Elettra Sincrotrone Trieste

Elettra
Sincrotrone
Trieste

TANGO

School on TANGO Controls system

Graphical User Interface Design

Giacomo Strangolino

IT programmer at Elettra – Sincrotrone Trieste

Assistant professor 2010-2014, University of Trieste,
Faculty of engineering, principles of computer science

mailto: giacomo.strangolino@elettra.eu
http://www.tango-controls.org

Elettra
Sincrotrone
Trieste

TANGO

Elettra – Sincrotrone Trieste

QTango

A multi threaded framework to develop Tango applications

Part 0

Prerequisites:

The Qt technology

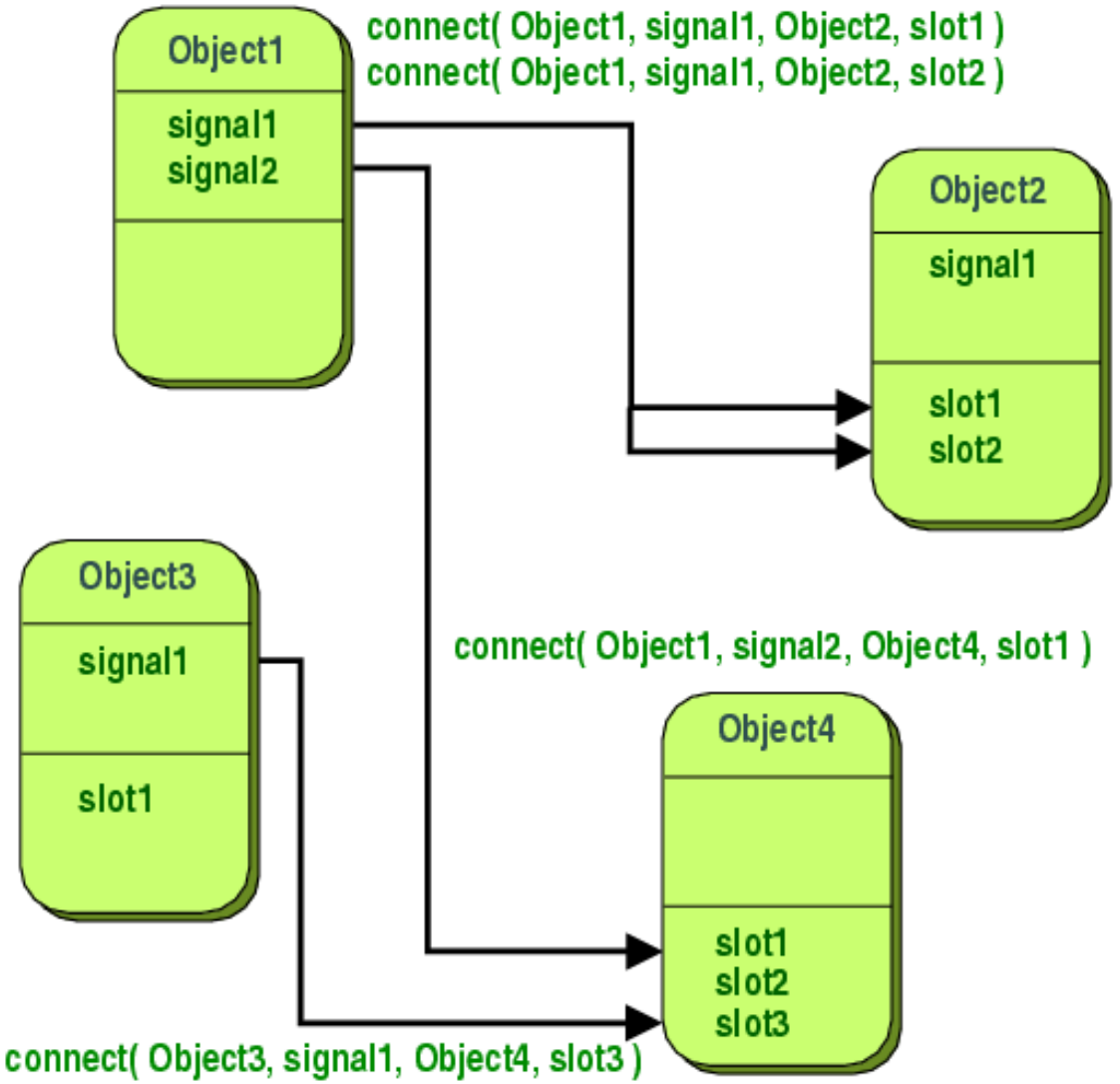
http://www.qt.io/

Prerequisites: Qt

- Qt development libraries installation, Qt *designer* and *qtcreator* IDE, http://doc.qt.io/qt-5/topics-app-development.html;
- http://doc.qt.io/qt-5/gettingstarted.html
- *QPainter* API;
- *QObjects*, Properties, and Events;
- *Signals* and *slots*
- *Qt designer*

Layouts!

Signals and slots (II)



Qt: see also...

- *QThread* http://doc.qt.io/qt-5/qthread.html
- *QWidget http://doc.qt.io/qt-5/qwidget.html*
- *QObject http://doc.qt.io/qt-5/qobject.html*
- *http://doc.qt.io/qt-5/groups.html* (classes grouped by functionality)

Part I

QtangoCore architecture overview

Overview (I)

- **Fast and easy development of graphical widgets integrated with the Tango control system;**
- **Integrated *Tango Exception* management and logging;**
- **Multi threaded environment for the creation of efficient and fully responsive graphical user interfaces:**
  - *Fulfils **Human Computer Interaction** Principles for GUI design;*

Overview (II)

- Reconnection to the device at startup;
- *asynchronous* execution of *targets* (write attributes, commands) (i.e. In the *DeviceThread*);
- get **attribute properties** at configuration time and get them asynchronously;
- get **device** and **class** properties through the *PropertyReader* utility class (blocking or asynchronous);
- monitor quantities and create custom widgets with *QTWatcher* and write or create writers with *QTWriter* utility classes;

