with Qt3 Designer and saved to the file ‘qt3_spinbox.ui’.

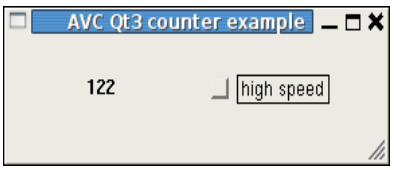
The key points of the example regarding **AVC** are the following.

- During Qt3 Designer editing, the name '**spin_value__spinbox**' was given to the spin box and the name '**spin_value__label**' was given to the label.
- The specific **AVC** module for Qt3 is imported at program begin (`from avc.avcqt3 import *`).
- The application class is derived from the **QApplication** class of Qt3 and from the **AVC** class of AVC (`class Example(QApplication,AVC):`).
- A integer variable with an initial value of 0 and name '**spin_value**' is declared in the application (`self.spin_value = 0`).
- The avc_init method is called after the instantation of the application class (`example.avc_init()`) to realize the connections of the two widgets through the '**spin_value**' variable and to initialize the widgets values with the initial value of the variable .

Example files in directory 'examples' of distribution: program 'qt3_spinbox.py', UI descriptor 'qt3_spinbox.ui'.

9.2 Counter example

For a functional description of the graphical interface see the GTK “Counter example”at page 18.



9.2.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2006 Fabrizio Pollastri
# .license    : GNU General Public License v3


from qt import *                        # Qt interface
from qtui import *                      # ui files realizer
import sys                              # system support

from avc.avcqt3 import *                # AVC for Qt3

UI_FILE = 'qt3_counter.ui'              # qt ui descriptor
LOW_SPEED = 500                         #--
HIGH_SPEED = 100                        #- low and high speed period (ms)


class Example(QApplication,AVC):
  """
  A counter displayed in a Label widget whose count speed can be
  accelerated by checking a check box.
  """

  def __init__(self):

    # create GUI
    QApplication.__init__(self,sys.argv)
```

```
    self.root = QWidgetFactory.create(UI_FILE)
    self.setMainWidget(self.root)
    self.root.show()

    # the counter variable and its speed status
    self.counter = 0
    self.high_speed = False

    # start counter incrementer at low speed
    self.timer = qt.QTimer(self)
    self.connect(self.timer,qt.SIGNAL("timeout()"),self.incrementer)
    self.timer.start(LOW_SPEED)


  def incrementer(self):
    """
    Counter incrementer: increment period = LOW_SPEED, if high speed
    is False, increment period = HIGH_SPEED otherwise.
    """
    self.counter += 1
    if self.high_speed:
      period = HIGH_SPEED
    else:
      period = LOW_SPEED
    self.timer.stop()
    self.timer.start(period)


#### MAIN

example = Example()                 # instantiate the application
example.avc_init()                  # connect widgets with variables
example.exec_loop()                 # run Qt event loop until quit

#### END
```

The GUI layout was previously edited with Qt3 Designer and saved to the file ‘qt3_counter.ui’.

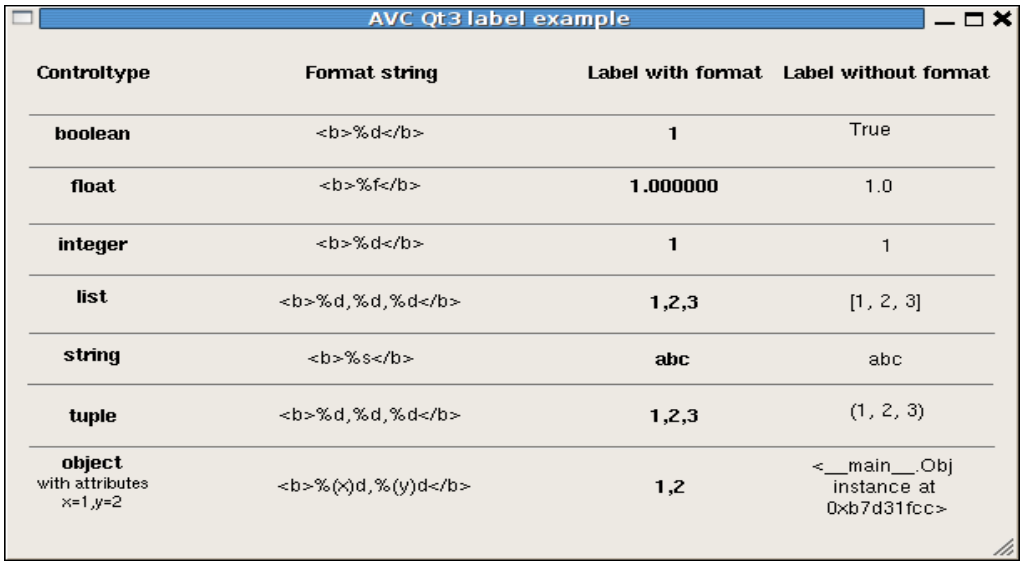
The key points of the example regarding **AVC** are the following.

- During Glade editing, the name '**counter**' was given to the label and the name '**high_speed**' was given to the check button.
- The specific **AVC** module for Qt3 is imported at program begin (`from avc.avcqt3 import *`).
- The application class is derived from the **QApplication** class of Qt3 and from the **AVC** class of AVC (`class Example(QApplication,AVC):`).
- A integer variable with an initial value of 0 and name '**counter**' is declared in the application to hold the counter value (`self.counter = 0`). A boolean variable with an initial value of False and name '**high_speed**' is declared in the application to hold the speed status of the counter increment (`self.high_speed = False`).
- The avc_init method is called after the instantation of the application class (`example.avc_init()`) to realize the connections between the '**counter**' variable and the label widget and between the the '**high_speed**' variable and the check button, the label widget is initialized with the initial value of the '**counter**' variable .

Example files in directory 'examples' of distribution: program 'qt3_counter.py', UI descriptor 'qt3_counter.ui'.

9.3 Label example

This example shows the formatting capabilities of the label widget. For each supported type of the connected variable, a formatting string is defined and a sample value of the connected variable is displayed into two label widgets: one with formatting and the other with the standard python string representation.



9.3.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2008 Fabrizio Pollastri
# .license    : GNU General Public License v3


from qt import *                # Qt interface
from qtui import *              # ui files realizer
import sys                      # system support

from avc.avcqt3 import *        # AVC for Qt3

UI_FILE = 'qt3_label.ui'        # qt ui descriptor


class Example(QApplication,AVC):
  """
  Showcase of formatting capabilities for the label widget
  """

  def __init__(self):

    # create GUI
    QApplication.__init__(self,sys.argv)
    self.root = QWidgetFactory.create(UI_FILE)
    self.setMainWidget(self.root)
    self.root.show()

    # all types of connected variables
    self.bool_value = True
```

```
    self.float_value = 1.0
    self.int_value = 1
    self.list_value = [1,2,3]
    self.str_value = 'abc'
    self.tuple_value = (1,2,3)
    class Obj:
      "A generic object with 2 attributes x,y"
      def __init__(self):
        self.x = 1
        self.y = 2
    self.obj_value = Obj()


#### MAIN

example = Example()                   # instantiate the application
example.avc_init()                    # connect widgets with variables
example.exec_loop()                   # run Qt event loop until quit

#### END
```

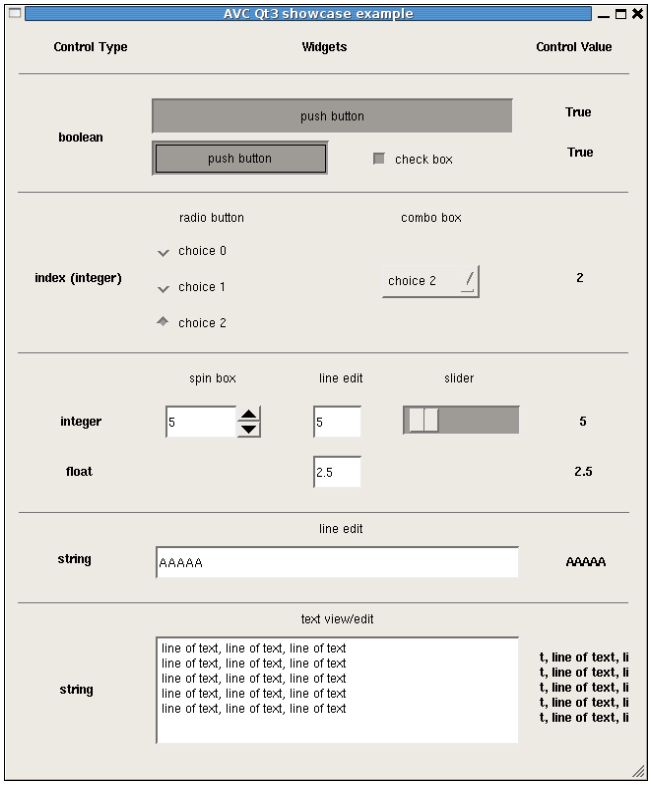
The GUI layout was previously edited with Qt3 Designer and saved to the file ‘qt3_label.ui’.

