Intuition ++
Tutorial

Contact :

Brulhart Dominique
12, rue Lissignol
1201 Geneva
Switzerland

Phone : (41 22) 738 34 38

E-Mail : brulhart@cuilima.unige.ch

Intuition++ Tutorial

This is the first release of I++ I wrote in one week, so be indulgent. The goal was to create classes for Windows and Screens that could be more easy to create, don't need initialisation and cleaning from the user and provide a way of handling events and messages easily.

So I created 12 classes shortly described below. A list of all methods are given in file 'Ipp_Reference_Guide'.

The general way of using I++ is proach of XView programming, and may seem different of normal Intuition, Exec and Graphics using. This is one approach, but it's be pleasant to program with.

In fact, if you know Intuition and C++, you know I++. Here are explained only methods that are not standard in Intuition.

Description of all classes:

1) The **CFont** class, which only opens a font and keep a pointer to it so it can be used by windows and screens.

2) The **CRastPortHdl** class, which only handles a RastPort the user pass to its constructor or to a method which turn handling on. So, a window or a screen which possess a valid RastPort can handle it trough this class. No initialisation, creation, etc.. of RastPort is made by it.

3) The **CWindow** class, which manages a NewWindow and TagItem and a Window pointer. An instance of CWindow remembers all its parameters between close() and open(). This class control window opening, closing, sizing, positioning, changing its flags, etc...

4) The **GfxWindow** class, which inherits class CWindow and class CRastPortHdl. This is a complete graphic window encapsulation.

5) The **IMessage** and **IEvent** classes, which only encapsulates IntuiMessage and a defines what I think an event is.

6) The **MsgWindow** class, which is the heart of event handling. It derives class CWindow for general window control and implement the following not standard methods:

- linkIevent(class, code, qualifier, object, callback)

    links an event to the window, so when it receives a message corresponding to the event, the desired callback is executed. For Gadget pass GADGETUP/DOWN for class, NULL for code, qualifier if you want, the address of the Gadget for object and a pointer to the function you to be executed when the Gadget is clicked for callback.
    For other types of message pass the desired class, code, qualifier, object that you normally test with Intuition.

    The callback must be declared like that:

```
void mycallback(IMessage& message)
{
        ....
}
```

    so you can watch to message which bring you there.
    See examples for further explanation.

- rmIevents()

    removes all events linked to the window.

- getImsg(message)

    standard GetMsg()

- waitImsg(message)

    standard WaitMsg() + GetMsg()

- clearImsg()

    clears UserPort message queue

- filterImsg(message)

  tries to handle the passed message with the
  list of events for the current window.
  Return the message only if no events could
  have been matched and callback executed.

- softcontrol(message)

  takes control of program execution and handle
  message flow by executing appropriate callback
  This method returns only if a message could not
  have been handled.

- hardcontrol()

  takes control of program execution, execute the
  appropriate callback for message it receives.
  Warning : hardcontrol() never returns, unap-
  prpriate messages are ignored.

7) The **MGWindow** class, which simply inherits GfxWindow and MsgWindow. This is a complete window handling.

8) The **Waiter** class, to which you can link as many window you want. The Waiter waits for messages concerning one of the windows you linked to it, and dispatch them to the right one.
Non standard methods:

- linkwindow(window)

  links a window to it.

- rmwindow(window);

  removes a specified window from it.

- rmwindows();

  removes all window linked to it.

- softcontrol(messagenothandled);
- hardcontrol();

  take control of program execution like MsgWindow does , but for multiple window application.

9) The **CScreen** class, which handles screen like CWindow does for windows. You can link as many window you want to a CScreen object so when a screen is closed, all its linked windows are closed. When it's reopened all windows that was opened reopen.
Non standard methods (no comment):

- linkwindow(window)
- rmwindow(window)
- rmwindows()
- openallwindows()
- closeallwindows()

10) The **GScreen** class, which inherits CScreen and CRastPortHdl is an encapsulation of a complete graphic screen.

11) The **WScreen** class, which inherits CScreen and Waiter. This screen control totally the program execution, the windows linked to it, ... everything.

12) The **WGScreen** class, which inherits WScreen and GScreen. A complete graphic, event handling screen.