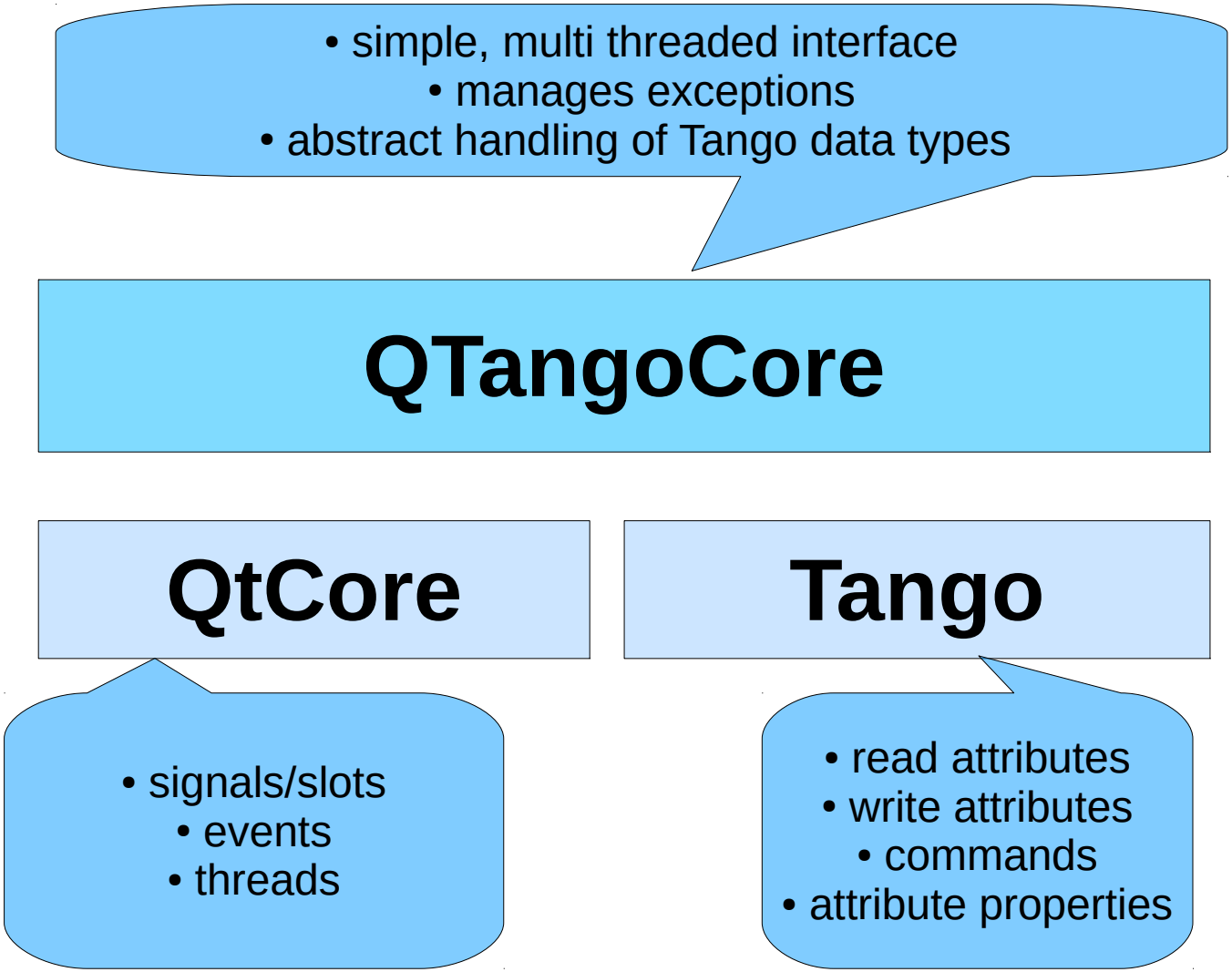
Overview (III)

→ Connection setup: try with events, fallback on polling (*AUTO_REFRESH*), unless otherwise specified:

- *ActionFactory::actionFactory() → setDefaultRefreshMode*
- *export DEFAULT_REFRESH_MODE=POLLED_REFRESH*
- The refresh mode can also be specified per widget (designer)

→ Polling is stopped when widget is not visible

Overview (III)