Apart the general requirements of AVC, already pointed out in the other examples, the relevant points of the label example about AVC are the following.

- For each control type (for each row) the two label widgets, one in the column “Label with format” and one in the column “Label without format”, are connected to the variable of the corresponding type. For example, in row “boolean”, both label widgets are called “bool_value”, so they connect to the variable `self.bool_value` .
- When the Qt3 event loop is entered both columns are set to display the initial values of the connected variables. For example, in row “integer”, both labels are set to display the integer value 1.
- The differences of representation between the column “Label with format” and the column “Label without format” reflect the different printout results coming from the formatting capabilities of the label widget and from `str`, the generic textual rendering function of python.

Example files in directory 'examples' of distribution: program 'qt3_label.py' , UI descriptor 'qt3_label.ui'.

9.4 Showcase example



This example shows a table of all widget/variable type combinations supported by **AVC**. The program creates a window with three columns: the first shows the type of the connected variable, the second shows all the widgets that can be connected to that type of variable, the third shows the current value of each variable. Each row of the window represent a widgets/variable combination.

- Row 1: memoryless button with boolean variable, pressed = True, unpressed = False.
- Row 2: buttons with memory, toggle and check buttons, pressed = True, unpressed = False.
- Row 3: mutually exclusive choices widgets, radiobuttons numbered from 0 to 2 and a combo box with 3 items, index variable = number of checked radiobutton and selected item of combo box.
- Row 4: integer input/output widgets, spin button, entry and slider.
- Row 5: float input/output widget, entry.
- Row 6: string input/output widget, entry.
- Row 7: string input/output widget, text view/edit.

The text label widget is used in all output modes for the column of the connected variable values. The program increment the value of each connected variable looping top-bottom at three rows per seconds. The user can also change the values of the connected variables interacting with the widgets.

9.4.1 Python source

```
#!/usr/bin/python
```

```
# .copyright  : (c) 2006 Fabrizio Pollastri
# .license    : GNU General Public License v3


from qt import *               # Qt interface
from qtui import *             # ui files realizer
import sys                     # system support

from avc.avcqt3 import *       # AVC for Qt3

UI_FILE = 'qt3_showcase.ui'    # qt ui descriptor
INCREMENTER_PERIOD = 333       # ms


class Example(QApplication,AVC):
  "A table of all supported widget/control type combinations"

  def __init__(self):

    # create GUI
    QApplication.__init__(self,sys.argv)
    self.root = QWidgetFactory.create(UI_FILE)
    self.setMainWidget(self.root)
    self.root.show()

    # the control variables
    self.boolean1 = False
    self.boolean2 = False
    self.radio = 0
    self.integer = 0
    self.float = 0.0
    self.string = ''
    self.textview = ''

    # start variables incrementer
    self.increment = self.incrementer()
    self.timer = qt.QTimer(self)
    self.connect(self.timer,qt.SIGNAL("timeout()"),self.timer_function)
    self.timer.start(INCREMENTER_PERIOD)


  def timer_function(self):
    self.increment.next()


  def incrementer(self):
    """
    Booleans are toggled, radio button index is rotated from first to last,
    integer is incremented by 1, float by 0.5, string is appended a char
    until maxlen when string is cleared, text view/edit is appended a line
    of text until maxlen when it is cleared.
    Return True to keep timer alive.
    """
    while True:

      self.boolean1 = not self.boolean1
      yield True

      self.boolean2 = not self.boolean2
      yield True

      if self.radio == 2:
```

```
        self.radio = 0
      else:
        self.radio += 1
      yield True

      self.integer += 1
      yield True

      self.float += 0.5
      yield True

      if len(self.string) >= 10:
        self.string = 'A'
      else:
        self.string += 'A'
      yield True

      if len(self.textview) >= 200:
        self.textview = ''
      else:
        self.textview += 'line of text, line of text, line of text\n'
      yield True


#### MAIN

example = Example()                   # instantiate the application
example.avc_init()                    # connect widgets with variables
example.exec_loop()                   # run Qt event loop until quit

#### END
```

The GUI layout was previously edited with Qt3 Designer and saved to the file ‘qt3_showcase.ui’.

The key points of the example regarding **AVC** are the following.

- During Glade editing, the following names were given to the widgets.

widget	name
Row 1:	
button	**boolean1__button**
output value label	**boolean1__var**
Row 2:	
togglebutton	**boolean2__togglebutton**
checkbutton	**boolean2__checkbutton**
output value label	**boolean2__var**
Row 3:	
radiobutton0	**radio__radiobutton0**
radiobutton1	**radio__radiobutton1**
radiobutton2	**radio__radiobutton2**
combobox	**radio__combobox**
output value label	**radio__var**
Row 4:	
spinbutton	**integer__spinbox**
entry	**integer__entry**

slider	**integer__slider**
output value label	**integer__var**
Row 5:	
entry	**float__entry**
output value label	**float__var**
Row 6:	
entry	**string__entry**
output value label	**string__var**
Row 7:	
textview	**textview__textview**
output value label	**textview__var**

- The specific **AVC** module for Qt3 is imported at program begin (`from avc.avcqt3 import *`).
- The application class is derived from the **QApplication** class of Qt3 and from the **AVC** class of AVC (`class Example(QApplication,AVC):`).
- The following variables are declared in the application.

```
self.boolean1 = False
self.boolean2 = False
self.radio = 0
self.integer = 0
self.float = 0.0
self.string = ''
self.textview = ''
```

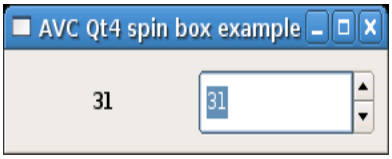
- The avc_init method is called after the instantation of the application class (`example.avc_init()`) to realize the connections of all widegts/variable combinations and to initialize the widgets values with the initial value of the connected variable.

Example files in directory 'examples' of distribution: program 'qt3_showcase.py', UI descriptor 'qt3_showcase.ui'.

10 Qt4 examples

10.1 Spin box example

For a functional description of the graphic interface see the GTK “Spin button example“ at page 17.



