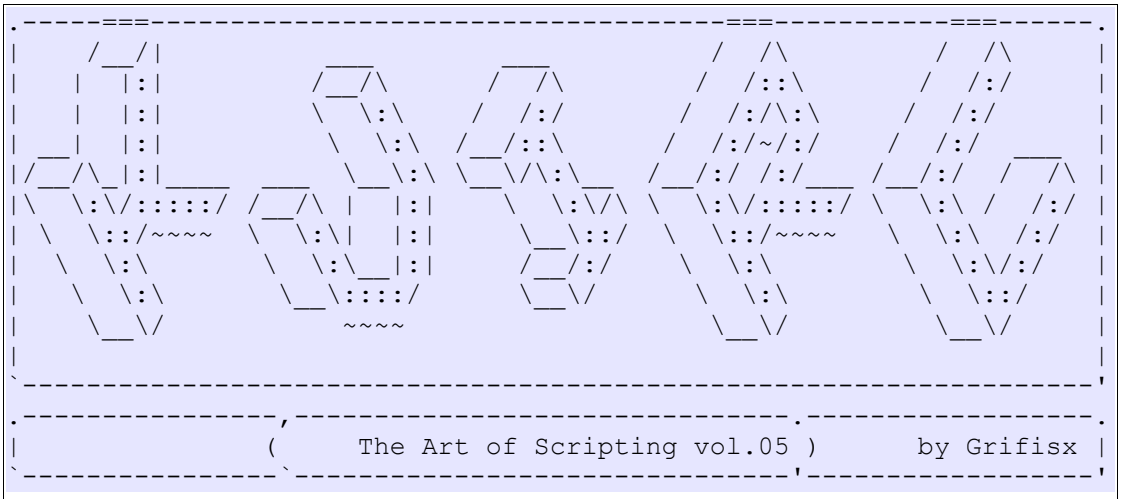


La classe **wrapper**

Start:

Questo tutorial sarà un po' più avanzato degli altri, spero che già abbiate acquisito un po' di esperienza con la grafica e la creazione degli oggetti perché adesso andremo a fare delle cose veramente incredibili e particolari.
Premetto che tutti gli esempi sono inutili e non hanno alcun'altra funzione se non quella di farvi prendere confidenza con la classe *wrapper* e con la funzione *$setProperty()*.
La classe wrapper è quella che amo di più, permette quasi di superare i limiti fisici che esistono tra programma compilato (il client KVIrc in se e per se) e fantasia e creatività dello scripter, definirla fantastica è riduttivo la definizione migliore che abbia sentito è: "questa classe è un hack" (Pragma) ...e presto capirete il perché.
Per prima cosa è necessario crearci un piccolo oggettino grafico, che poi utilizzeremo per i nostri esempi, ecco un paio di semplici righe di codice che ci serviranno dopo:

```
# ES:1
# Creo la widget principale
%mainWin=$new(widget)
%mainWin->$setCaption("Qt .:KVIrc:. ")
%mainWin->$setIcon(16)
%mainWin->$resize(200,200)
# Creo gli oggetti figli della widget principale
%mainWin_layout=$new(layout,%mainWin)
%button_close=    $new(button,%mainWin)
%textBox=$new(multilineedit,%mainWin)
# Setto le proprietà dei figli della widget principale
%button_close->$setText("Close")
# Metto i figli della widget principale in layout ricordando la sintassi:$addMultiCellWidget(widget>,<start_row>,<end_row>,<start_col>,<end_col>)
%mainWin_layout->$addmulticellwidget(%textBox,0,0,0,0)
%mainWin_layout->$addmulticellwidget(%button_close,1,1,0,0)
# Mostro l'oggetto grafico creato
%mainWin->$show()
```

Incolliamo tutto, come sempre, nello *script tester* e proviamone l'effetto.

Naturalmente non sto a spiegare il codice poiché, a parte che è commentato, a questo punto dei tutorial dovrebbe essere chiaro =).
Insisto su una cosa, **consultate sempre l'help-guida del KVIrc** perché troverete tutte le funzioni, comandi e classi da utilizzare.
Adesso torniamo alla nostra classe *wrapper* per vedere a cosa serve e cosa è di preciso.
E' una classe che permette di "agganciarsi" a tutti gli elementi grafici del KVIrc dando la possibilità di cambiare l'aspetto di tutto l'ambiente grafico del nostro client, modificandone le proprietà, aggiungendo altre widget come figlie di quelle proprie del KVIrc e altre cosette interessanti; in poche parole ci permetterà di superare i limiti tra script e codice c++ compilato, dando la possibilità agli scripters di aggiungere, ad esempio, un pulsante nello sfondo di una finestra di query, o di far comparire la calcolatrice che ci siamo fatti in scripting attaccata alle finestre oppure allo sfondo del KVIrc, piazzare pulsanti sulle barre del nostro client(come la status bar ad esempio) ed insomma dar vita a qualsiasi "mutazione genetica" che la nostra mente malata possa partorire.
Partiamo da un concetto di base: quando noi costruiamo un oggetto grafico (l'interfaccia di un nostro script ad esempio)lo possiamo rappresentare come un albero i cui rami seguono i "rapporti di parentela" "oggetto padre->oggetto figlio", ad esempio, nel codice di prima abbiamo creato un albero di questo tipo:

```
%mainWin                (padre)
      -%mainWin_layout  (figlio di %mainWin)
      -%button_close    (figlio di %mainWin)
      -%textBox         (figlio di %mainWin)
```