Elettra
Sincrotrone
Trieste

TANGO

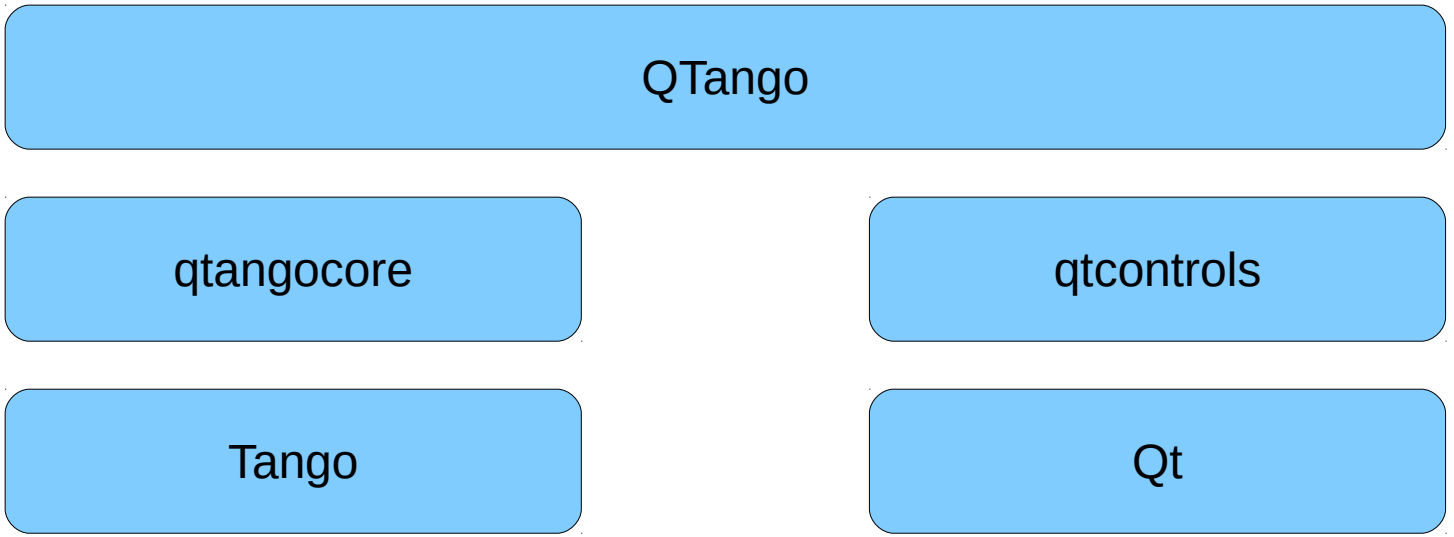
Part II

QTango

a set of Qt widgets integrated

with QTangoCore

QTango infrastructure



Overview

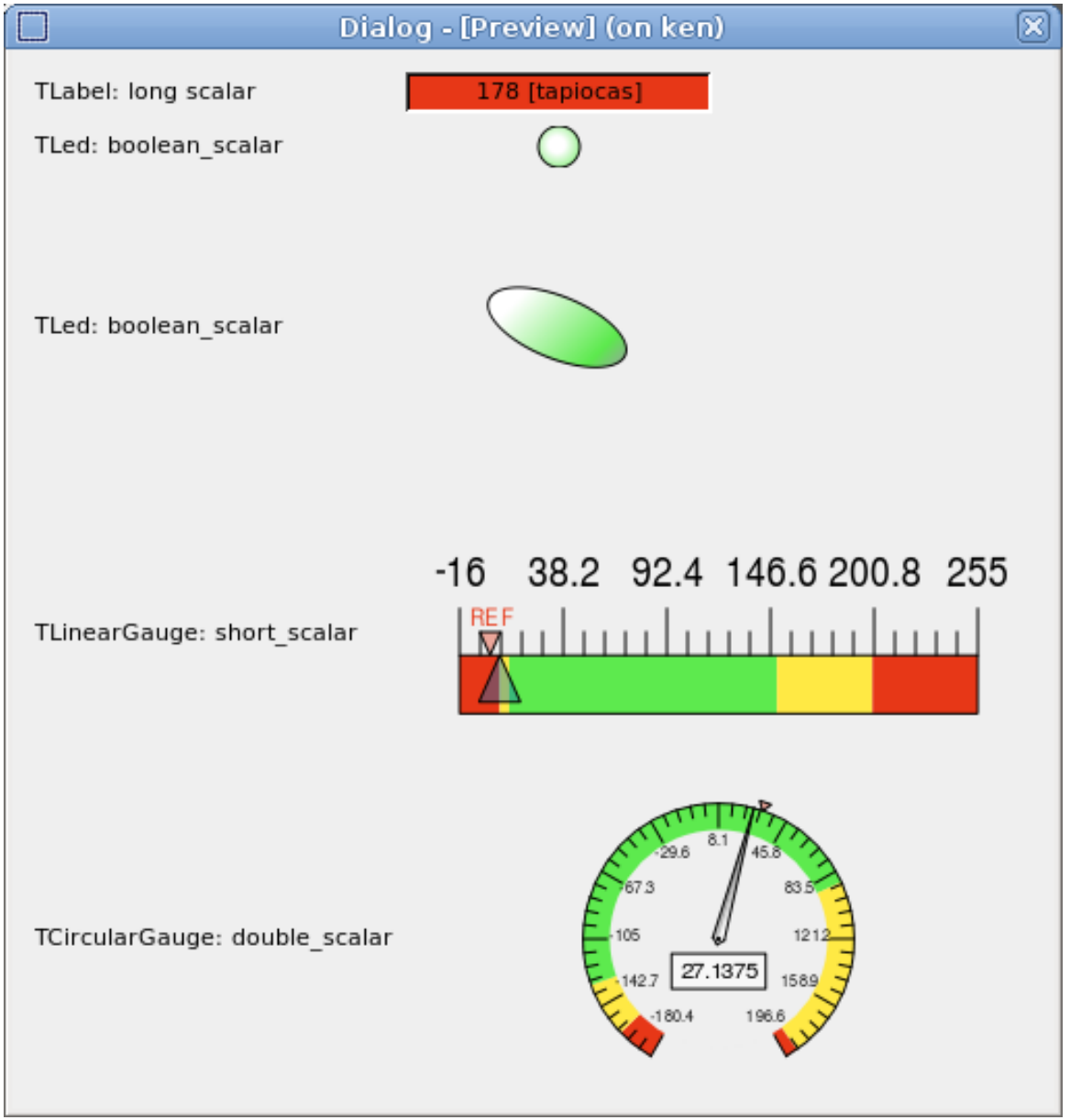
Right click on a widget:

- view trend of scalar attribute values (plot);
- show tango point information (connection status, time stamp, data type, refresh mode, polling period, and so on…);
- helper application (defined as an attribute or device property or in a widget property);
- copy source into clipboard.