10.1.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2006 Fabrizio Pollastri
# .license    : GNU General Public License v3


from PyQt4.QtCore import *            # Qt core
from PyQt4.QtGui import *             # Qt GUI interface
from PyQt4.uic import *               # ui files realizer
import sys                            # system support

from avc.avcqt4 import *              # AVC for Qt4

UI_FILE = 'qt4_spinbox.ui'            # qt ui descriptor


class Example(QApplication,AVC):
  "A spin box whose value is replicated into a text label"

  def __init__(self):

    # create GUI
    QApplication.__init__(self,sys.argv)
    self.root = loadUi(UI_FILE)
    self.root.show()

    # the variable holding the spin box value
    self.spin_value = 0


#### MAIN

example = Example()                   # instantiate the application
example.avc_init()                    # connect widgets with variables
example.exec_()                       # run Qt event loop until quit

#### END
```

The GUI layout was previously edited with Qt4 Designer and saved to the file ‘qt4_spinbox.ui’.

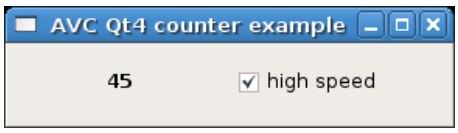
The key points of the example regarding **AVC** are the following.

- During Qt4 Designer editing, the name '**spin_value__spinbox**' was given to the spin box and the name '**spin_value__label**' was given to the label.
- The specific **AVC** module for Qt4 is imported at program begin (`from avc.avcqt4 import *`).
- The application class is derived from the **QApplication** class of Qt4 and from the **AVC** class of AVC (`class Example(QApplication,AVC):`).
- A integer variable with an initial value of 0 and name '**spin_value**' is declared in the application (`self.spin_value = 0`).
- The avc_init method is called after the instantation of the application class (`example.avc_init()`) to realize the connections of the two widgets through the '**spin_value**' variable and to initialize the widgets values with the initial value of the variable .

Example files in directory 'examples' of distribution: program 'qt4_spinbox.py', UI descriptor 'qt4_spinbox.ui'.

10.2 Counter example

For a functional description of the graphical interface see the GTK “Counter example“ at page 18.



10.2.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2006 Fabrizio Pollastri
# .license    : GNU General Public License v3


from PyQt4.QtCore import *            # Qt core
from PyQt4.QtGui import *             # Qt GUI interface
from PyQt4.uic import *               # ui files realizer
import sys                            # system support

from avc.avcqt4 import *              # AVC for Qt4

UI_FILE = 'qt4_counter.ui'            # qt ui descriptor
LOW_SPEED = 500                       #--
HIGH_SPEED = 100                      #- low and high speed count period (ms)


class Example(QApplication,AVC):
  """
  A counter displayed in a Label widget whose count speed can be
  accelerated by checking a check box.
  """

  def __init__(self):

    # create GUI
```

```
    QApplication.__init__(self,sys.argv)
    self.root = loadUi(UI_FILE)
    self.root.show()

    # the counter variable and its speed status
    self.counter = 0
    self.high_speed = False

    # start counter incrementer at low speed
    self.timer = qt.QTimer(self)
    self.connect(self.timer,qt.SIGNAL("timeout()"),self.incrementer)
    self.timer.start(LOW_SPEED)


  def incrementer(self):
    """
    Counter incrementer: increment period = LOW_SPEED, if high speed
    is False, increment period = HIGH_SPEED otherwise.
    """
    self.counter += 1
    if self.high_speed:
      period = HIGH_SPEED
    else:
      period = LOW_SPEED
    self.timer.stop()
    self.timer.start(period)


#### MAIN

example = Example()                 # instantiate the application
example.avc_init()                  # connect widgets with variables
example.exec_()                     # run Qt event loop until quit

#### END
```

The GUI layout was previously edited with Qt4 Designer and saved to the file ‘qt4_counter.ui’.

The key points of the example regarding **AVC** are the following.

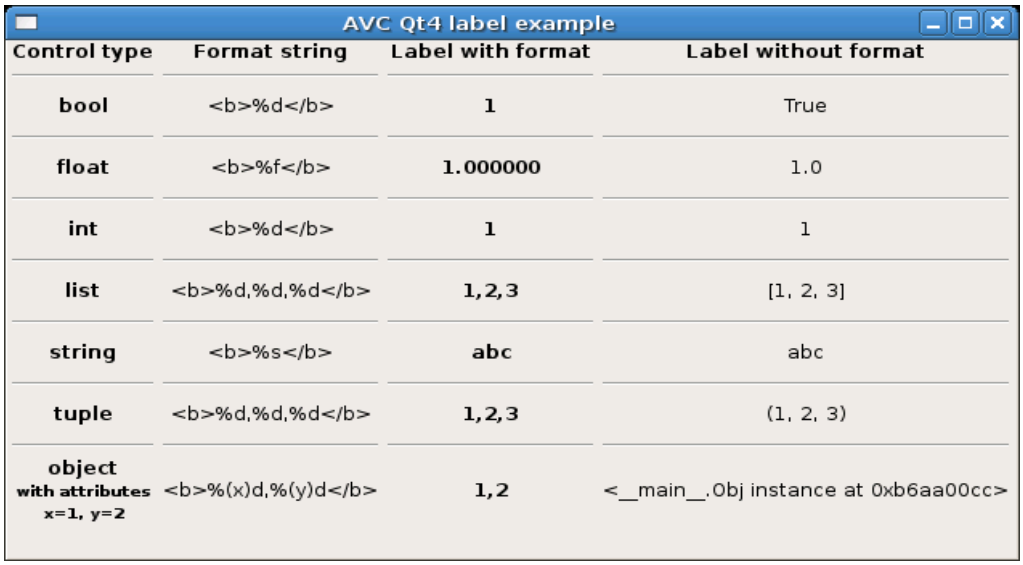
- During Qt4 Designer editing, the name '**counter**' was given to the label and the name '**high_speed**' was given to the check button.
- The specific **AVC** module for Qt4 is imported at program begin (`from avc.avcqt4 import *`).
- The application class is derived from the **QApplication** class of Qt4 and from the **AVC** class of AVC. (`class Example(QApplication,AVC):`).
- A integer variable with an initial value of 0 and name '**counter**' is declared in the application to hold the counter value (`self.counter = 0`).
- A boolean variable with an initial value of False and name '**high_speed**' is declared in the application to hold the speed status of the counter increment speed (`self.high_speed = False`).
- The avc_init method is called after the instantation of the application class (`example.avc_init()`) to realize the connections between the '**counter**' variable and the label widget and between the the '**high_speed**' variable and the check button, the label widget is initialized with the initial value of the '**counter**' variable .

Example files in directory 'examples' of distribution: program 'qt4_counter.py', UI descriptor

'qt4_counter.ui'.

10.3 Label example

This example shows the formatting capabilities of the label widget. For each supported type of the connected variable, a formatting string is defined and a sample value of the connected variable is displayed into two label widgets: one with formatting and the other with the standard python string representation.



10.3.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2008 Fabrizio Pollastri
# .license    : GNU General Public License v3


from PyQt4.QtCore import *            # Qt core
from PyQt4.QtGui import *             # Qt GUI interface
from PyQt4.uic import *               # ui files realizer
import sys                            # system support

from avc.avcqt4 import *              # AVC for Qt4

UI_FILE = 'qt4_label.ui'       # qt ui descriptor


class Example(QApplication,AVC):
  """
  Showcase of formatting capabilities for the label widget
  """

  def __init__(self):

    # create GUI
    QApplication.__init__(self,sys.argv)
    self.root = loadUi(UI_FILE)
```

```
    self.root.show()

    # all types of connected variables
    self.bool_value = True
    self.float_value = 1.0
    self.int_value = 1
    self.list_value = [1,2,3]
    self.str_value = 'abc'
    self.tuple_value = (1,2,3)
    class Obj:
      "A generic object with 2 attributes x,y"
      def __init__(self):
        self.x = 1
        self.y = 2
    self.obj_value = Obj()


#### MAIN

example = Example()                    # instantiate the application
example.avc_init()                     # connect widgets with variables
example.exec_()                        # run Qt event loop until quit

