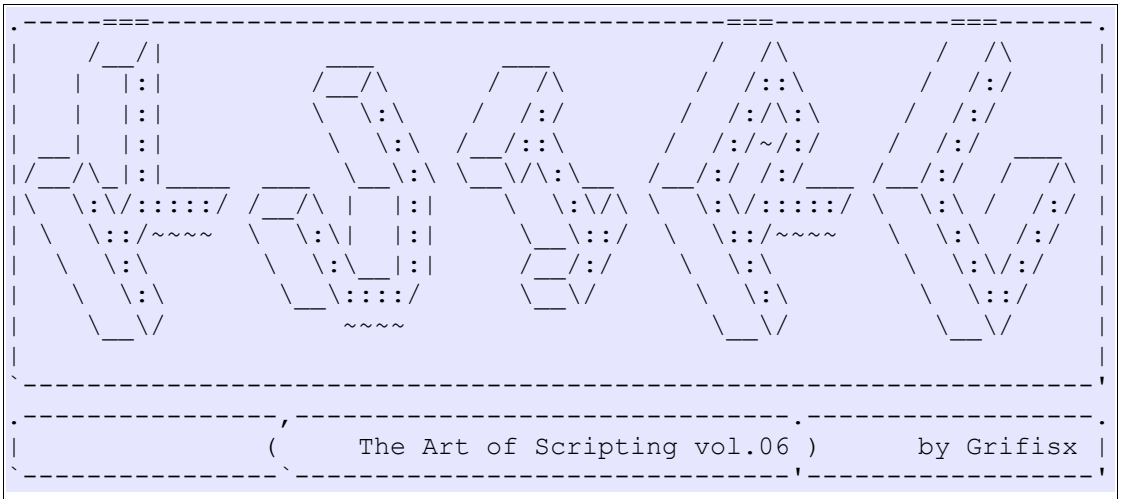

Oggetti, SLOTS e segnali.

Aggiornamento 20.07.2010
Start:

Eccoci arrivati al sesto volume di questa serie di tutorials nel quale ci addentreremo nell'uso delle **SLOT** e dei **segnali.**
Anche questa volta gli esempi serviranno solo a comprendere meglio i meccanismi senza avere un vero e proprio scopo pratico.
Abbiamo visto, nei volumi precedenti, come reimplementare alcune funzioni con il comando *privateimpl()*, in modo che, ad esempio, tramite il click su di un pulsante potesse essere eseguito un certo codice, adesso vedremo come connettere tra di loro gli oggetti, e fargli eseguire certe sequenze di comandi (o, se preferite, funzioni) senza utilizzare completamente il comando *privateimpl()* oppure utilizzandolo in modo "particolare".
Intanto dobbiamo chiarire cosa sono le *SLOTS* e cosa sono i *segnali:*
I segnali e le slots creano un sistema di comunicazione tra gli oggetti che creiamo.
I segnali sono emessi dagli oggetti in modo automatico (ma, come vedremo potremo anche farglieli emettere noi manuale tramite il comando "$emit()") quando avviene un cambiamento del proprio stato.
Ad esempio, quando andiamo a clickare su di un pulsante che abbiamo creato esso emette, in modo automatico, il segnale **clicked.**
Se andiamo a vedere la documentazione della classe **button** nell'Help del KVIrc Vediamo alla fine:

Signals

$clicked()
This signal is emitted by the default implementation of clickEvent(). If you reimplement that function you will have to emit the signal manually (if you still need it).

Pertanto, noi, in realtà, non ci accorgiamo che i seganli vengono emessi, loro partono e, se c'è qualcuno che li intercetta ed a cui sono collegati va bene, altrimenti non succede niente.
Adesso che sappiamo cosa sono i segnali, dobbiamo capire come sfruttarli.