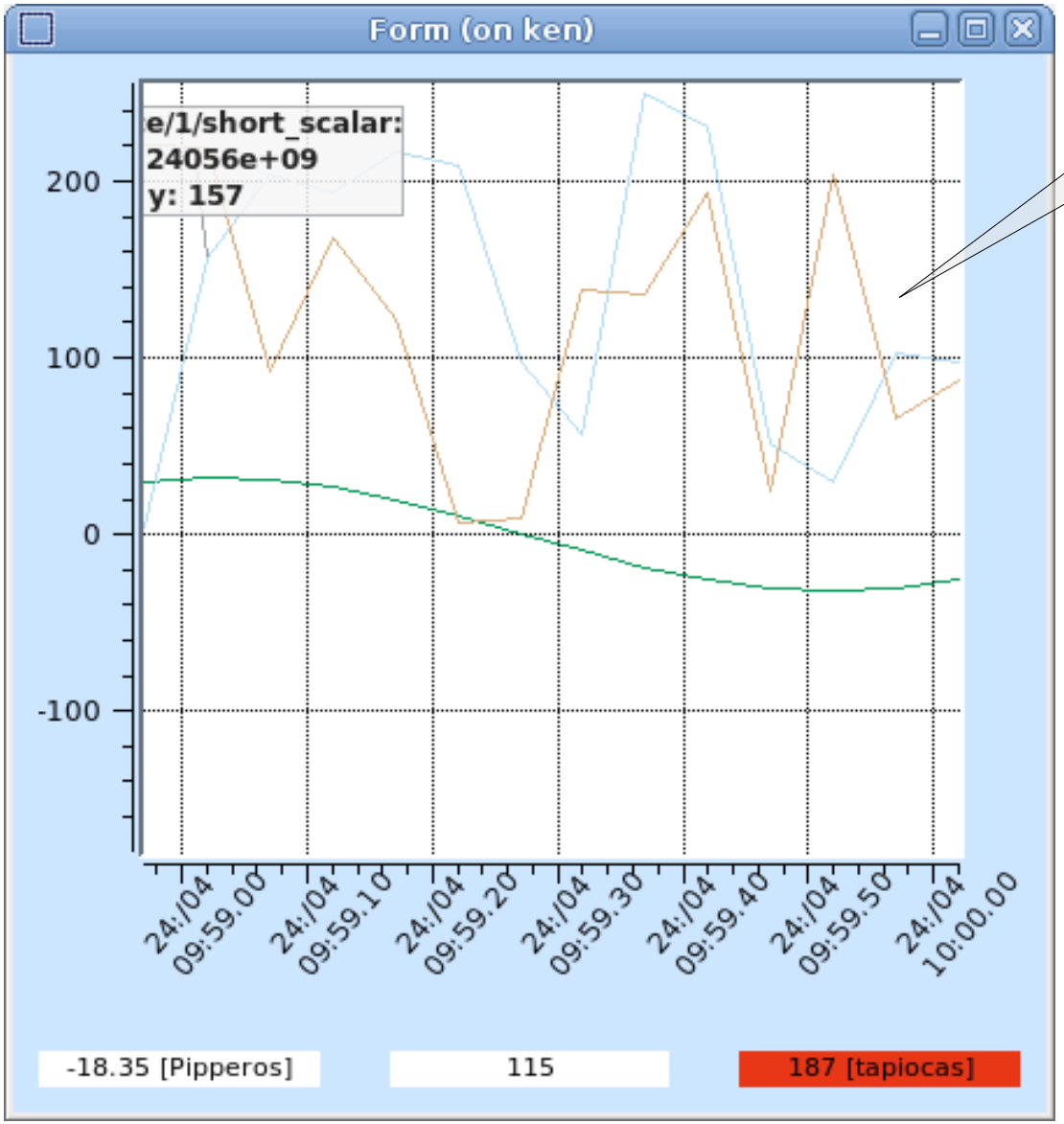
- Stop reading while hidden;

Elettra
Sincrotrone
Trieste

Readers



Readers (II)

