By E.M. note sull'uso di GAMBAS

GAMBAS viene installato sotto Xubuntu nella directory:
/usr/share/gambas2

link
problema su uso dell'esempio SerialPort
http://gambasrad.org/zforum/view_topic?topic_id=1057
Doc sulle funzioni di Gambas relative alla Seriale
http://gambasdoc.org/help/comp/gb.net/serialport
Doc su Gambas
http://gambasdoc.org/help/

Uso delle SERIALI via USB

Prima di tutto si deve individuare il nome del device USB-Seriale e per fare questo si possono seguire i passi sotto:
1) **lsusb**
Visualizza i bus USB e le interfacce USB collegate al computer
2) **dmesg**
Visualizza i convertitori Seriali/USB con il loro nome logico (is used to examine or control the kernel ring buffer)

Individuato il device a cui ci si deve collegare, tipicamente **/dev/ttyUSB0** lo si deve rendere utilizzabile in scrittura e lettura per tutti gli utenti del computer e per fare ciò usate i comandi sotto (da terminale in Super Utente):
1) **cd /dev**
2) **chmod 777 ttyUSB0**
   (abbiamo supposto che il device si chiami ttyUSB0)

Aprite Gambas2 e caricate l'esempio SerialPort.
Lanciate SerialPort e se tutto funziona siete a posto ma se non vi funziona, il programma da errori o sembra bloccarsi, allora serve fare le modifiche sotto riportate.
Il progetto d'esempio (SerialPort) fornito da Gambas2 alla data di questo scritto, 16-11-08, non è funzionante per due motivi:

1) Tipicamente l'installazione di Gambas2 viene fatta in: ***/usr/share/gambas2*** ed è in sola lettura per cui bisogna andare nella directory:
/usr/share/gambas2/examples/Networking/SerialPort
e cambiare le proprietà di tutti i file presenti mettendoli in lettura e scrittura per tutti gli utenti (**chmod 777 nome-file**)

2) Un altro problema deriva dal fatto che viene caricato: **gb.gui** al posto del **gb.qt**
Per modificare l'impostazione (nell'installazione di Gambas2 Italiana) fate così:
- Selezionate PROGETTO e poi PROPRIETA'
- Dalla finestra che compare selezionate COMPONENTI disabilitate gd.gui e abilitate gd.qt
- In generale i componenti che devono essere abilitati sono riportati nella finestra sotto:

- Lanciate il programma e tutto deve funzionare

L'Intruzione **OPEN** di Gambas non si comporta nella stessa maniera che in Visual Basic. Non ritorna il file come un Integer, ma come un ogetto File.
In pratica, invece di digitare:

```
DIM handle AS Integer
...
OPEN "ilmiofile" FOR READ AS #handle
```

Devi digitare :

```
DIM handle AS File
...
handle = OPEN "ilmiofile" FOR READ
```

Puoi realizzare un file eseguibile di tutto il tuo progetto.
Seleziona Crea eseguibile nel menu Progetto.
Quando Gambas crea un file eseguibile, colloca il risultato direttamente nella directory del tuo progetto e il nome del file eseguibile sarà quello del tuo progetto

---

Non ci sono **variabili globali in Gambas**
Come sostituto, dichiarale nel modulo principale come PUBLIC. Se non hai un modulo principale nel tuo progetto, ma un form principale, allora dichiarale come STATIC PUBLIC.
Per accedere a queste variabili, devi usare il nome del modulo principale o form: MyMainModule.MyGlobalVariable o MyMainForm.MyGlobalVariable.

Per sapere se una stringa è vuota non è necessario usare la funzione Len() . Puoi direttamente testare la stringa, visto che una stringa vuota è FALSE e una non vuota è TRUE.
Per esempio, invece di :

```
IF Len(Lamiastringa) > 0 THEN ...
IF Len(Lamiastringa) = 0 THEN ...
```

Puoi fare :

```
IF Lamiastringa THEN ...
IF NOT Lamiastringa THEN ...
```

Tutti i controlli e tutti gli ogetti che possono avere eventi, hanno un Osservatore di eventi e un nome di gruppo del evento.
L'osservatore di eventi coglie tutti gli eventi prodotti dall'ogetto, e il nome del gruppo del evento è il prefisso del procedimento incaricato di gestire l'evento.
Di default, questo osservatore di eventi è l'ogetto dove si è creato il controllo, e il nome di gruppo è il nome del controllo.