Ogni segnale emesso da un oggetto può essere collegato ad una **SLOT**, la SLOT non è altro che una funzione di un altro oggetto o dello stesso oggetto che ha emesso il segnale (il nome "SLOT" invece del normale "funzione" è dato esclusivamente come convenzione ma in pratica funzioni e slot sono la stessa cosa).
Noi possiamo collegare più di una SLOT allo stesso segnale e, allo stesso tempo, possiamo collegare più segnali alla stessa SLOT, in questo modo un oggetto può notificare il cambiamento del proprio stato a più oggetti i quali si comporteranno di conseguenza.
Capisco che non è molto semplice come concetto, quindi, adesso, buttiamo giù un po' di sano codice =D.

creaiamo la classe esempio0

```
class(exemple0,object)
{
	segnale()
	{
		// creo la funzone segnale che emette il "biiip"
		@$emit(biiip)
	}
	constructor()
	{
		// connetto il segnale biiip alle slot
		objects.connect $$ biiip $$ slot1
		objects.connect $$ biiip $$ slot2
	}
	slot1()
	{
		echo Bip emesso, SLOT1
	}
	slot2()
	{
		echo Bip emesso, SLOT2
	}
}
```

e adesso proviamo la funzione segnale

```
%exp=$new(exemple0)
%exp->$segnale()
```

come vedrete all'emissione del segnale "biiip" è stato eseguito il codice contenuto nelle funzioni slot1() e slot2() -che avrebbero potuto chiamarsi in qualsiasi altro modo-.

Altro esempio

```
class(exemple0,object)
{
	constructor()
	{
		objects.connect $$ bip $$ slot1
		objects.connect $$ bip2 $$ slot2
	}
	slot1()
	{
```

```
            echo Bip emesso, SLOT1
      }
      slot2()
      {
            echo Bip emesso, SLOT2
      }
}
```

come vedete abbiamo tolto la funzione segnale, infatti useremo direttamente la funzione $emit(), dato che la funzione segnale non fa ceva altro che chiamare l'emit, ed abbiamo collegato il segnale "bip" alla slot1 ed il segnale "bip2" alla funzione slot2.
Proviamo:

```
%exp=$new(exemple0)
%exp->$emit(bip)
```

Creiamo adesso una widget con dentro almeno un pulsante per sfruttare il segnale *clicked*, che abbiamo visto sopra, e poi proviamo a far cambiare il colore della widget in base a quello che gli diamo noi ...cominciamo a buttare giù la prima parte del codice che realizza l'interfaccia, il codice è abbastanza grezzo e poco elegante, in seguito lo raffineremo:

```
// Es.1
// Creo la widget principale:
%Main_widget=$new(widget)
%Main_widget->$resize(300,60)
// Creo il layout per ordinare gli oggetti che andrò a mettere nella mia finestra e creo i vari oggetti
%main_layout=$new(layout,%Main_widget)
%Color_label=$new(label,%Main_widget)
// Setto il testo della label e sfrutto i tag html <b></b> per far apparire il testo in grassetto
%Color_label->$setText("<b>Cambia Colore di sfondo</b>")
%Color_lineedit=$new(lineedit,%Main_widget)
%Color_button=$new(button,%Main_widget)
%Color_button->$setText("Change")
%main_layout->$addMulticellwidget(%Color_label,0,0,0,0)
%main_layout->$addMulticellwidget(%Color_lineedit,1,1,0,0)
%main_layout->$addMulticellwidget(%Color_button,1,1,1,1)
// Mostro il tutto
%Main_widget->$show
```

Fino a qui tutto normale, adesso però dobbiamo andare a sfruttare il segnale **clicked** della classe **button** per far si che quando premiamo il pulsante *%Color_button* cambi il colore della widget principale.
Andiamo per passi:
per prima cosa vediamo il comando che serve per la connessione sengale-SLOT e questo è **objects.connect** la cui sintassi è

objects.connect <source_object> <signal_name> <target_object> <slot_name>

a questo punto dobbiamo creare una classe che, all'interno di se, abbia una funzione che ci permetta di cambiare lo sfondo della widget.
Niente di più facile, basta un semplice codice del genere :

```
class(mySLOT,object)
{
	changeWidgetBackGroundColor()
	{
		%Main_widget->$setBackgroundcolor(%Color_lineedit->$text())
		%Color_label->$setBackgroundcolor(%Color_lineedit->$text())
	}
}
```