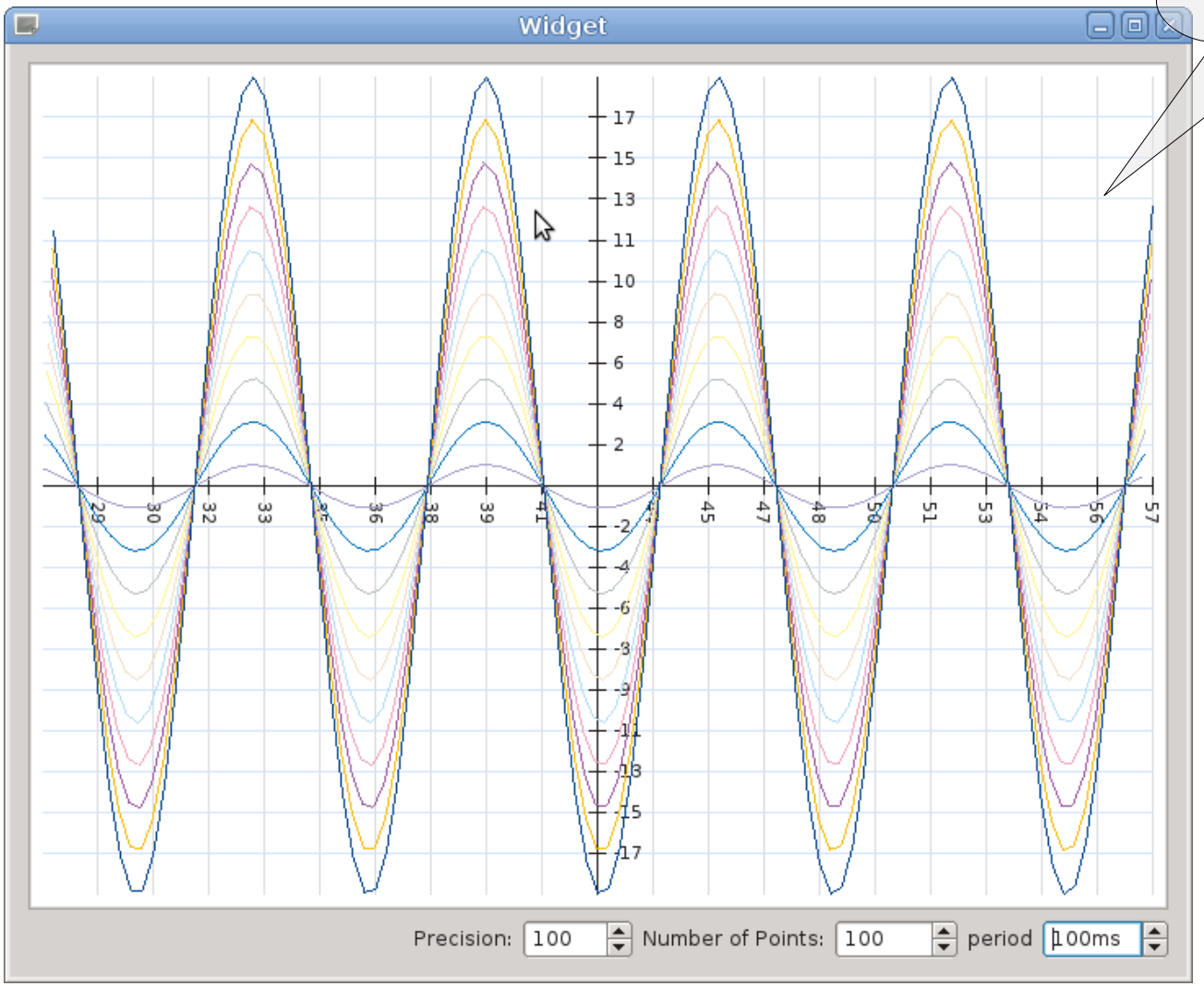


TPLotLight

Elettra
Sincrotrone
Trieste

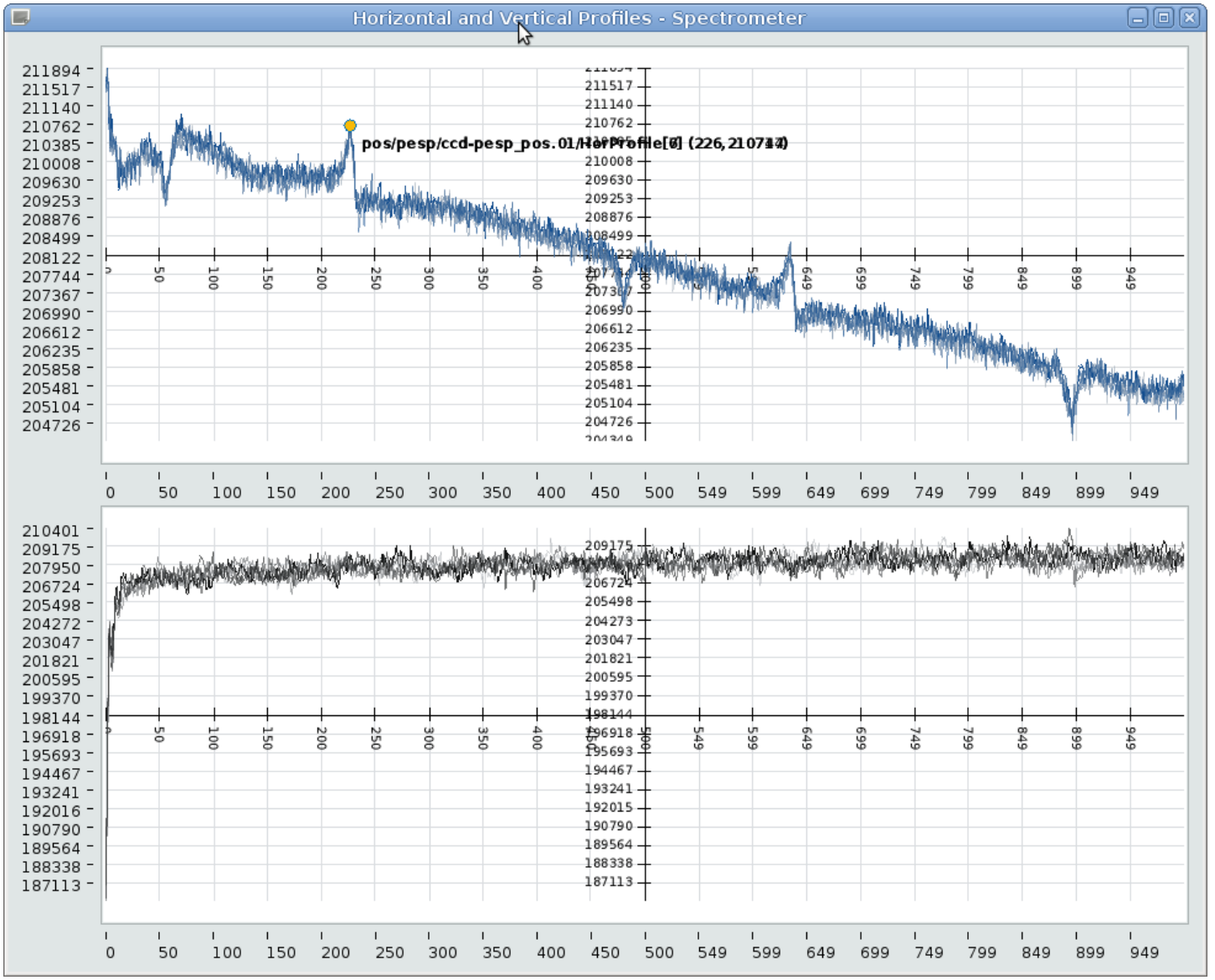
Readers (III)