#### END
```

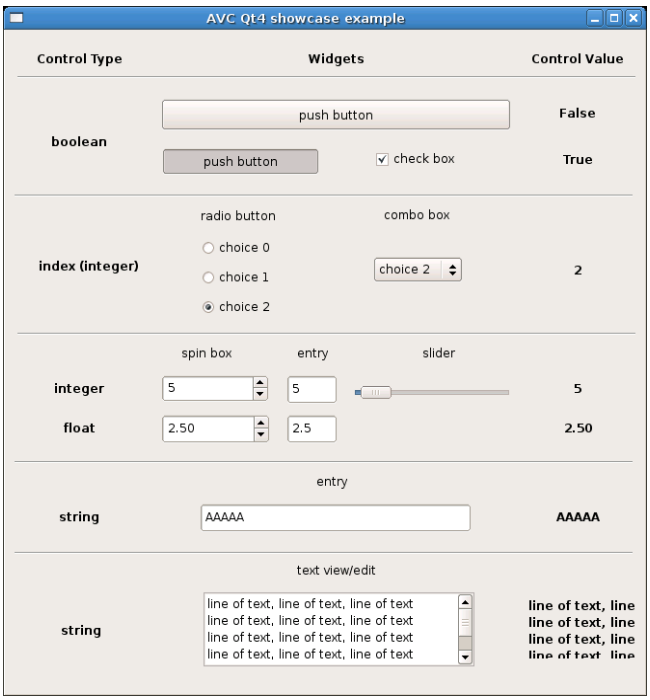
The GUI layout was previously edited with Qt4 Designer and saved to the file ‘qt4_label.ui’.

Apart the general requirements of AVC, already pointed out in the other examples, the relevant points of the label example about AVC are the following.

- For each control type (for each row) the two label widgets, one in the column “Label with format” and one in the column “Label without format”, are connected to the variable of the corresponding type. For example, in row “boolean”, both label widgets are called “bool_value”, so they connect to the variable `self.bool_value` .
- When the Qt4 event loop is entered both columns are set to display the initial values of the connected variables. For example, in row “integer”, both labels are set to display the integer value 1.
- The differences of representation between the column “Label with format” and the column “Label without format” reflect the different printout results coming from the formatting capabilities of the label widget and from `str`, the generic textual rendering function of python.

Example files in directory 'examples' of distribution: program 'qt4_label.py' , UI descriptor 'qt4_label.ui'.

10.4 Showcase example



This example shows a table of all widget/variable type combinations supported by **AVC**. The program creates a window with three columns: the first shows the type of the connected variable, the second shows all the widgets that can be connected to that type of variable, the third shows the current value of each variable. Each row of the window represent a widgets/variable combination.

- Row 1: memoryless button with boolean variable, pressed = True, unpressed = False.
- Row 2: buttons with memory, toggle and check buttons, pressed = True, unpressed = False.
- Row 3: mutually exclusive choices widgets, radio buttons numbered from 0 to 2 and a combo box with 3 items, index variable = number of checked radio button and selected item of combo box.
- Row 4: integer input/output widgets, spin button, entry and slider.
- Row 5: float input/output widgets, spin button and entry.
- Row 6: string input/output widget, entry.
- Row 7: string input/output widget, text view/edit.

The text label widget is used in all output modes for the column of the connected variable values. The program increment the value of each connected variable looping top-bottom at three rows per seconds. The user can also change the values of the connected variables interacting with the widgets.

10.4.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2006 Fabrizio Pollastri
# .license    : GNU General Public License v3
```

```
from PyQt4.QtCore import *           # Qt core
from PyQt4.QtGui import *            # Qt GUI interface
from PyQt4.uic import *              # ui files realizer
import sys                           # system support

from avc.avcqt4 import *             # AVC for Qt4

UI_FILE = 'qt4_showcase.ui'          # qt ui descriptor
INCREMENTER_PERIOD = 333             # ms


class Example(QApplication,AVC):
  "A table of all supported widget/control type combinations"

  def __init__(self):

    # create GUI
    QApplication.__init__(self,sys.argv)
    self.root = loadUi(UI_FILE)
    self.root.show()

    # group all radio buttons into a button group. Button group not
    # managed by Qt4 Designer ?!
    self.radio_button0 = self.root.findChild(QWidget,'radio__button0')
    self.radio_button1 = self.root.findChild(QWidget,'radio__button1')
    self.radio_button2 = self.root.findChild(QWidget,'radio__button2')
    self.radio_button_group = QButtonGroup()
    self.radio_button_group.addButton(self.radio_button0,0)
    self.radio_button_group.addButton(self.radio_button1,1)
    self.radio_button_group.addButton(self.radio_button2,2)

    # the control variables
    self.boolean1 = False
    self.boolean2 = False
    self.radio = 0
    self.integer = 0
    self.float = 0.0
    self.string = ''
    self.textview = ''

    # start variables incrementer
    self.increment = self.incrementer()
    self.timer = QTimer(self)
    self.connect(self.timer,SIGNAL("timeout()"),self.timer_function)
    self.timer.start(int(INCREMENTER_PERIOD))


  def timer_function(self):
    self.increment.next()


  def incrementer(self):
    """
    Booleans are toggled, radio button index is rotated from first to last,
    integer is incremented by 1, float by 0.5, string is appended a char
    until maxlen when string is cleared, text view/edit is appended a line
    of text until maxlen when text is cleared, status bar message is toggled.
    Return True to keep timer alive.
    """
    while True:
```

```
      self.boolean1 = not self.boolean1
      yield True

      self.boolean2 = not self.boolean2
      yield True

      if self.radio == 2:
        self.radio = 0
      else:
        self.radio += 1
      yield True

      self.integer += 1
      yield True

      self.float += 0.5
      yield True

      if len(self.string) >= 10:
        self.string = 'A'
      else:
        self.string += 'A'
      yield True

      if len(self.textview) >= 200:
        self.textview = ''
      else:
        self.textview += 'line of text, line of text, line of text\n'
      yield True


#### MAIN

example = Example()                   # instantiate the application
example.avc_init()                    # connect widgets with variables
example.exec_()                       # run Qt event loop until quit

#### END
```

The GUI layout was previously edited with Qt4 Designer and saved to the file ‘qt4_showcase.ui’.

The key points of the example regarding **AVC** are the following.

- During Qt designer editing, the following names were given to the widgets.

widget	name
Row 1:	
button	**boolean1__button**
output value label	**boolean1__var**
Row 2:	
togglebutton	**boolean2__togglebutton**
checkbutton	**boolean2__checkbutton**
output value label	**boolean2__var**
Row 3:	
radiobutton0	**radio__radiobutton0**
radiobutton1	**radio__radiobutton1**
radiobutton2	**radio__radiobutton2**

combobox	**radio__combobox**
output value label	**radio__var**
Row 4:	
spinbutton	**integer__spinbox**
entry	**integer__entry**
slider	**integer__slider**
output value label	**integer__var**
Row 5:	
spinbutton	**float__spinbutton**
entry	**float__entry**
output value label	**float__var**
Row 6:	
entry	**string__entry**
output value label	**string__var**
Row 7:	
textview	**textview__textview**
output value label	**textview__var**

- The specific **AVC** module for Qt4 is imported at program begin (`from avc.avcqt4 import *`).
- The application class is derived from the **QApplication** class of Qt4 and from the **AVC** class of AVC (`class Example(QApplication,AVC):`).
- The following variables are declared in the application.

```
self.boolean1 = False
self.boolean2 = False
self.radio = 0
self.integer = 0
self.float = 0.0
self.string = ''
self.textview = ''
```

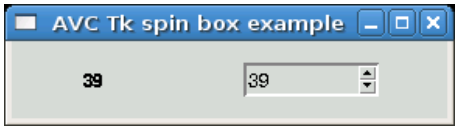
- The avc_init method is called after the instantation of the application class (`example.avc_init()`) to realize the connections of all widegts/variable combinations and to initialize the widgets values with the initial value of the connected variable.

Example files in directory 'examples' of distribution: program 'qt4_showcase.py', UI descriptor 'qt4_showcase.ui'.

11 Tk examples

11.1 Spin box example

For a functional description of the graphical interface see the GTK “Spin button example“ at page 17 .