cosi ho la mia classe, che ha la sua SLOT (o funzione come meglio preferite) **changeWidgetBackGroundColor** che si occupa di eseguire i comandi necessari a far cambiare il colore di sfondo della widget principale (ovviamente cambiamo anche quello della *label* altrimenti quest'ultima rimarrà del suo colore normale.
Quindi il codice verrà così trasformato:

```
//ES.2
// Creo la classe che deve contenere la funzione che utilizzerò come SLOT
class(mySLOT,object)
{
	changeWidgetBackGroundColor()
	{
		%Main_widget->$setBackgroundcolor(%Color_lineedit->$text())
		%Color_label->$setBackgroundcolor(%Color_lineedit->$text())
	}
}

%Main_widget=$new(widget)
%Main_widget->$resize(300,60)

%main_layout=$new(layout,%Main_widget)
// Gli oggetti li creo come variabili globali (lettera maiuscola) perche'
devono essere visibili anche dalla classe mySLOT che ho creato.
%Color_label=$new(label,%Main_widget)
%Color_label->$setText("<b>Cambia Colore di sfondo</b>")
%Color_lineedit=$new(lineedit,%Main_widget)
%Color_button=$new(button,%Main_widget)
%Color_button->$setText("Change")
%main_layout->$addMulticellwidget(%Color_label,0,0,0,10)
%main_layout->$addMulticellwidget(%Color_lineedit,1,1,0,1)
%main_layout->$addMulticellwidget(%Color_button,1,1,2,3)
// Creo l'oggetto '%MySlotObject' alla cui SLOT
'changeWidgetBackGroundColor' sarà collegato il segnale 'clicked' del mio
pulsante.
%MySlotObject=$new(mySLOT,%Main_widget)
// Connetto il segnale alla SLOT
objects.connect %Color_button clicked %MySlotObject changeWidgetBackGroundColor
%Main_widget->$show()
```

Eseguiamolo e proviamo a mettere nella lineedit "FFFFFF" e poi clickiamo sul pulsante per vederne il risultato.
Una piccola nota, avrete notato che ho creato l'oggetto **%MySlotObject** come figlio di **%Main_widget**, questo non è necessario, io l'ho fatto solo per ragioni di comodità, infatti dato che quando si distrugge un oggetto padre si distruggono automaticamente anche tutti gli oggetti figli, quando distruggerò l'oggetto *%Main_widget* automaticamente distruggerò anche l'oggetto *%MySlotObject* (che altrimenti avrei dovuto eliminare a parte con un delete per liberare la memoria da esso occupata).
Adesso però, siccome noi siamo tipi precisini, vogliamo poter settare il colore anche premendo il tasto "Invio" oltre che con il pulsante. Vediamo un po' che segnali ci sono nella classe lineedit:

Signals

$returnPressed()
This signal is emitted by the default implementation of returnPressedEvent().

$lostFocus()
This signal is emitted by the default implementation of lostFocusEvent().

$textChanged()
This signal is emitted by the default implementation of textChangedEvent().