TGraphicsPlot

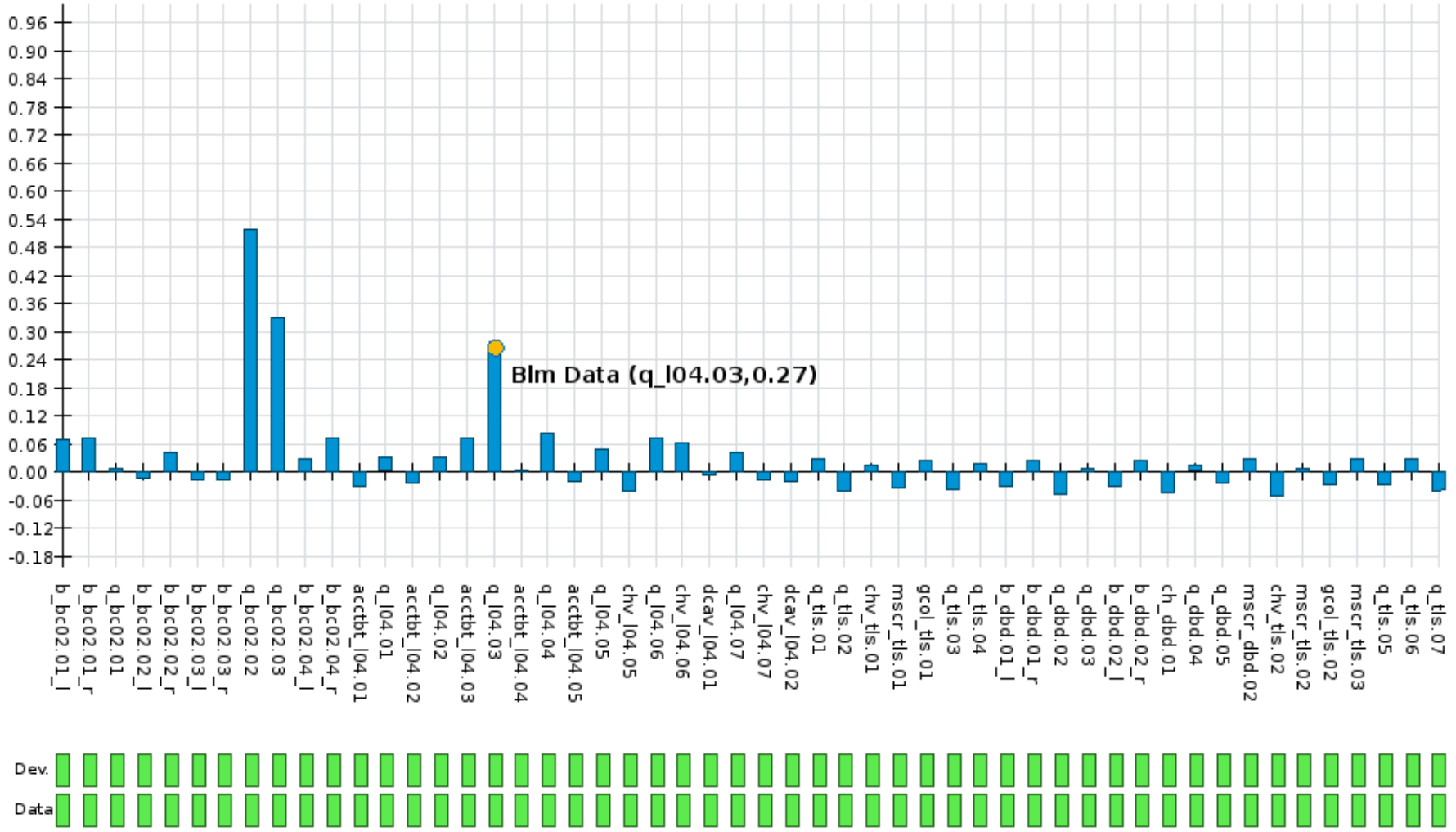


Uses Qt *QGraphicsView/QGraphicsScene* technology

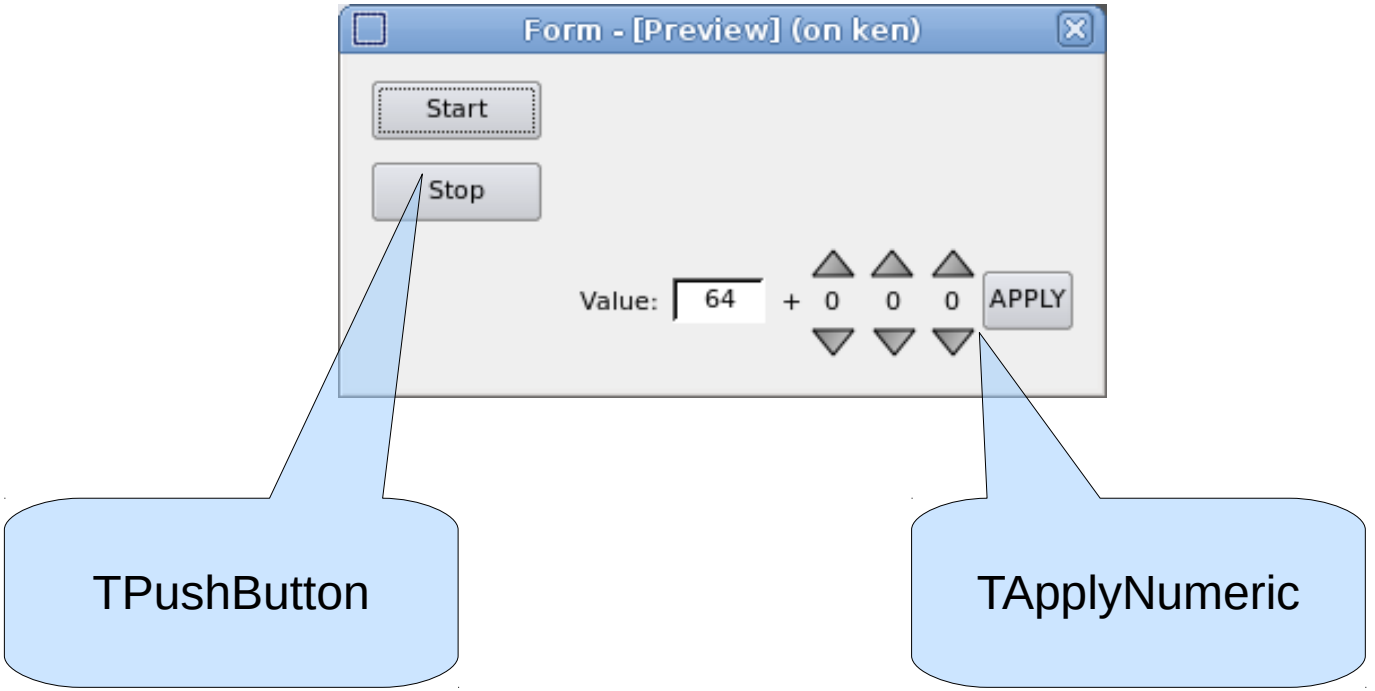
Readers (IV): QGraphicsPlot (II)