11.1.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2007 Fabrizio Pollastri
# .license    : GNU General Public License v3


from Tkinter import *                 # Tk interface

from avc.avctk import *               # AVC for Tk

TCL_FILE = 'tk_spinbox.tcl'           # GUI description as tcl script


class Example(AVC):
  """
  A spin control whose value is replicated into a label
  """

  def __init__(self):

    # create GUI
    self.root = Tk()
    self.root.eval('set argc {}; set argv {}; proc ::main {argc argv} {};')
    self.root.tk.evalfile(TCL_FILE)

    # terminate program at toplevel window destroy: connect toplevel
    # destroy signal to termination handler.
    self.root.bind_class('Toplevel','<Destroy>',lambda event: self.root.quit())

    # the variable holding the spin control value
    self.spin_value = 0


#### MAIN

example = Example()                   # instantiate the application
example.avc_init()                    # connect widgets with variables
Tkinter.mainloop()                    # run Tk event loop until quit

#### END
```

The GUI layout was previously edited with Visual Tcl and saved to the file ‘tk_spinbox.tcl’.

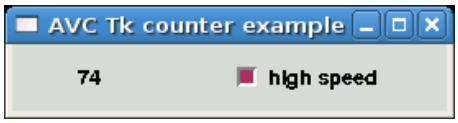
The key points of the example regarding **AVC** are the following.

- During Visual Tcl editing, the name '**spin_value__spinbox**' was given to the spin box and the name '**spin_value__label**' was given to the label.
- The specific **AVC** module for Tk is imported at program begin (`from avc.avctk import *`).
- The application class is derived from the from the **AVC** class of AVC (`class Example(AVC):`).
- A integer variable with an initial value of 0 and name '**spin_value**' is declared in the application (`self.spin_value = 0`).
- The avc_init method is called after the instantation of the application class, to realize the connections of the two widgets through the '**spin_value**' variable and to initialize the widgets values with the initial value of the variable (`example.avc_init()`).

Example files in directory 'examples' of distribution: program 'tk_spinbox.py', graphic interface descriptor as tcl script 'tk_spinbox.tcl'.

11.2 Counter example

For a functional description of the graphical interface see the GTK “Counter example“ at page 18.



11.2.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2006 Fabrizio Pollastri
# .license    : GNU General Public License v3


from Tkinter import *                  # Tk interface

from avc.avctk import *                # AVC for Tk

TCL_FILE = 'tk_counter.tcl'            # GUI description as tcl script
LOW_SPEED = 500                        #--
HIGH_SPEED = 100                       #- low and high speed count period (ms)


class Example(AVC):
  """
  A counter displayed in a Label widget whose count speed can be doubled
  by pressing a Toggle Button.
  """

  def __init__(self):

    # create GUI
    self.root = Tk()
    self.root.eval('set argc {}; set argv {}; proc ::main {argc argv} {};')
    self.root.tk.evalfile(TCL_FILE)

    # terminate program at toplevel window destroy: connect toplevel
    # destroy signal to termination handler.
```

```
    self.root.bind_class('Toplevel','<Destroy>',lambda event: self.root.quit())

    # the counter variable and its speed status
    self.counter = 0
    self.high_speed = False

    # start counter incrementer at low speed
    self.root.after(LOW_SPEED,self.incrementer)


  def incrementer(self):
    """
    Counter incrementer: increment period = LOW_SPEED, if high speed is False,
    increment period = HIGH_SPEED otherwise.
    """
    self.counter += 1
    if self.high_speed:
      period = HIGH_SPEED
    else:
      period = LOW_SPEED
    self.root.after(period,self.incrementer)


#### MAIN

example = Example()                  # instantiate the application
example.avc_init()              # connect widgets with variables
Tkinter.mainloop()              # run Tk event loop until quit

#### END
```

The GUI layout was previously edited with Visual Tcl and saved to the file ‘tk_counter.tcl’.

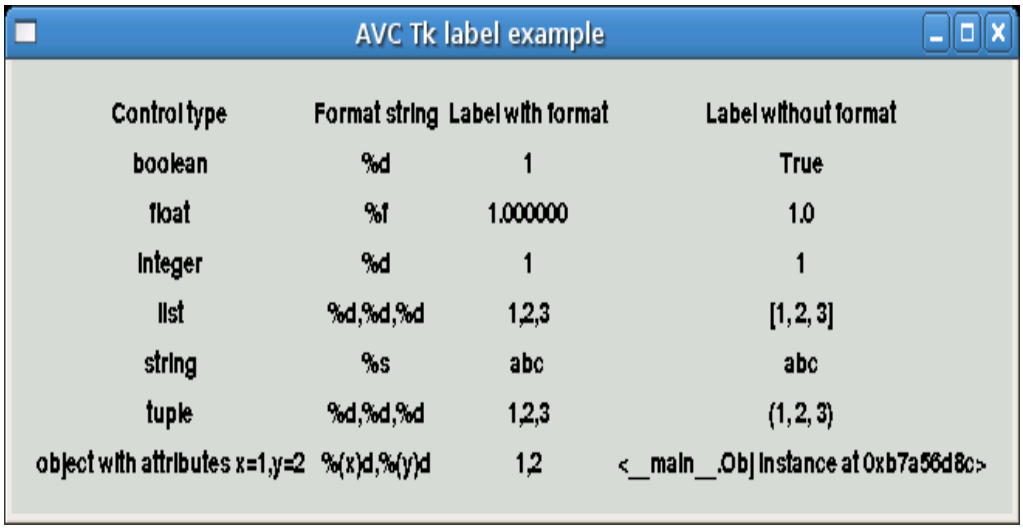
The key points of the example regarding **AVC** are the following.

- During Visual Tcl editing, the name '**counter**' was given to the label and the name '**high_speed**' was given to the check button.
- The specific **AVC** module for Tk is imported at program begin (`from avc.avctk import *`).
- The application class is derived from the **AVC** class of AVC. (`class Example(AVC):`).
- A integer variable with an initial value of 0 and name '**counter**' is declared in the application to hold the counter value (`self.counter = 0`).
- A boolean variable with an initial value of False and name '**high_speed**' is declared in the application to hold the speed status of the counter increment (`self.high_speed = False`).
- The avc_init method is called after the instantation of the application class (`example.avc_init()`) to realize the connections between the '**counter**' variable and the label widget and between the the '**high_speed**' variable and the check button, the label widget is initialized with the initial value of the '**counter**' variable .

Example files in directory 'examples' of distribution: program 'tk_counter.py', graphic interface descriptor as tcl script 'tk_counter.tcl'.

11.3 Label example

This example shows the formatting capabilities of the label widget. For each supported type of the connected variable, a formatting string is defined and a sample value of the connected variable is displayed into two label widgets: one with formatting and the other with the standard python string representation.



11.3.1 Python source

```
#!/usr/bin/python
# .copyright   : (c) 2008 Fabrizio Pollastri
# .license     : GNU General Public License v3


from Tkinter import *                 # Tk interface

from avc.avctk import *               # AVC for Tk

TCL_FILE = 'tk_label.tcl'             # GUI description as tcl script


class Example(AVC):
  """
  Showcase of formatting capabilities for the label widget
  """

  def __init__(self):

    # create GUI
    self.root = Tk()
    self.root.eval('set argc {}; set argv {}; proc ::main {argc argv} {};')
    self.root.tk.evalfile(TCL_FILE)

    # terminate program at toplevel window destroy: connect toplevel
    # destroy signal to termination handler.
    self.root.bind_class('Toplevel','<Destroy>',lambda event: self.root.quit())

    # all types of connected variables
    self.bool_value = True
```

```
    self.float_value = 1.0
    self.int_value = 1
    self.list_value = [1,2,3]
    self.str_value = 'abc'
    self.tuple_value = (1,2,3)
    class Obj:
      "A generic object with 2 attributes x,y"
      def __init__(self):
        self.x = 1
        self.y = 2
    self.obj_value = Obj()


#### MAIN

example = Example()                    # instantiate the application
example.avc_init()                     # connect widgets with variables
Tkinter.mainloop()                     # run Tk event loop until quit

#### END
```

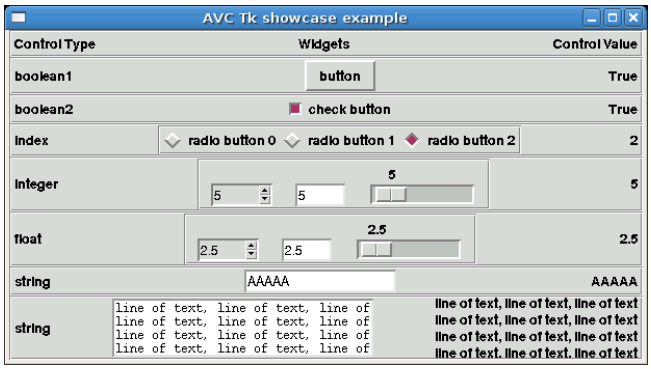
The GUI layout was previously edited with Visual Tcl and saved to the file ‘tk_label.tcl’.