returnPressed è proprio quello che cercavamo quindi possiamo aggiornare il nostro codice, connettendo il segnale della lineedit alla stessa slot che cambia il colore di sfondo (ecco l'utilità di poter connettere più segnali alla stessa SLOT).

```
//ES.3
class(mySLOT,object)
{
	changeWidgetBackGroundColor()
	{
		%Main_widget->$setBackgroundcolor(%Color_lineedit->$text())
		%Color_label->$setBackgroundcolor(%Color_lineedit->$text())
	}
}
%Main_widget=$new(widget)
%Main_widget->$resize(300,80)

%main_layout=$new(layout,%Main_widget)
%Color_label=$new(label,%Main_widget)
%Color_label->$setText("<b>Cambia Colore di sfondo</b>")
%Color_lineedit=$new(lineedit,%Main_widget)
/* Aggiungo questa funzione per dare un minimo di controllo sui dati inseriti anche se in realtà la maschera cosi inserita non è giustissima, infatti questa maschera permette di inserire anche lettere superiori alla 'F' (cosa non ammessa nei colori) vi consiglio di andare a vedere bene come  funziona il $setInputMask() della lineedit nella guida ufficiale di KVIrc
*/
%Color_lineedit->$setInputMask(">NNNNNN;0")
%Color_button=$new(button,%Main_widget)
%Color_button->$setText("Change")
%main_layout->$addMulticellwidget(%Color_label,0,0,0,10)
%main_layout->$addMulticellwidget(%Color_lineedit,1,1,0,1)
%main_layout->$addMulticellwidget(%Color_button,1,1,2,3)
%MySlotObject=$new(mySLOT,%Main_widget)
objects.connect          %Color_button          clicked          %MySlotObject
changeWidgetBackGroundColor
// Aggiungiamo il nuovo connect per legare il segnale returnpressed con la SLOT che cambia i colori
objects.connect       %Color_lineedit       returnpressed       %MySlotObject
changeWidgetBackGroundColor
%Main_widget->$show
```

potremmo anche voler esagerare e far si che il colore cambi (e si adatti) ogni volta che digitiamo il nuovo valore senza dover premere niente, in questo caso ci viene in aiuto l'altro segnale della *lineedit* il **textChanged**, che viene emesso dal nostro oggetto ogni volta che viene cambiato il testo ella lineedit.

Modifichiamo le ultime righe del nostro codice semplicemente così:

```
objects.connect %Color_button clicked %MySlotObject changeWidgetBackGroundColor
objects.connect %Color_lineedit returnpressed %MySlotObject
changeWidgetBackGroundColor
// Ecco il textChanged =)
objects.connect %Color_lineedit textchanged %MySlotObject changeWidgetBackGroundColor
```

Come vedrete provando il codice e digitando vari valori (provate a mettere "999999" e poi a cambiare i valori ad uno ad uno per avere risultati subito visibili) ogni volta che digitate un valore automaticamente la slot verrà richiamata e quindi varierà il colore dello sfondo.

```
//ES.4
class(mySLOT,object)
{
	changeWidgetBackGroundColor()
	{
		%Main_widget->$setBackgroundcolor(%Color_lineedit->$text())
		%Color_label->$setBackgroundcolor(%Color_lineedit->$text())
	}
// Aggiungo 2 funzioni, o SLOT come preferite, una per modificare l'altezza
// ed una per modificare la larghezza della widget principale, tramite il
// valore che potranno assumere gli 'slider' che creerò per modificare
questi valori.
	changeH()
	{
		%Main_widget->$resize(%Main_widget->$width(),%H_slider->$value())
	}
	changeW()
	{
		%Main_widget->$resize(%W_slider->$value(),%Main_widget->$height())
	}
}
%Main_widget=$new(widget)
%Main_widget->$resize(300,80)

%main_layout=$new(layout,%Main_widget)
%Color_label=$new(label,%Main_widget)
%Color_label->$setText("<b>Cambia Colore di sfondo</b>")
%Color_lineedit=$new(lineedit,%Main_widget)

%Color_lineedit->$setInputMask(">NNNNNN;0")
%Color_button=$new(button,%Main_widget)
%Color_button->$setText("Change")
// Creo 2 oggetti di tipo slider, ne setto l'orientamento e i valori massimi
// e minimi che potranno raggiungere. Li uso per modificare, in base al
// valore che cambierò spostando lo slider, l'altezza e la larghezza della
// widget principale.
%H_slider=$new(slider,%Main_widget)
%H_slider->$setOrientation(Vertical)
%H_slider->$setMinValue(80)
%H_slider->$setMaxValue(300)

%W_slider=$new(slider,%Main_widget)
%W_slider->$setOrientation(Horizontal)
%W_slider->$setMinValue(300)
%W_slider->$setMaxValue(400)

%main_layout->$addMulticellwidget(%Color_label,0,0,0,9)
%main_layout->$addMulticellwidget(%Color_lineedit,1,1,0,1)
%main_layout->$addMulticellwidget(%Color_button,1,1,2,3)
%main_layout->$addMulticellwidget(%H_slider,0,1,10,10)
%main_layout->$addMulticellwidget(%W_slider,2,2,0,9)
%MySlotObject=$new(mySLOT,%Main_widget)
```

```
objects.connect %Color_button clicked %MySlotObject changeWidgetBackGroundColor
objects.connect %Color_lineedit returnpressed %MySlotObject
changeWidgetBackGroundColor
objects.connect %Color_lineedit textchanged %MySlotObject changeWidgetBackGroundColor
// Collego il segnale  'valuechanged' degli slider alle 2 SLOT che ho creato
objects.connect %H_slider valuechanged  %MySlotObject changeH
objects.connect %W_slider valuechanged  %MySlotObject changeW
%Main_widget->$show()
```