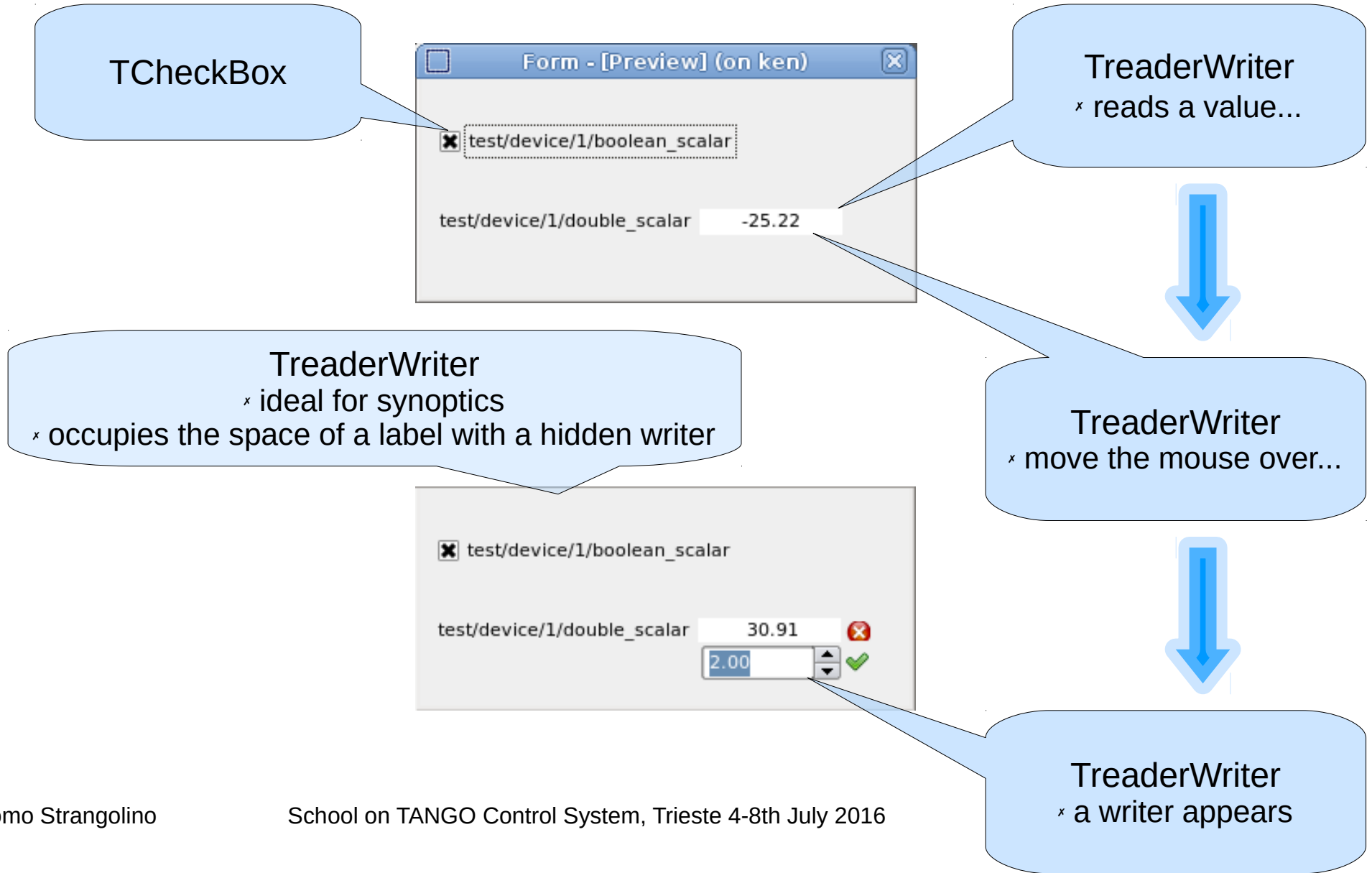
Readers (IV): QGraphicsPlot (IV)



Writers

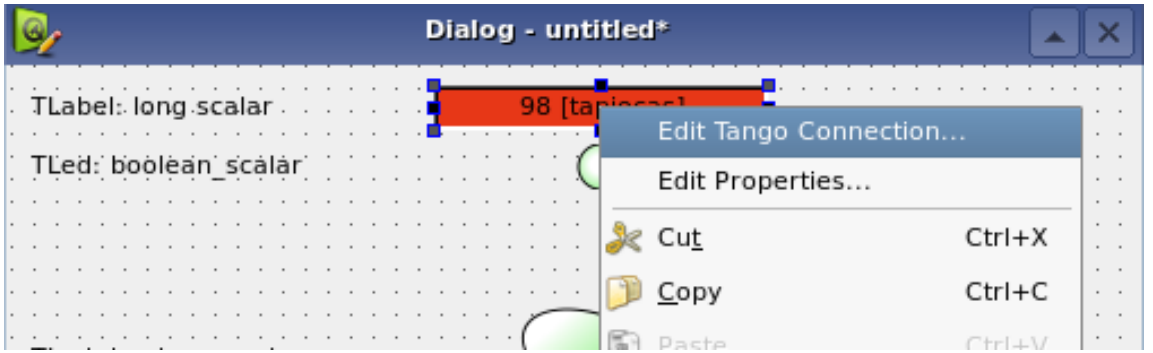

Form - [Preview] (on ken)
Start
Stop
Value:
64
+
0
0
0
APPLY
TPushButton
TApplyNumeric