Apart the general requirements of AVC, already pointed out in the other examples, the relevant points of the label example about AVC are the following.

- For each control type (for each row) the two label widgets, one in the column “Label with format” and one in the column “Label without format”, are connected to the variable of the corresponding type. For example, in row “boolean”, both label widgets are called “bool_value”, so they connect to the variable `self.bool_value` .
- When the Tk event loop is entered both columns are set to display the initial values of the connected variables. For example, in row “integer”, both labels are set to display the integer value 1.
- The differences of representation between the column “Label with format” and the column “Label without format” reflect the different printout results coming from the formatting capabilities of the label widget and from `str`, the generic textual rendering function of python.

Example files in directory 'examples' of distribution: program 'tk_label.py', graphic interface descriptor as tcl script 'tk_label.tcl'.

11.4 Showcase example



This example shows a table of all widget/variable type combinations supported by **AVC**. The program creates a window with three columns: the first shows the type of the connected variable, the second shows all the widgets that can be connected to that type of variable, the third shows the current value of each variable. Each row of the window represent a widgets/variable combination as follows.

- Row 1: memoryless button with boolean variable, pressed = True, unpressed = False.
- Row 2: button with memory, check button, pressed = True, unpressed = False.
- Row 3: mutually exclusive choices widgets, radio buttons numbered from 0 to 2, index variable = number of checked radio button.
- Row 4: integer input/output widgets, spin button, entry and slider.
- Row 5: float input/output widgets, spin button, entry and slider.
- Row 6: string input/output widget, entry.
- Row 7: string input/output widget, text view/edit.

The text label widget is used in all output modes for the column of the connected variable values. The program increment the value of each connected variable looping top-bottom at three rows per seconds. The user can also change the values of the connected variables interacting with the widgets.

11.4.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2007 Fabrizio Pollastri
# .license    : GNU General Public License v3


from Tkinter import *                   # Tk interface

from avc.avctk import *                 # AVC for Tk

TCL_FILE = 'tk_showcase.tcl'            # GUI description as tcl script
INCREMENTER_PERIOD = 0.333              # seconds


class Example(AVC):
  "A table of all supported widget/control type combinations"

  def __init__(self):
```

```
    # create GUI
    self.root = Tk()
    self.root.eval('set argc {}; set argv {}; proc ::main {argc argv} {};')
    self.root.tk.evalfile(TCL_FILE)

    # terminate program at toplevel window destroy: connect toplevel
    # destroy signal to termination handler.
    self.root.bind_class('Toplevel','<Destroy>',lambda event: self.root.quit())

    # the control variables
    self.boolean1 = False
    self.boolean2 = False
    self.radio = 0
    self.integer = 0
    self.float = 0.0
    self.string = ''
    self.textview = ''

    # start variables incrementer
    increment = self.incrementer()
    self.timer_function = increment.next
    self.root.after(int(INCREMENTER_PERIOD * 1000.0),self.timer_wrap)


  def timer_wrap(self):
    "Call given function, reschedule it after return"
    self.timer_function()
    self.root.after(int(INCREMENTER_PERIOD * 1000.0),self.timer_wrap)


  def incrementer(self):
    """
    Booleans are toggled, radio button index is rotated from first to last,
    integer is incremented by 1, float by 0.5, string is appended a char
    until maxlen when string is cleared, text view/edit is appended a line
    of text until maxlen when it is cleared.
    Return True to keep timer alive.
    """
    while True:
      self.boolean1 = not self.boolean1
      yield True

      self.boolean2 = not self.boolean2
      yield True

      if self.radio == 2:
        self.radio = 0
      else:
        self.radio += 1
      yield True

      self.integer += 1
      yield True

      self.float += 0.5
      yield True

      if len(self.string) >= 20:
        self.string = 'A'
      else:
        self.string += 'A'
```

```
      yield True

      if len(self.textview) >= 200:
        self.textview = ''
      else:
        self.textview += 'line of text, line of text, line of text\n'
      yield True


#### MAIN

example = Example()                    # instantiate the application
example.avc_init()                     # connect widgets with variables
Tkinter.mainloop()                     # run Tk event loop until quit

#### END
```

The GUI layout was previously edited with Visual Tcl and saved to the file 'tk_showcase.tcl'.

The key points of the example regarding **AVC** are the following.

- During Visual Tcl editing, the following names were given to the widgets.

Row	widget	name
1	button	**boolean1__button**
	output value label	**boolean1__var**
2	checkbutton	**boolean2__checkbutton**
	output value label	**boolean2__var**
3	radiobutton0	**radio__radiobutton0**
	radiobutton1	**radio__radiobutton1**
	radiobutton2	**radio__radiobutton2**
	output value label	**radio__var**
4	spinbutton	**integer__spinbox**
	entry	**integer__entry**
	slider	**integer__hscale**
	output value label	**integer__var**
5	spinbutton	**float__spinbox**
	entry	**float__entry**
	slider	**float__hscale**
	output value label	**float__var**
6	entry	**string__entry**
	output value label	**string__var**
7	textview	**textview__textview**
	output value label	**textview__var**

- The specific **AVC** module for Tk is imported at program begin (`from avc.avctk import *`).
- The application class is derived from the **AVC** class (`class Example(AVC):`).
- The following variables are declared in the application.

```
      self.boolean1 = False
      self.boolean2 = False
      self.radio = 0
```

```
        self.integer = 0
        self.float = 0.0
        self.string = ''
        self.textview = ''
        self.status = ''
```

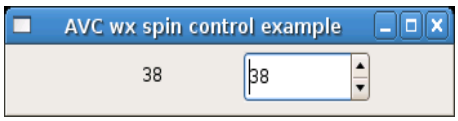
- The avc_init method is called after the instantation of the application class (`example.avc_init()`) to realize the connections of all widegts/variable combinations and to initialize the widgets values with the initial value of the connected variable .

Example files in directory 'examples' of distribution: program 'tk_showcase.py', graphic interface descriptor as tcl script 'tk_showcase.tcl'.

12 wxWidgets examples

12.1 Spin control example

For a functional description of the graphic interface see the GTK “Spin button example“ at page 17.



12.1.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2007 Fabrizio Pollastri
# .license    : GNU General Public License v3


import wx                               # wx tool kit bindings
from wx import xrc                      # xml resource support

from avc.avcwx import *                 # AVC for wx

WXGLADE_XML = 'wx_spinctrl.xrc'         # GUI wxGlade descriptor


class Example(wx.PySimpleApp,AVC):
  """
  A spin button whose value is replicated into a static text
  """

  def __init__(self):

    ## create GUI

    # init wx application base class
    wx.PySimpleApp.__init__(self)

    # create GUI
    xml_resource = xrc.XmlResource(WXGLADE_XML)
    self.root = xml_resource.LoadFrame(None,'frame_1')
    self.root.Show()

    ## the variable holding the spin button value
    self.spin_value = 0


#### MAIN

example = Example()                     # instantiate the application
example.avc_init()                      # connect widgets with variables
example.MainLoop()                      # run wx event loop until quit

#### END
```

The GUI layout was previously edited with wxGlade and saved to the file ‘wx_spinctrl.xrc’.