Adesso, per capire meglio, cambiamo il codice di prima in quello che segue cosi avremo un idea migliore (e più pratica)del concetto di albero e di parentela tra le widget delle nostre interfacce grafiche create in scripting.

```
# ES: 2
%mainWin=$new(widget)
%mainWin->$setCaption("Qt .:KVIrc:. ")
%mainWin->$setIcon(16)
%mainWin->$resize(200,200)
# Creo la widget che ospiterà solo le caselle di testo figlia di quella principale
%sonWidget1=$new(widget,%mainWin)
# Creo la widget che ospiterà solo i pulsanti figlia di quella principale
%sonWidget2=$new(widget,%mainWin)
# Creo i 3 layout per le 3 widget che ho creato, la principale e le sue due figlie
%mainWin_layout=$new(layout,%mainWin)
%sonWidget1_layout=$new(layout,%sonWidget1)
%sonWidget2_layout=$new(layout,%sonWidget2)
# Creo i pulsanti, figli della widget che deve "ospitare" solo pulsanti
%button_close=    $new(button,%sonWidget2)
%button_open=     $new(button,%sonWidget2)
# Creo le caselle di testo, figlie della widget che deve "ospitare" solo  caselle di testo
%textBox=$new(multilineedit,%sonWidget1)
%lineedit=$new(lineedit,%sonWidget1)
# Setto il testo dei pulsanti
```

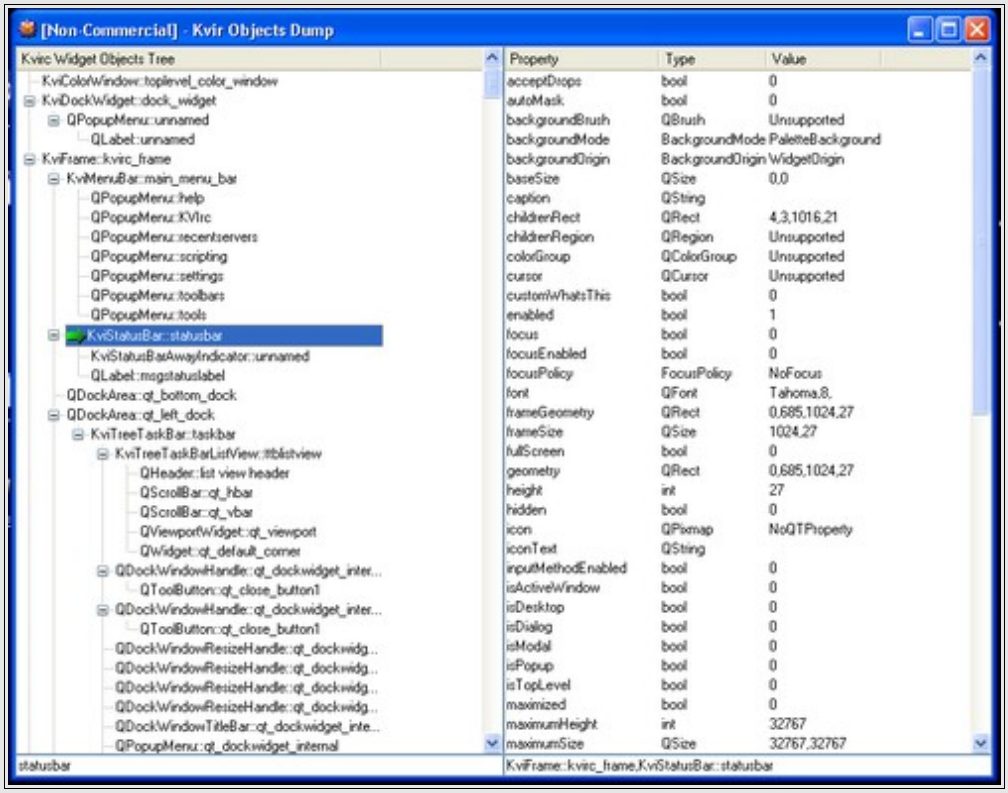
```
%button_close->$setText("Close")
%button_open->$setText("Open")
# Inserisco gli oggetti nei loro rispettivi layout:
# prima le caselle di testo:
%sonWidget1_layout->$addmulticellwidget(%textbox,0,0,0,0)
%sonWidget1_layout->$addmulticellwidget(%lineedit,1,1,0,0)
# dopo i pulsanti:
%sonWidget2_layout->$addmulticellwidget(%button_close,0,0,0,0)
%sonWidget2_layout->$addmulticellwidget(%button_open,1,1,0,0)
# ed infine mettiamo nel layout della widget principale
# i due oggetti grafici che ospitano pulsanti e caselle di testo:
%mainWin_layout->$addmulticellwidget(%sonWidget1,0,0,0,0)
%mainWin_layout->$addmulticellwidget(%sonWidget2,1,1,0,0)
# Mostriamo il risultato
%mainWin->$show()
```