In questo modo, un form riceve tutti gli eventi prodotti dai controlli che tu ci hai creato dentro.

```
' Gambas form

DIM hButton AS Button

PUBLIC SUB _new()
  ...
  hButton = NEW Button(ME) AS "Ilmiopulsante"
  ...
END

PUBLIC SUB Ilmiopulsante_Click()
  ...
END
```

La parola chiave **LAST** ritorna l'ultimo controllo che ha ricevuto un evento. E' molto utile quando vuoi scrivere un gestore di eventi che sia indipendente dal nome del controllo. Supponiamo di voler scrivere un programma calcolatrice. Hai definito dieci pulsanti, uno per ogni numero e tutti con lo stesso group "Digit". Il valore del Tag di ogni controllo sarà il numero visualizzato da ogni pulsante. Il tuo gestore di eventi sarà più o meno così :

```
PUBLIC SUB Digit_Click()

  Display = Display & LAST.Tag
  RefreshDisplay

END
```

Le famose routines **Left$, Right$, Mid$** di BASIC hanno un comportamento molto utile in Gambas.
Il secondo parametro di Left$ e Right$ è facoltativo, e per default è pari a uno.

```
Left$("Gambas") ritorna "G"
Right$("Gambas") ritorna "s"
```

Il secondo parametro può essere negativo e in questo caso rappresenta il numero di caratteri da non ritornare.

```
Left$("Gambas", -2) ritorna "Gamb"
Right$("Gambas", -2) ritorna "mbas"
```

Allo stesso modo, il terzo parametro di Mid$ può essere negativo, e quindi rapresentrare il numero di caratteri dalla fine della stringa da non ritornare.

```
Mid$("Gambas", 2, -2) ritorna "amb"
```

Gambas uses the **UTF-8 charset to represent strings in memory.**
But all standard Gambas string functions deal with ASCII:
Left, Mid, Right, UCase...
If you want to manipulate UTF-8 string, you must use the methods of the String static class, which have the same name as their standard counterparts.

```
PRINT Len("bébé");; Left$("bébé", 3)
6 bé
PRINT String.Len("bébé");; String.Left("bébé", 3)
4 béb
```

Error management in Gambas is done with the following instructions: **TRY, ERROR, CATCH, and FINALLY.**
TRY tries to execute a statement without raising an error. The ERROR instruction is used just after to know if the statement was executed correctly.

```
TRY MyFile = OPEN "/etc/password" FOR WRITE
IF ERROR THEN PRINT "I cannot do what I want!"
```

Error management in Gambas is done with the following instructions: TRY, ERROR, CATCH, and FINALLY.
CATCH indicates the beginning of the error management part of a function or procedure. It is put at the end of the function code.
The catch part is executed when an error is raised between the beginning of the function execution and its end.
If an error is raised during the execution of the catch part, it is normally propagated.

```
SUB ProcessFile(FileName AS STRING)
  ...
  OPEN FileName FOR READ AS #hFile
  ...
  CLOSE #hFile
CATCH ' Executed only if there is an error
  PRINT "Cannot process file "; FileName
END
```

Lo sapevi che puoi concatenare nomi di directory e nomi di file con l'operatore **&/**
Questo operatore aggiunge se necessario la slash ("/") in maniera tale che l'indirizzo risultante sia perfetto.
Per esempio :

```
PRINT "/home/gambas" &/ ".bashrc"
/home/gambas/.bashrc
PRINT "/home/gambas/" &/ "/tmp" &/ "foo.bar"
/home/gambas/tmp/foo.bar
```