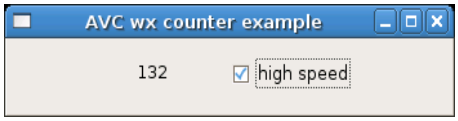
The key points of the example regarding **AVC** are the following.

- During wxGlade editing, the same name '**spin_value**' was given to the spin button and to the label.
- The specific **AVC** module for wxWidgets is imported at program begin (`from avc.avcwx import *`).
- The application class is derived from the class **PySimpleApp** of wxWidgets and from the class **AVC** of AVC (`class Example(wx.PySimpleApp,AVC):`).
- A integer variable with an initial value of 0 and name '**spin_value**' is declared in the application (`self.spin_value = 0`).
- The avc_init method is called after the instantation of the application class, to realize the connections of the two widgets through the '**spin_value**' variable and to initialize the widgets values with the initial value of the variable (`example.avc_init()`).

Example files in directory 'examples' of distribution: program 'wx_spinctrl.py' , UI descriptor 'wx_spinctrl.xrc'.

12.2 Counter example

For a functional description of the graphical interface see the GTK “Counter example“ at page 18.



12.2.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2007 Fabrizio Pollastri
# .license    : GNU General Public License v3

import wx                               # wx tool kit bindings
from wx import xrc                      # xml resource support

from avc.avcwx import *                 # AVC for wx

WXGLADE_XML = 'wx_counter.xrc'          # GUI wxGlade descriptor
LOW_SPEED = 500                         #--
HIGH_SPEED = 100                        #- low and high speed period (ms)


class Example(wx.PySimpleApp,AVC):
  """
  A counter displayed in a Label widget whose count speed can be
  accelerated by checking a check button.
  """

  def __init__(self):

    # init wx application base class
    wx.PySimpleApp.__init__(self)
```

```
    # create GUI
    xml_resource = xrc.XmlResource(WXGLADE_XML)
    self.root = xml_resource.LoadFrame(None,'frame_1')
    self.root.Show()

    ## the counter variable and its speed status
    self.counter = 0
    self.high_speed = False

    # start counter incrementer at low speed
    self.timer = wx.Timer(self.root,wx.NewId())
    self.root.Bind(wx.EVT_TIMER,self.incrementer,self.timer)
    self.timer.Start(LOW_SPEED,oneShot=True)


  def incrementer(self,event):
    """
    Counter incrementer: increment period = LOW_SPEED, if high speed is False,
    increment period = HIGH_SPEED otherwise. Return False to destroy previous
    timer.
    """
    self.counter += 1
    if self.high_speed:
      period = HIGH_SPEED
    else:
      period = LOW_SPEED
    self.timer.Start(period,oneShot=True)

  def high_speed_changed(self,value):
    "Notify change of counting speed to terminal"
    if value:
      print 'counting speed changed to high'
    else:
      print 'counting speed changed to low'


#### MAIN

example = Example()                 # instantiate the application
example.avc_init()                  # connect widgets with variables
example.MainLoop()                  # run wx event loop until quit

#### END
```

The GUI layout was previously edited with wxGlade and saved to the file ‘wx_counter.xrc’.

The key points of the example regarding **AVC** are the following.

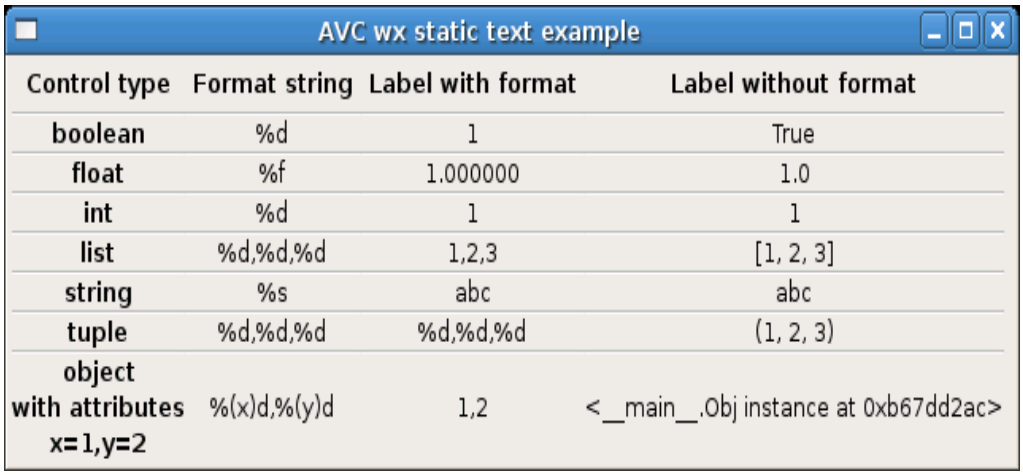
- During wxGlade editing, the name '**counter**' was given to the static text and the name '**high_speed**' was given to the check box.
- The specific **AVC** module for GTK is imported at program begin (`from avc.avcwx import *`).
- The application class is derived from the class **PySimpleApp** fo wxWidgets and from the class **AVC** of AVC (`class Example(wx.PySimpleApp,AVC):`).
- A integer variable with an initial value of 0 and name '**counter**' is declared in the application to hold the counter value (`self.counter = 0`).
- A boolean variable with an initial value of False and name '**high_speed**' is declared in the application to hold the speed status of the counter increment speed (`self.high_speed = False`).

- The avc_init method is called after the instantation of the application class (`example.avc_init()`) to realize the connections between the '**counter**' variable and the label widget and between the the '**high_speed**' variable and the check button, the label widget is initialized with the initial value of the '**counter**' variable.

Example files in directory 'examples' of distribution: program 'wx_counter.py' , UI descriptor 'wx_counter.xrc'.

12.3 Label example

This example shows the formatting capabilities of the label widget. For each supported type of the connected variable, a formatting string is defined and a sample value of the connected variable is displayed into two label widgets: one with formatting and the other with the standard python string representation.



12.3.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2008 Fabrizio Pollastri
# .license    : GNU General Public License v3


import wx                               # wx tool kit bindings
from wx import xrc                      # xml resource support

from avc.avcwx import *                 # AVC for wx

WXGLADE_XML = 'wx_label.xrc'            # GUI wxGlade descriptor


class Example(wx.PySimpleApp,AVC):
  """
  Showcase of formatting capabilities for the label widget
  """

  def __init__(self):

    # init wx application base class
```

```
    wx.PySimpleApp.__init__(self)

    # create GUI
    xml_resource = xrc.XmlResource(WXGLADE_XML)
    self.root = xml_resource.LoadFrame(None,'frame_1')
    self.root.Show()

    # all types of connected variables
    self.bool_value = True
    self.float_value = 1.0
    self.int_value = 1
    self.list_value = [1,2,3]
    self.str_value = 'abc'
    self.tuple_value = (1,2,3)
    class Obj:
      "A generic object with 2 attributes x,y"
      def __init__(self):
        self.x = 1
        self.y = 2
    self.obj_value = Obj()


#### MAIN

example = Example()                 # instantiate the application
example.avc_init()                  # connect widgets with variables
example.MainLoop()                  # run wx event loop until quit

#### END
```

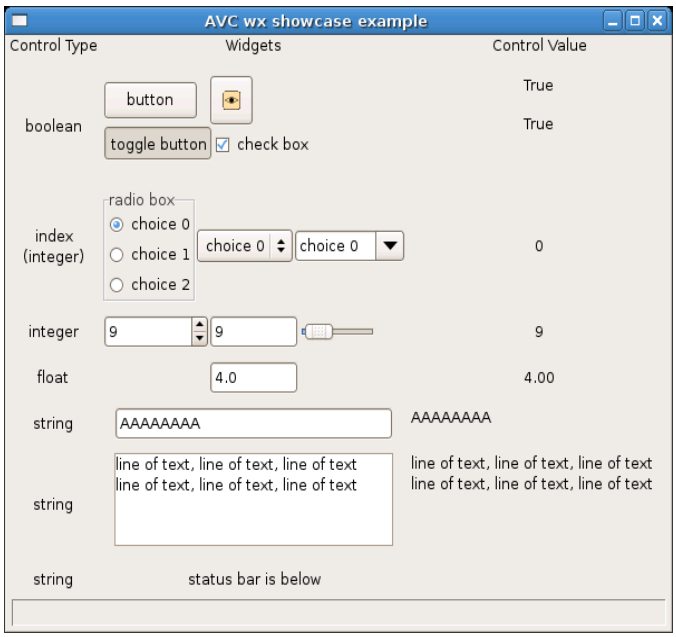
The GUI layout was previously edited with wxGlade and saved to the file ‘wx_label.xrc’.