Anche per questo codice non c'è niente di particolare da aggiungere, è solo un'ulteriore applicazione dei segnali, nella fattispecie abbiamo usato il segnale **valuechanged** dell'oggetto **slider.**

Ovviamente, come vi avevo detto, il superiore codice non lo si può certamente definire "elegante", ma è stato necessario proporlo sotto questa veste, per farvi capire che i collegamenti SLOT-Segnali possono avvenire anche tra oggetti diversi (Nel nostro caso tra gli oggetti "MainWidget" e figli e la classe "mySlot", per completezza vi propongo un codice più elegante che riunisce tutto in una classe unica, derivata dalla classe "hbox" ed utilizza un sistema di layout annidando vbox ed hbox, invece del multicelwidget.
Creiamo la classe exemple4, come al solito potete metterla nello script-tester ed eseguirla oppure (ve lo consiglio) utilizzare l'editor delle classi

```
class(exemple4,hbox)
{
	constructor()
	{
		@$resize(300,80)

		%left_vbox=$new(vbox,$$)
		@%color_label=$new(label,%left_vbox)
		@%color_label->$setText("<b>Cambia Colore di sfondo</b>")

		@%color_lineedit=$new(lineedit,%left_vbox)

		@%color_lineedit->$setInputMask(">NNNNNN;0")
		@%color_button=$new(button,%left_vbox)
		@%color_button->$setText("Change")

		@%h_slider=$new(slider,$$)
		@%h_slider->$setOrientation(Vertical)
		@%h_slider->$setMinValue(80)
		@%h_slider->$setMaxValue(300)

		@%w_slider=$new(slider,%left_vbox)
		@%w_slider->$setOrientation(Horizontal)
		@%w_slider->$setMinValue(300)
		@%w_slider->$setMaxValue(400)

		objects.connect	@%color_button	clicked	$$
changeWidgetBackGroundColor
		objects.connect	@%color_lineedit	returnpressed	$$
changeWidgetBackGroundColor
		objects.connect	@%color_lineedit	textchanged	$$
changeWidgetBackGroundColor
		objects.connect @%h_slider valuechanged  $$ changeH
		objects.connect @%w_slider valuechanged  $$ changeW
```

```
	}
	changeWidgetBackGroundColor()
	{
		@$setBackgroundcolor(@%color_lineedit->$text())
		@%color_label->$setBackgroundcolor(@%color_lineedit-
>$text())
	}
	changeH()
	{
		@$resize(@$width(),@%h_slider->$value())
	}
	changeW()
	{
		@$resize(@%w_slider->$value(),@$height())
	}
}
```

a questo punto creiamo l'oggetto e mostriamolo!

```
%expl4=$new(exemple4)
%expl4->$show()
```

L'eleganza del codice è importante, vi fa essere soddisfatti di quello che avete scritto =)
Adesso vi chiederete: Si possono disconnettere i segnali? (sicuramente non ve lo siete chieste ma io ve lo dico lo stesso =P)
Certo che si, per disconnettere (ad esempio potreste averne bisogno nel momento in cui delegate all'utente, tramite ad esempio una **checkbox**, la possibilità che un pulsante emetta un segnale (e quindi esegua una funzione) piuttosto che un altro) il segnale dalla SLOT a cui l'avevamo connesso.Il comando da usare è semplicemente **objects.disconnect**
con la sintassi:
objects.disconnect <source_object> <signal_name> <target_object> <slot_name>
Il suo uso ovviamente è chiaro, ad esempio se volessimo disconnettere il segnale **textchanged** dell'oggetto **%Color_lineedit** dalla SLOT **changeWidgetBackGroundColor**, dovremmo semplicemente scrivere:

```
objects.disconnect %Color_lineedit textchanged %MySlotObject
changeWidgetBackGroundColor
```