Mettiamo tutto nello *script tester* e proviamone il risultato.
Adesso esaminiamo l'albero del nostro nuovo codice:

```
%mainWin                (padre)
      -%mainWin_layout  (figlio di %mainWin)
      -%sonWidget1      (figlio di %mainWin)
            -%sonWidget1_layout     (figlio di %sonWidget1)
            -%textbox               (figlio di %sonWidget1)
            -%lineedit              (figlio di %sonWidget1)
      -%sonWidget2      (figlio di %mainWin)
            -%sonWidget2_layout     (figlio di %sonWidget2)
            -%button_close          (figlio di %sonWidget2)
            -%button_open           (figlio di %sonWidget2)
```

Vediamo che, ad esempio, *%button_open* è figlio di *%sonWidget2* che a sua volta è figlio di *%mainWin*, quindi per risalendo "l'albero genealogico" avremo:
%button_open->%sonWidget2->%mainWin.
Questo succede non soltanto nei nostri script ma anche in tutti gli oggetti grafici del KVIrc, infatti anche loro sono creati (a livello di codice del client) come alberi, ad esempio la barra di stato, a livello di codice, è figlia della finestra (frame)principale di del KVIrc ed ha come "percorso" dell' "albero di parentela":
"**KviFrame::kvirc_frame->KviStatusBar::statusbar**"
(non fatevi impressionare da questo, perché alla fine i percorsi degli oggetti del KVIrc li ricaviamo in modo automatico, quindi niente paura)
Ora, per poterci agganciare ad un oggetto di KVIrc e poter fare i nostri esperimenti, dobbiamo avere il "percorso delle parentele" di questo al fine di raggiungerlo ed agganciarlo.
Per semplificare il modo di trovare le parentele io ed il mio amico Noldor, abbiamo creato uno script che ci crea proprio graficamente l'albero di tutte le parentele degli oggetti agganciabili del KVIrc.
Lo script si chiama **dump.kvs** è un semplice alias, quindi scarichiamolo da qui:
https://cvs.kvirc.de/kvirc/cvsweb/scripts/
Nella cartella objectsdumptree oppure usiamo quello che ho allegato al tutorial.
Una volta che lo abbiamo a disposizione installiamolo da:
Scripting->Esegui script
Una volta fatto ciò lanciamo il comando **/dump** e vedremo apparire una finestrona che sulla sinistra ci mostra tutto l'albero delle widget

del KVIrc, sulla destra le loro proprietà, sulla destra in basso, quando selezioniamo un oggetto, il percorso completo dell'albero per arrivare ad esso(ed è quello che volevamo) mentre sulla sinistra in basso abbiamo una lineedit per effettuare ricerche all'interno dell'albero.
Proviamo, ad esempio, a trovare il percorso della status-bar; assicuriamoci che sia visibile e poi lanciamo il **/dump**.
Nella lineedit di ricerca (quella in basso a sinistra) scriviamo *statusbar* e poi diamo invio.
Risultato:



Percorso ^

Come vedete in basso a destra ci appare il percorso che ci interessa per agganciarci alla status bar.
Adesso che sappiamo come raggiungere gli oggetti di KVIrc dobbiamo vedere un po' come si fa ad agganciarsi ad essi =).
Per prima cosa dobbiamo creare un oggetto di tipo *wrapper* figlio della nostra status-bar (cosa che, avendo il percorso bello e pronto non è difficile da fare, dobbiamo solo copia-incollarlo dal risultato del dump)

```
%Prova=$new(wrapper,0,prova,KviFrame::kvirc_frame,KviStatusBar::statusbar)
```

A questo punto abbiamo agganciato la status bar tramite l'oggetto %Prova, quindi possiamo cominciare a farci i nostri primi esperimenti:

```
%Prova=$new(wrapper,0,test,KviFrame::kvirc_frame,KviStatusBar::statusbar)
```

```
# Sfondo bianco
%Prova->$setBackGroundColor(FFFFFF)
# Scritte verdi
%Prova->$setForeGroundColor(008800)
```

Mettiamolo nello script tester, come al solito, e poi eseguiamo
Noterete i cambiamenti della status bar =), ovviamente le funzioni $setBackGroundColor() e $setForeGroundColor() sono le funzioni della nostra classe base la classe "widget" da cui, come sappiamo, tutte quante derivano, e quindi anche il nostro wrapper.
Questo è solo l'inizio, proviamo ad eseguire questo codice:

```
# Oggetto wrapper
%Prova=$new(wrapper,0,test,KviFrame::kvirc_frame,KviStatusBar::status
bar)
# Aggiungiamo un pulsante
%Button=$new(button,%Prova)
%Button->$setText("Prova")
# Mostriamo
%Button->$show()
```

Come vedrete appena eseguirete questo scriptino sulla status bar apparirà un pulsante.
E se volessimo "wrappare" la label della staus-bar (dove appare il server e il nick e i modi tipo **[irc.azzurra.org]Grifisx(+ix)**)e farla diventare bianca?
Niente di più semplice: lanciamo il nostro **/dump**, cerchiamo *statusbar* come prima, e poi cerchiamo la label figlia di questa, notiamo che l'albero appare più o meno così:

```
.
.
KviFrame::kvirc_frame
      .
      .
      KviStatusBar::statusbar
            .
            KviStatusBarAwayIndicator::unnamed
            Qlabel::msgstatuslabel  <-- Ecco quello che cerchiamo!
```