Apart the general requirements of AVC, already pointed out in the other examples, the relevant points of the label example about AVC are the following.

- For each control type (for each row) the two label widgets, one in the column “Label with format” and one in the column “Label without format”, are connected to the variable of the corresponding type. For example, in row “boolean”, both label widgets are called “bool_value”, so they connect to the variable `self.bool_value` .
- When the wxWidget event loop is entered both columns are set to display the initial values of the connected variables. For example, in row “integer”, both labels are set to display the integer value 1.
- The differences of representation between the column “Label with format” and the column “Label without format” reflect the different printout results coming from the formatting capabilities of the label widget and from `str`, the generic textual rendering function of python.

Example files in directory 'examples' of distribution: program 'wx_label.py', UI descriptor 'wx_label.xrc'.

12.4 Showcase example



This example shows a table of all widget/variable type combinations supported by **AVC**. The program creates a window with three columns: the first shows the type of the connected variable, the second shows all the widgets that can be connected to that type of variable, the third shows the current value of each variable. Each row of the window represent a widgets/variable combination as follows.

- Row 1: memoryless button and bitmap button with boolean variable, pressed = True, unpressed = False.
- Row 2: buttons with memory, toggle and check box, pressed = True, unpressed = False.
- Row 3: mutually exclusive choices widgets, radio box buttons numbered from 0 to 2, a choice with 3 items and a combo box with 3 items, index variable = number of checked radio button and selected item of combo box.
- Row 4: integer input/output widgets, spin control, text control and slider.
- Row 5: float input/output widget, text control.
- Row 6: string input/output widget, text control.
- Row 7: string input/output widget, text control view/edit.
- Row 8: status messages, status bar.

The text label widget is used in all output modes for the column of the connected variable values. The program increment the value of each connected variable looping top-bottom at three rows per seconds. The user can also change the values in the connected variables interacting with the widgets.

12.4.1 Python source

```
#!/usr/bin/python
# .copyright  : (c) 2007 Fabrizio Pollastri
# .license    : GNU General Public License v3
```

```
import wx                               # wx tool kit bindings
from wx import xrc                      # xml resource support

from avc.avcwx import *                 # AVC for wx

WXGLADE_XML = 'wx_showcase.xrc'         # GUI wxGlade descriptor
INCREMENTER_PERIOD = 333                # ms


class Example(wx.PySimpleApp,AVC):
  "A table of all supported widget/control type combinations"

  def __init__(self):

    # init wx application base class
    wx.PySimpleApp.__init__(self)

    # create GUI
    xml_resource = xrc.XmlResource(WXGLADE_XML)
    self.root = xml_resource.LoadFrame(None,'frame_1')
    self.root.Show()

    # the control variables
    self.boolean1 = False
    self.boolean2 = False
    self.index = 0
    self.integer = 0
    self.float = 0.0
    self.string = ''
    self.textview = ''
    self.status = ''

    # start counter incrementer at low speed
    self.timer = wx.Timer(self.root,wx.NewId())
    self.root.Bind(wx.EVT_TIMER,self.incrementer_wrap,self.timer)
    self.timer.Start(int(INCREMENTER_PERIOD),oneShot=False)
    self.increment = self.incrementer()


  def incrementer_wrap(self,event):
    "Discard event argument and call the real incrementer iterator"
    self.increment.next()

  def incrementer(self,*args):
    """
    Booleans are toggled, radio button index is rotated from first to last,
    integer is incremented by 1, float by 0.5, string is appended a char
    until maxlen when string is cleared, text view/edit is appended a line
    of text until maxlen when it is cleared. Status bar message is toggled.
    Return True to keep timer alive.
    """
    while True:

      self.boolean1 = not self.boolean1
      yield True

      self.boolean2 = not self.boolean2
      yield True

      if self.index >= 2:
        self.index = 0
```

```
      else:
        self.index += 1
      yield True

      self.integer += 1
      yield True

      self.float += 0.5
      yield True

      if len(self.string) >= 10:
        self.string = ''
      else:
        self.string += 'A'
      yield True

      if len(self.textview) >= 200:
        self.textview = ''
      else:
        self.textview += 'line of text, line of text, line of text\n'
      yield True

      if not self.status:
        self.status = 'status message'
      else:
        self.status = ''
      yield True


#### MAIN

example = Example()                    # instantiate the application
example.avc_init()                     # connect widgets with variables
example.MainLoop()                     # run wx event loop until quit

#### END
```

The GUI layout was previously edited with wxGlade and saved to the file ‘wx_showcase.xrc’.

The key points of the example regarding **AVC** are the following.

- During Glade editing, the following names were given to the widgets.

Row	widget	name
1	button	**boolean1__button**
	bitmap button	**boolean1__bitmapbutton**
	output value label	**boolean1__var**
2	togglebutton	**boolean2__togglebutton**
	checkbox	**boolean2__checkbox**
	output value label	**boolean2__var**
3	radiobox	**index__radiobox**
	choice	**index__choice**
	combobox	**index__combobox**
	output value label	**index__var**
4	spinctrl	**integer__spinctrl**
	textctrl	**integer__textctrl**
	slider	**integer__slider**
	output value label	**integer__var**
5	textctrl	**float__entry**
	output value label	**float__var**

6	textctrl	**string__textctrl**
	output value label	**string__var**
7	textctrl	**textview__textctrl**
	output value label	**textview__var**
8	statusbar	**status__statusbar**
	output value label	**status__var**

- The specific **AVC** module for GTK is imported at program begin (`from avc.avcwx import *`).
- The application class is derived from the class PySimpleApp of wxWidgets and from the class **AVC** of AVC (`class Example(wx.PySimpleApp,AVC):`).
- The following variables are declared in the application.

```
self.boolean1 = False
self.boolean2 = False
self.index = 0
self.integer = 0
self.float = 0.0
self.string = ''
self.textview = ''
self.status = ''
```

- The avc_init method is called after the instantation of the application class (`example.avc_init()`) to realize the connections of all widegts/variable combinations and to initialize the widgets values with the initial value of the connected variable .

Example files in directory 'examples' of distribution: program 'wx_showcase.py' , UI descriptor 'wx_showcase.xrc'.

13 References

[1] Python, http://www.python.org/

[2] GTK, http://www.gtk.org/

[3] Qt3, http://trolltech.com/products/qt/qt3/

[4] Qt4, http://trolltech.com/products/qt/

[5] Tk, http://www.tcl.tk/

[6] wxWidgets, http://www.wxwidgets.org/

[7] Pygtk, http://www.pygtk.org/

[8] PyQt v3 and v4, http://www.riverbankcomputing.co.uk/pyqt/

[9] Tkinter, http://effbot.org/tkinterbook/

[10] wxPython, http://www.wxpython.org/

[11] Glade, http://glade.gnome.org/

[12] Qt designer, http://trolltech.com/products/qt/features/designer/

[13] Visual Tcl, http://vtcl.sourceforge.net/

[14] wxGlade, http://www.wxglade.org/

[15] GNU General Public License, http://www.gnu.org/licenses/gpl.html/

[16] GNU Free Documentation License, http://www.gnu.org/copyleft/fdl.html

GNU Free Documentation License
Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially.

Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document traightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

END