Facile no? =)

Ok, adesso tiriamo un bel respiro profondo e passiamo ad un altro modo di usare le SLOT =) ..cioè senza dover necessariamente creare una nuova classe e un nuovo oggetto e questo, come il grande "Noldor" insegna, semplicemente sfruttando la potenza del **privateimpl.**
Con il **privateimpl** come sappiamo possiamo reimplementare un evento o una funzione di un oggetto, bhè sappiate che possiamo anche implementarne di completamente nuove ..quindi facciamo subito una prova su due piedi:

```
// ES.5
// Creo un pulsante
%Btn=$new(button)
// Implemento una nuova funzione 'slotTesto' che apparterrà al mio
pulsante, e quindi si aggiungerà a quelle della classe button che il
mio pulsante ha già.
```

```
privateimpl(%Btn,slotTesto)
{
	%Btn->$setText("Funziona")
}
// Connetto il segnale clicked del mio pulsante ella funzione che ho implementato
objects.connect %Btn clicked %Btn slotTesto
%Btn->$show()
```

Interessante vero? Vediamo di cambiare l'esempio N.4 ed adattarlo a questo metodo diverso di usare le SLOT

```
// ES.5
%Main_widget=$new(widget)
%Main_widget->$resize(300,80)

%main_layout=$new(layout,%Main_widget)
%Color_label=$new(label,%Main_widget)
%Color_label->$setText("<b>Cambia Colore di sfondo</b>")
%Color_lineedit=$new(lineedit,%Main_widget)

%Color_lineedit->$setInputMask(">NNNNNN;0")
%Color_button=$new(button,%Main_widget)
%Color_button->$setText("Change")

%H_slider=$new(slider,%Main_widget)
%H_slider->$setOrientation(Vertical)
%H_slider->$setMinValue(80)
%H_slider->$setMaxValue(300)

%W_slider=$new(slider,%Main_widget)
%W_slider->$setOrientation(Horizontal)
%W_slider->$setMinValue(300)
%W_slider->$setMaxValue(400)

%main_layout->$addMulticellwidget(%Color_label,0,0,0,9)
%main_layout->$addMulticellwidget(%Color_lineedit,1,1,0,1)
%main_layout->$addMulticellwidget(%Color_button,1,1,2,3)
%main_layout->$addMulticellwidget(%H_slider,0,1,10,10)
%main_layout->$addMulticellwidget(%W_slider,2,2,0,9)
// Implemento,nell'oggetto '%Main_widget' le funzioni che userò come SLOTS tramite il privateImpl
privateImpl(%Main_widget,changeWidgetBackGroundColor)
{
	%Main_widget->$setBackgroundcolor(%Color_lineedit->$text())
	%Color_label->$setBackgroundcolor(%Color_lineedit->$text())
}
privateImpl(%Main_widget,changeH)
{
	%Main_widget->$resize(%Main_widget->$width(),%H_slider->$value())
}
privateImpl(%Main_widget,changeW)
{
	%Main_widget->$resize(%W_slider->$value(),%Main_widget->$height())
}
// Ovviamente devo cambiare l'oggetto di destinazione rispetto allo script precedente infatti ora le SLOT sono funzioni che appartengono all'oggetto %Main_widget e non funzioni di una classe e di un oggetto diverso.
objects.connect %Color_button clicked %Main_widget changeWidgetBackGroundColor
objects.connect %Color_lineedit returnpressed %Main_widget changeWidgetBackGroundColor
objects.connect %Color_lineedit textchanged %Main_widget changeWidgetBackGroundColor
objects.connect %H_slider valuechanged  %Main_widget changeH
objects.connect %W_slider valuechanged  %Main_widget changeW
```

```
%Main_widget->$show()
```