Quindi evidenziamolo e copiamoci il percorso per creare un oggetto che lo "wrappi":
Il percorso è:
KviFrame::kvirc_frame,KviStatusBar::statusbar,QLabel::msgstatuslabel

Quindi agganciamoci all'oggetto e settiamo il colore di sfondo bianco:

```
%Prova=$new(wrapper,0,test,KviFrame::kvirc_frame,KviStatusBar::status
bar,QLabel::msgstatuslabel)
%Prova->$setBackGroundColor(FFFFFF)
```

Eseguiamo e godiamo dei risultati =D.
Adesso la status bar possiamo farla diventare tutta bianca volendo, in questo modo:

```
%Prova=$new(wrapper,0,test,KviFrame::kvirc_frame,KviStatusBar::status
bar,QLabel::msgstatuslabel)
```

```
%Prova->$setBackGroundColor(FFFFFF)

%Prova2=$new(wrapper,0,test2,KviFrame::kvirc_frame,KviStatusBar::stat
usbar)
%Prova2->$setBackGroundColor(FFFFFF)
```

Oppure tutta nera con le scritte verde limone :

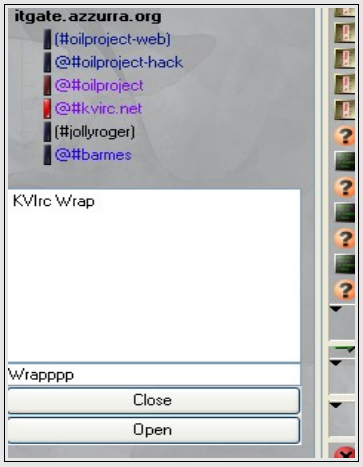
```
%Prova=$new(wrapper,0,test,KviFrame::kvirc_frame,KviStatusBar::status
bar,QLabel::msgstatuslabel)
%Prova->$setBackGroundColor(000000)
%Prova2=$new(wrapper,0,test,KviFrame::kvirc_frame,KviStatusBar::statu
sbar)
%Prova2->$setBackGroundColor(000000)
%Prova2->$setForeGroundColor(33FF11)
```

...ora dovreste cominciare a capire la potenza della classe: modificare quasi tutto quello che vogliamo e come più ci piace.
Supponiamo che volessimo far apparire ls finestrella grafica dell'Esempio N.2 di questo tutorial, nella lista ad albero dei canali.
Per prima cosa lanciamo il **/dump** per trovare il "percorso" della nostra lista dei canali ad albero, proviamo a fare una ricerca ad esempio con la parola "Tree" e vediamo se esce fuori qualche cosa, altrimenti, con pazienza, dovremo sfogliarci l'albero.
La ricerca, in questo caso, ha prodotto i suoi effetti e troviamo il percorso:
KviFrame::kvirc_frame,QDockArea::qt_left_dock,KviTreeTaskBar::taskbar,KviTreeTaskBarListView::ttblistview
Attenzione, non sarete sempre così fortunati ed a volte la ricerca dovrete farla a mano, con molta pazienza e con molte prove prima di trovare l'oggetto giusto.
Proviamo a sfruttare la nostra classe wrapper:

```
# Creo l'oggetto per agganciarmi alla lista dei canali ad albero
%WrapTree=$new(wrapper,0,kvirctreelistview,KviFrame::kvirc_frame,QDoc
kArea::qt_left_dock,KviTreeTaskBar::taskbar,KviTreeTaskBarListView::t
tblistview)
# Ovviamente %MainWin la creerò figlia del mio oggetto wrapper
# Uso variabili maiuscole (globali), per poi avere la possibilità di
# eliminarle con delete, se le facessi non globali le perderei dopo
# averle create e non potrei più deletarle.
%MainWin=$new(widget,%WrapTree)
%MainWin->$setCaption("Qt .:KVIrc:. ")
%MainWin->$setIcon(16)
%MainWin->$setGeometry(0,300,200,200)

%sonWidget1=$new(widget,%MainWin)

%sonWidget2=$new(widget,%MainWin)

%mainWin_layout=$new(layout,%MainWin)
%sonWidget1_layout=$new(layout,%sonWidget1)
%sonWidget2_layout=$new(layout,%sonWidget2)

%button_close=    $new(button,%sonWidget2)
%button_open=     $new(button,%sonWidget2)
```

```
%textBox=$new(multilineedit,%sonWidget1)
%lineedit=$new(lineedit,%sonWidget1)

%button_close->$setText("Close")
%button_open->$setText("Open")
# Aggiungo,al pulsante "Close", la funzione per chiudere la widget
#  creata eliminando gli oggetti principali
privateImpl(%button_close,mousePressEvent)
{
      delete %MainWin
      delete %WrapTree
}
%sonWidget1_layout->$addmulticellwidget(%textbox,0,0,0,0)
%sonWidget1_layout->$addmulticellwidget(%lineedit,1,1,0,0)

%sonWidget2_layout->$addmulticellwidget(%button_close,0,0,0,0)
%sonWidget2_layout->$addmulticellwidget(%button_open,1,1,0,0)

%mainWin_layout->$addmulticellwidget(%sonWidget1,0,0,0,0)
%mainWin_layout->$addmulticellwidget(%sonWidget2,1,1,0,0)

%MainWin->$show()
```

Eseguiamo nello script tester e vediamo il risultato.