Come vedete il codice si semplifica e non ho il problema di dover creare una nuova classe e un nuovo oggetto, dato che le SLOT ora sono funzioni che appartengono al mio oggetto %Main_widget (si aggiungono insomma a quelle della classe widget che l'oggetto da me creato ha in quanto widget).
A questo punto ci sorge una domanda... ma i segnali possiamo anche crearceli noi? Ad esempio se volessi che un mio oggetto, quando viene eseguita una certa funzione emetta un segnale particolare, da me creato, che informi un altro oggetto che quella funzione è stata eseguita in modo che quest'ultimo si possa comportare di conseguenza, posso farlo? Certo che si =D!
Andiamo a vedere come possiamo far emettere, ai nostri oggetti, segnali particolari attraverso la funzione **$emit().**
Premessa: la funzione *$emit()* appartiene alla classe di base **object** e come tale può essere sfruttata in tutte le classi, la sintassi è abbastanza semplice, dalla documentazione ufficiale (alla voce *object class*):

```
$emit(<signal_name>[,parameters])
Emette il segnale <signal_name> passando i [parameters] opzionali.
```

Buttiamo giù qualche semplice riga di codice che ci faccia vedere subito gli effetti della funzione **$emit():**

```
// ES.6
%Widget=$new(widget)
%Widget->$resize(100,100)
// Mi creo la SLOT 'SLOTChangeColor' come funzione aggiuntiva dell'oggetto grafico %Widget
privateimpl(%Widget,SLOTChangeColor)
{
	%Widget->$setBackGroundColor($rand(9)$rand(9)9999)
}
// Creo un pulsante per chiudere la widget, lo faccio perché
// userò un timer e dovrò fermarne l'esecuzione tramite il comando 'killtimer'
%Btn=$new(button,%Widget)
%Btn->$setGeometry(75,60,40,40)
%Btn->$setText("Close")
// Connetto il segnale  changeColor (che mi sono inventato) alla SLOT
objects.connect %Widget changeColor %Widget SLOTChangeColor
// Reimplemento l'evento  'mousePressEvent' per fargli killare il timer quando premerò il pulsante
privateimpl(%Btn,mousePressEvent)
{
	delete	%Widget
	killtimer color
}
// Avvio un timer
timer(color,100)
{
// Genero tramite la funzione $emit l'emissione del mio segnale changeColor
	%Widget->$emit(changeColor)
}
%Widget->$show()
```

Vediamo un po' gli aspetti più interessanti del codice, cioè
objects.connect %Widget changeColor %Widget SLOTChangeColor
e
%Widget->$emit(changeColor)
come potete notare abbiamo collegato alla *SLOTChangeColor*, che abbiamo creato noi, un segnale che in realtà ancora non esiste, ovvero *changeColor* il segnale infatti prenderà vita nel momento in cui noi lo faremo emettere all'oggetto tramite la funzione *$emit()* e cesserà di esistere subito dopo, i segnali sono come degli impulsi che vengono emessi quando si verificano le condizioni dettate dal nostro codice.
Come potete vedere anche dalla sintassi della funzione $emit(), i segnali possono essere veicoli di parmametri, cioè tramite segnale posso passare dei parametri alla SLOT che li riceve, facciamo subito l'esempio modificando il codice dell'esempio n.6.

```
// ES.7
%Widget=$new(widget)
%Widget->$resize(100,100)
privateimpl(%Widget,SLOTChangeColor)
{
// Setto come colore il primo parametro che la SLOT riceve (sintatticamente parlando '$0')
      %Widget->$setBackGroundColor($0)
}
%Btn=$new(button,%Widget)
%Btn->$setGeometry(75,60,40,40)
%Btn->$setText("Close")
objects.connect %Widget changeColor %Widget SLOTChangeColor
privateimpl(%Btn,mousePressEvent)
{
      delete      %Widget
      killtimer color
}
timer(color,100)
{
// Creo una variabile che contenga il valore da assegnare al colore
      %colValue=$rand(9)$rand(9)9999
// Invio questa variabile come parametro con il segnale, il segnale diventerà il veicolatore di %colValue
      %Widget->$emit(changeColor,%colValue)
}
%Widget->$show()
```

Come potete notare in questo codice il valore l'abbiamo generato, all'interno del timer e abbiamo spedito il valore stesso tramite il segnale alla SLOT che si occupa di cambiare colore.
Questo è utilissimo, perché potremmo avere la necessità di far transitare dati attraverso i segnali come, ad esempio, il testo di una *lineedit* o lo stato di una *checkbox*, o il valore di uno *slider* etc etc.
Concludiamo con un altro piccolo esempio che potrebbe essere utile anche per capire come funziona il collegamento tra i menù a tendina e le SLOT, infatti uno dei loro principali usi è proprio legato ai menù popup:

```
class(exemple,hbox)
```

```
{
	constructor()
	{
		@$resize(300,300)
		@%svListView=$new(listview,$$)
		@%svListView->$addColumn($tr(Server),80)

		@%svPopupmenu=$new(popupmenu,@%svListView)
		@%svConnect=@%svPopupmenu->$insertItem("Connessione",59)
		@%svPopupmenuJoin=$new(popupmenu,@%svListView)
		@%svJoin=@%svPopupmenuJoin->$insertItem("Join",39)
		// Chiamiamo una funzione che ci riempie la listview, giusto per mettere degli elementi ho creato un processo automatizzato
		@$fillList()

		// Creiamo le connessioni
		objects.connect @%svPopupmenu activated $$ serverConnect
		objects.connect @%svPopupmenuJoin activated $$ chanJoin
		objects.connect @%svListView popuprequest $$ contextMenu

		//	Aggiungiamo un emit nell'evento customContextMenuRequestedEvent della nostra listview, facendogli emettere il segnale popuprequest (guardate la documentazione)
		privateimpl(@%svListView,customContextMenuRequestedEvent)
		{
			@$emit(popuprequest,$0,$1)
		}

	}
	contextMenu()
	{
		if (@%svListView->$currentItem())
		{
			// A seconda se l'item ha figli (server) o meno (canale) apro il popup appropriato
			if(@%svListView->$currentItem()->$childCount())
				@%svPopupmenu->$exec(@%svListView,$0,$1)
			else
				@%svPopupmenuJoin->$exec(@%svListView,$0,$1)

		}
	}
	serverConnect()
	{
		// Giusto per provare se funzionano i collegamenti
		echo Connessione Server
	}
	chanJoin()
	{
		// Giusto per provare se funzionano i collegamenti
		echo Connessione Canale
	}

	fillList()
	{
		// Funzioncina creata giusto per riempire la lista con padri e figli
		%servers[]=$array(irc.a.b,irc.c.d,irc.d.e)
```

```
            %channels[]=$array(chana,chanb,chanc,chane)
            foreach (%server,%servers[])
            {
                  %itemserver=$new(listviewitem,@%svListView)
                  %itemserver->$setText(0,"%server")
                  %itemserver->$setPixmap(0,14)
                  foreach (%chan,%channels[])
                  {
                        %item=$new(listviewitem,%itemserver)
                        %item->$setText(0,%chan)
                        %item->$setPixmap(0,31)
                  }

            }
      }
}

// Eseguiamo !

%expl=$new(exemple)
%expl->$show()
```

E con questo abbiamo, bene o male, completato l'argomento ...insomma possiamo crearci, in tutta tranquillità, la nostra autostrada di segnali.
Buona connessione e ...=D alla prossima!
/ECHO STOP

- - - - - - -- - - - - - - -- - - - - - - - -- - - - - -- - - - - -
" Tu vedi cose e ne spieghi il perché, io invece immagino cose che non sono mai esistite e mi chiedo perché no." (George Bernad Shaw)
- - - - - - - - -- - - - - - - -- -- - - - - - - - - - - - - - - -
Grifisx (Tonino Imbesi)