Lista agganciata con successo ...direi che è proprio quello che volevamo =).
Ammettiamo invece che volessimo crearlo proprio attaccato allo sfondo del KVIrc, cerchiamo con il solito metodo la widget dello sfondo del KVIrc e questa volta la ricerca non ci può aiutare molto, comunque il percorso è questo:
KviFrame::kvirc_frame,QSplitter::main_splitter,KviMdiManager::mdi_manager,QViewportWidget::qt_viewport
adattiamo il codice di prima al nuovo percorso, modificando la riga:

```
%WrapTree=$new(wrapper,0,test,KviFrame::kvirc_frame,QSplitter::main_splitter,KviMdiManager::mdi_manager,QViewportWidget::qt_viewport)
```

ed eseguiamo.
Appena eseguiremo, vedremo che abbiamo agganciato la nostra widget allo sfondo del KVIrc.
Ottimo per adesso, ma ora voglio farvi conoscere alcuni oggetti molto

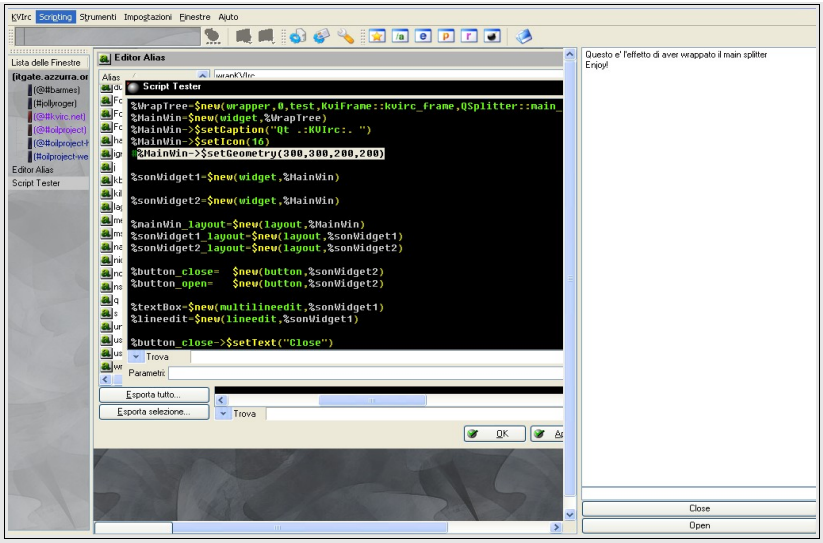
utili da wrappare, cioè gli **splitter**, questi (in poche parole e senza scendere nel troppo tecnico) sono oggetti che hanno il compito di dividere una finestra in 2 parti.
Bene, vediamo subito degli esempi e gli effetti visivi.
Modifichiamo il codice di prima in questo modo:

```
%WrapTree=$new(wrapper,0,test,KviFrame::kvirc_frame,QSplitter::main_splitter)
```

e commentiamo la riga

```
%MainWin->$setGeometry(300,300,200,200)
```

Eseguiamo ed ecco il risultato:



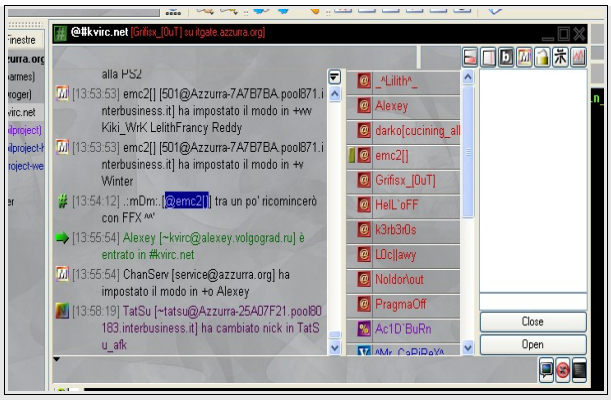
Come potete vedere è apparso alla destra della finestra principale una specie di divisore, che contiene la nostra widget.
Proviamo a fare la stessa cosa con la finestra del canale **#kvirc.net** ad esempio.

-percorso:
KviFrame::kvirc_frame,QSplitter::main_splitter,KviMdiManager::mdi_manager,QViewportWidget::qt_viewport,KviMdiChild::mdi_child_#kvirc.net,KviChannel::#kvirc.net,QSplitter::main_splitter

-modifica al nostro codice di esempio:

```
%WrapTree=$new(wrapper,0,test,KviFrame::kvirc_frame,QSplitter::main_splitter,KviMdiManager::mdi_manager,QViewportWidget::qt_viewport,KviMdiChild::mdi_child_#kvirc.net,KviChannel::#kvirc.net,QSplitter::main_splitter)
```

-Risultato:



La nostra widget è apparsa agganciata nella finestra del canale, accanto alla lista degli utenti, bello vero?
Altro piccolo esempio per chiudere questo argomento e poi passiamo all'uso di una funzione per utenti più avanzati.
Vogliamo creare un pulsante ed una lineedit accanto ai pulsantini che appaiono sopra la lista degli utenti dei canali, stessa procedura con **/dump** e andiamo ad individuare il percorso, che nel caso del canale *#kvirc.net* è il seguente:
KviFrame::kvirc_frame,QSplitter::main_splitter,KviMdiManager::mdi_manager,QViewportWidget::qt_viewport,KviMdiChild::mdi_child_#kvirc.net,KviChannel::#kvirc.net,QHBox::button_box
(Notate che mi inserisco nel QHBox, che per chi sa usare le librerie QT corrisponde ad una forma di layout)
Proviamo, quindi, questo codice:

```
%Wrap=$new(wrapper,0,test,KviFrame::kvirc_frame,QSplitter::main_splitter,KviMdiManager::mdi_manager,QViewportWidget::qt_viewport,KviMdiChild::mdi_child_#kvirc.net,KviChannel::#kvirc.net,QHBox::button_box)
%Button=$new(button,%Wrap)
%Button->$settext("New")
%Text=$new(lineedit,%Wrap)
%Button->$show()
%Text->$show()
```

Risultato:

Come potete notare ecco le nostre due widget, ben allineate in alto.
Tutto questo vi farà sicuramente capire quanto sia potente questa misteriosa classe.
Ora una piccola chicca per coloro che sanno utilizzare le librerie QT o per gli utenti ancora più smaliziati e desiderosi di oltrepassare i limiti (a questo punto mi chiederei ...quali ^_^?).

$setProperty()

Quando noi creiamo un oggetto grafico, come ad esempio una widget, sappiamo di potergli attribuire delle proprietà tramite le varie funzioni di quella classe come la caption (con un $setCaption()), il colore di sfondo ($setBackGroundColor()) l'icona etc. etc.
Anche gli oggetti del KVIrc hanno le loro proprietà, non dimentichiamo che il linguaggio di scripting del KVIrc è basato sul c++ e sulle librerie QT, quindi anche tutti quegli oggetti che vediamo, pulsanti, finestre, caselle di testo, liste ad albero, e che fanno parte del client in se e per se, hanno delle loro proprietà, ad esempio la finestra di un canale avrà il suo backgroundcolor, la sua caption la sua icon la sua geometry etc. etc. e tutti questi valori sono settati a monte, cioè nel codice del KVIrc e se alcuni di essi, tramite le opzioni, possono essere cambiati altri invece bisognerebbe modificare il codice del programma per cambiarli.
Ebbene, vi rivelo un segreto ..le proprietà degli oggetti del KVIrc possono essere variate a seconda dei nostri desideri tramite la funzione
$setProperty(<proprietà>,<valore>)
(magico vero?)
Provate ad esempio:

```
%Prova=$new(wrapper,0,test,KviFrame::kvirc_frame,QToolButton::kvirc.identityoptions)
%Prova->$setProperty(usesBigPixmap,0)
```

oppure:

```
%Prova->$setProperty(usesBigPixmap,1)
```

E vedrete cambiare la proprietà **usesBigPixmap**, della toolbar di KVIrc, come per magia.
Per usare al meglio questa funzione, in teoria, dovreste saper usare le librerie QT (in modo da conoscere le proprietà che un determinato tipo di oggetto grafico possa aver settate) ma in ogni caso, con il nostro **/dump** ricaviamo anche le proprietà attualmente settate sugli oggetti e possiamo, oltre che conoscere quali esse siano senza dover imparare ad usare le QT, modificarle al volo cliccando una volta su di esse e cambiandole manualmente.
Se volessimo sapere quali sono le proprietà della finestra di un canale, ad esempio, dovremmo semplicemente portarci su di essa seguendo il nostro albero del dump:
Intanto il percorso sarà:
KviFrame::kvirc_frame,QSplitter::main_splitter,KviMdiManager::mdi_manager,QViewportWidget::qt_viewport,KviMdiChild::mdi_child_#kvirc.net,KviChannel::#kvirc.net
Mettiamoci sull'oggetto "**KviChannel::#kvirc.net**" e vediamo che proprietà appaiono nella lista a destra:

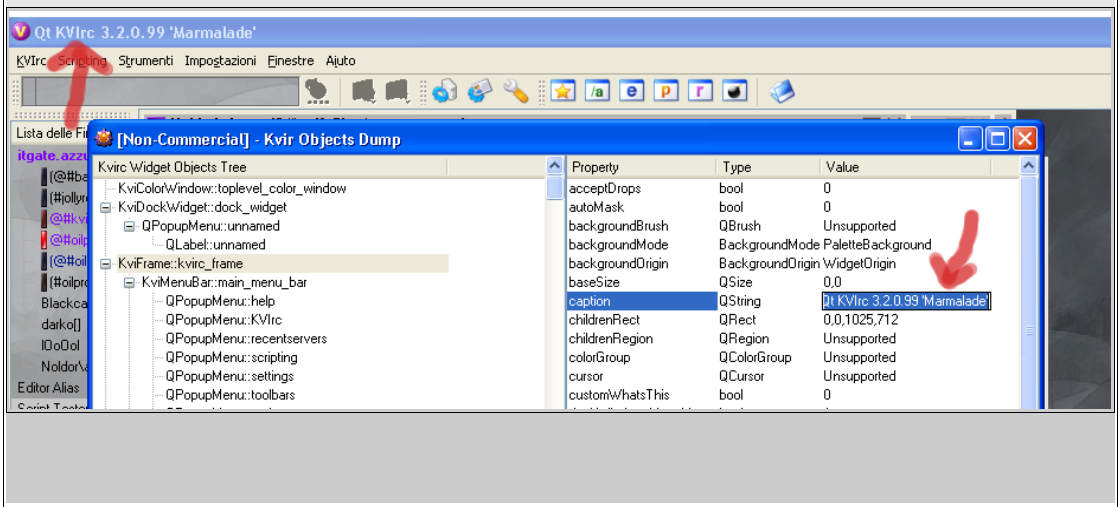
Property	Type	Value
maximumSize	QSize	32767 ,32767
maximumWidth	int	32767
microFocusHint	QRect	325 0 1,341
minimized	bool	0
minimumHeight	int	100
minimumSize	QSize	100 100
minimumSizeHint	QSize	-1,1
minimumWidth	int	100
mouseTracking	bool	0
name	QCString	#kvirc.net

```
ownCursor                       bool            0

ownFont                         bool            0
ownPalette                      bool            0
palette                         QPalette        Unsupported
paletteBackgroundColor          QColor          ece 9d8
paletteBackgroundPixmap         QPixmap         Unsupported

paletteForegroundColor          QColor          000000
pos                             QPoint          4,26
rect                            QRect           0,0,651,341
shown                           bool            1

.
.[omissis]
.

x                               int             4
y                               int             26
```

Benissimo, abbiamo la lista delle proprietà che possiamo modificare, il tipo di dato che esse vogliono (Intero, Booleano, Stringa, Colore Rect (4 interi come coordinate di un rettangolo)) e i valori che queste proprietà hanno attualmente.
Andiamo ad agire contro le leggi fisiche e proviamo a modificarle, proviamo ad esempio a modificare la grandezza massima che la finestra del canale possa raggiungere:

```
%Wrap=$new(wrapper,0,test,KviFrame::kvirc_frame,QSplitter::main_splitter,KviMdiManager::mdi_manager,QViewportWidget::qt_viewport,KviMdiChild::mdi_child_#kvirc.net,KviChannel::#kvirc.net)
%Wrap->$setProperty(maximumSize,30,30)
```

Come per magia la vedrete accartocciarsi su se stessa fino a raggiungere una dimensione di 30,30 (mi sembra di essere uno scienziato pazzo che fa esperimenti o.o).
Faccio un esempio modificando la proprietà **caption** della finestra principale di KVIrc, prima tramite il /dump e poi tramite 2 righe di script



Come potete vedere cambiando il valore della proprietà **caption** in

“Qt KVIrc etc. etc.” subito si riscontra l'effetto.
Facciamo lo stesso tramite script:

```
%Wrap=$new(wrapper,0,test,KviFrame::kvirc_frame)
%Wrap->$setProperty(caption,"Qt Il Wrap roXXa")
```

Mettiamolo nello script tester e poi eseguiamo per vederne gli effetti.
Come ho premesso tutti gli esempi fatti sono abbastanza inutili e servono solo a farvi prendere confidenza con il misterioso e potente wrapper. Sta alla vostra fantasia e creatività sfruttare al massimo questa classe, gli script con essa diventano non solo mezzi per costruire qualche cosa di nuovo e fonderlo fisicamente con l'interfaccia di KVIrc, ma anche per modificare ciò che già esiste.
Devo pero' ridimensionare la vostra (e la mia) smania di onnipotenza, poiché alcune delle proprietà sono a sola lettura e quindi non possono essere modificate, come fare a capire quali sono a sola lettura e quali no?
Dobbiamo ricorrere alla funzione **$listProperty()**, vediamo un po' come fare:

```
%Wrap=$new(wrapper,0,test,KviFrame::kvirc_frame,QSplitter::main_splitter,KviMdiManager::mdi_manager,QViewportWidget::qt_viewport,KviMdiChild::mdi_child_#kvirc.net,KviChannel::#kvirc.net)
%Wrap->$listProperty()
```

Eseguiamolo.
Nella finestra attuale ricevere la lista delle proprietà dell'oggetto (io ho messo la finestra del canale #KVIrc.net)
Ecco una parte di tutte quelle che mi sono apparse a me:

```
[12:37:36] Elenco Proprietà Qt per l'oggetto widget test (3283.1130931456)
[12:37:36] Proprietà classe: KviChannel
[12:37:36] Proprietà classe: KviWindow
[12:37:36] Proprietà: KviProperty_ChildFocusOwner, tipo: int
[12:37:36] Proprietà classe: QWidget
[12:37:36] Proprietà: isTopLevel, tipo: bool
.
.
.
[12:37:36] Proprietà: enabled, tipo: bool, scrivibile
[12:37:36] Proprietà: geometry, tipo: QRect, scrivibile
```

ecco come potete vedere le proprietà che possiamo cambiare sono quelle accanto alle quali appare “**scrivibile**”, le altre, purtroppo, non subiranno l'effetto delle nostre modifiche.
Concludo informandovi che, naturalmente, la lista delle proprietà potete ottenerla, anche dai vostri oggetti, non soltanto da quelli di KVIrc. Provate ad esempio il seguente codice:

```
%Btn=$new(button)
%Btn->$listProperty()
%Btn->$show()
```

Eseguite e vedrete tutta la serie di proprietà modificabili del pulsante che avete creato.

E cosi abbiamo concluso anche questa puntata, buon wrapping =D!.
/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

Allegato:

```
alias(dump)
{
	# Objects Tree Dump  script
	# Written by Noldor & Grifisx  (2005)
	# This script allow you to see all kvirc's interface objects in a listview.
	# It's just an alias, install it and "/dump"  =) EnjoY!

	class (mylistview,listview)
	{
		constructor
		{
		@%list_name[]= ""
		}
		selectionChangedEvent
		{
			%tmp=$0
			%i=0
			while(%tmp->$classname != "mylistview")
			{
				%temp_object[%i]=%tmp->$text(0);
				%i++;
				%tmp=%tmp->$parent();
			}
			@$killPropertyChildren()
			%i=%temp_object[]#
			while(%i)
			{
			 %gerarchia=%gerarchia%temp_object[$(%i-1)]
			if (%i !=1) %gerarchia=%gerarchia"->"
			%i --;
			}
			%tmp_property=$new(listviewitem,$$->$parent->%property)
			$$->$parent->%property->$setTooltip(%gerarchia)
			%ger=$str.replace(%gerarchia,",","->")
			eval \%Wrapper\=\$new\(wrapper,0,test,%ger\)
			$$->$parent->%line_ger->$setText(%ger)
			if (!%Wrapper)
			{
				echo"No such object!"
				return
			}
			%array_property[]=$split("\|",%Wrapper->$listproperty(0))
			%i=0
			while(%i !=%array_property[]#)
			{
			%property=$str.section(%array_property[%i],"\,",0,0)
			@%list_name[%i]=$new(listviewitem,$$->$parent->%property)
			@%list_name[%i]->$setText(0,%property)
			@%list_name[%i]-
>$setText(1,$str.section(%array_property[%i],"\,",1,1))
			@%list_name[%i]->$setText(2,%Wrapper->$property(%property,0))
			@%list_name[%i]->$setRenameEnabled(2)
			%i++;
			}

		}
		killPropertyChildren()
		{
			%i=0
			while(%i != @%list_name[]#)
```

```
                        {
                                delete @%list_name[%i]
                                %i++
                        }
                        @%list_name[]= ""
                }
                itemRenamed
                {
                        %property=$0->$text(0)
                        %value=$2
                        eval \%Wrapper\-\>\$setProperty\(%property\,%value\)
                }

        }
        class (dump,widget)
        {
                constructor
                {
                        %Flag=0
                        @$resize(800,600)
                        @%tree[]=$split("\|",$objects.dump(0))
                        @$setCaption("Kvir Objects Dump")
                        @%lay=$new(layout,$$)
                        @%dump=$new(mylistview,$$)
                        @%tree=$new(object)
                        @%property=$new(listview,$$)

                        privateimpl(@%property,itemRenamedEvent)
                        {
                                $$->$parent->%dump->$itemRenamed($0,$1,$2)
                        }
                        @%property->$addColumn("Property",-1)
                        @%property->$addColumn("Type",-1)
                        @%property->$addColumn("Value",-1)
                        @%dump->$addcolumn("Kvirc Widget Objects Tree",300)
                        @%dump->$setRootIsDecorated(1)
                        @%lay->$addwidget(@%dump,0,0)
                        @%lay->$addwidget(@%property,0,1)
                        @%line_find=$new(lineedit,$$)
                        @%line_ger=$new(lineedit,$$)
                        @%lay->$addwidget(@%line_find,1,0)
                        @%lay->$addwidget(@%line_ger,1,1)
                        objects.connect @%line_find returnPressed $$ find
                }
                        createtree()
                        {
                        %string=$0
                        %i=0
                        while (%i != @%tree[]#)
                        {
                                if ($str.findfirst(@%tree[%i],"\>") == -1)
                                {
                                        %parent_current[0]=$new(listviewitem,@%dump)
                                        %child=0
                                }
                                else
                                {
                                        %child=1;
                                        while ($str.find(@%tree[%i],"\>",%child) != -1)
                                        {
                                                %child++;
                                        }
                                        %child--;
                                        %parent_current[%child]=""

        %parent_current[%child]=$new(listviewitem,%parent_current[$(%child-1)])
                                }
                                        %parent_current[%child]->$setopen(1)
                                        if ($str.findfirst(@%tree[%i],%string) != -1)
%parent_current[%child]->$setpixmap(0,39)

        %tmp=$str.mid(@%tree[%i],%child,$($str.len(@%tree[%i])-%child))
                                        %parent_current[%child]->$settext(0,%tmp)
```

```
                                %i++;
                                }
                        @$show()
                        }
                find()
                {
                %tmp=@%line_find->$text()
                @%line_find->$setText("")
                delete @%dump
                @%dump=$new(mylistview,$$)
                @%dump->$addcolumn("Kvirc Widget Objects Tree",300)
                @%dump->$setRootIsDecorated(1)
                @%lay->$addwidget(@%dump,0,0)
                @%dump->$show
                $$->$createtree(%tmp)
                }
        }
        %A=$new(dump)
        %A->$createtree()
}
```