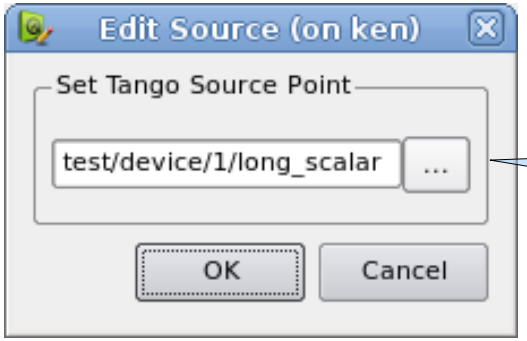
Readers *and* Writers



Qt Designer integration

Easy configuration of tango ***source*** (for readers) and ***target*** (for writers)



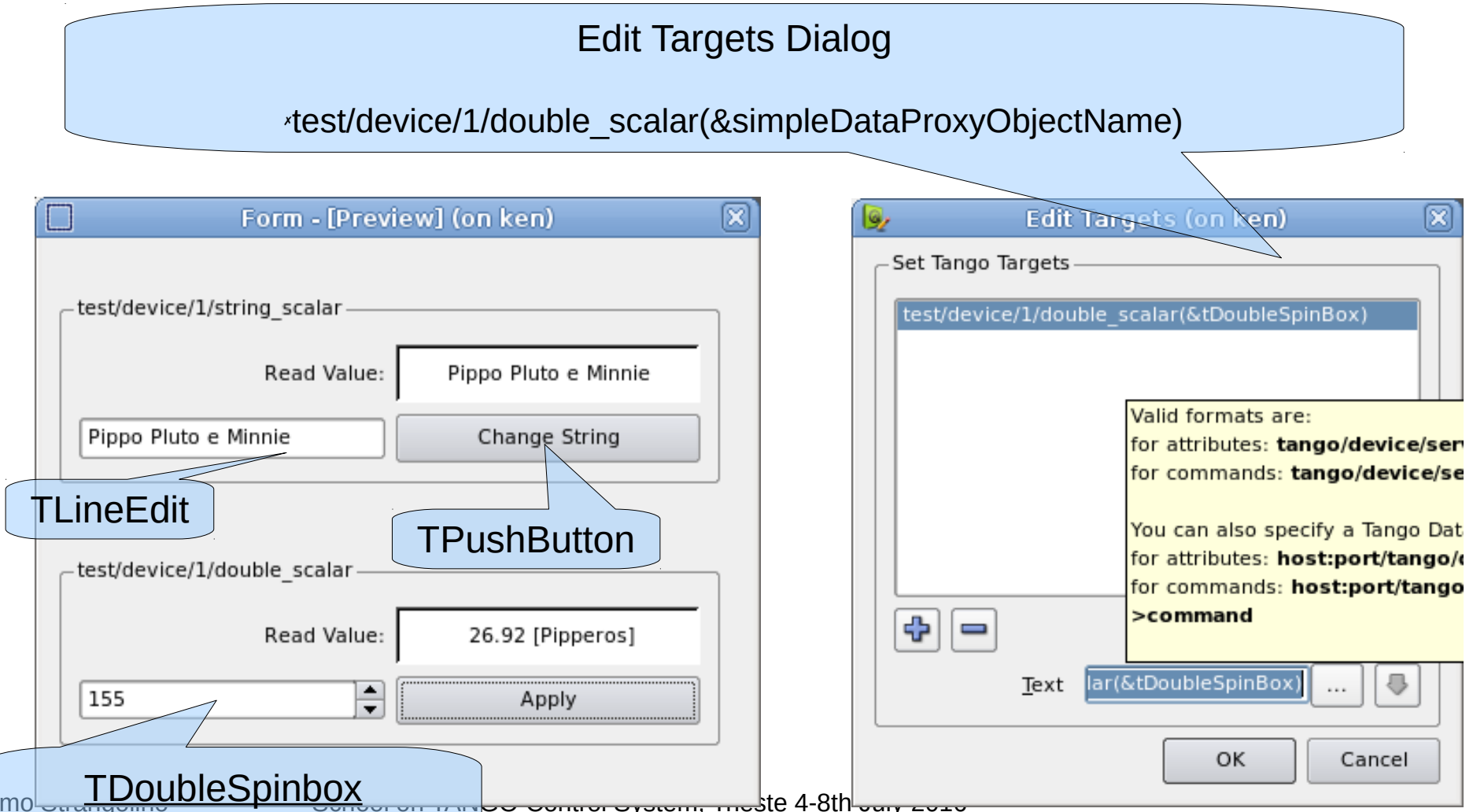


Edit Source dialog
- test/device/instance/attribute_name
- test/device/instance->command_name(argin)

Drag and drop from Jive!

SimpleDataProxy for writers

SimpleDataProxy elements *display* data that can be used as *input arguments* for commands or attributes on *writers*



Reading and writing *Spectrum* attributes

TSpectrumButton: writes into a *spectrum* attribute fetching data from *SimpleDataProxy* widgets (*TLineEdit*, *TNumeric*, user defined...);

TwidgetGroup: groups a set of readers and refreshes them with the values extracted from a *spectrum* attribute;

→ Full *Qt designer* integration and configuration!

Elettra Sincrotrone Trieste

Part III

Programming with QtangoCore

Create an object (*QWidget* or *QObject*) reading from and writing to a *Tango* device server

Includes

.pro project file:

```
include(/usr/local/qtango/include/qtango6/qtango.pri)
```

QTangoCore stuff in .h files

```
#include <com_proxy_reader.h>
#include <com_proxy_writer.h> /* for writers */
#include <tvariant.h>
```

Connection

Readers

setSource

Attributes

test/device/1/double_scalar

test/device/1->DevDouble

Commands

Writers

setTargets

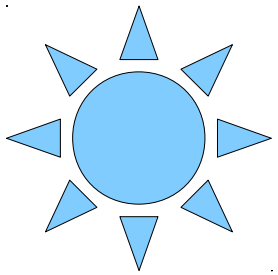
unsetSource, clearTargets, setPeriod, setRefreshMode

Reader

- Readers must inherit from *QTangoComProxyReader*
- readers must implement the *pure virtual* method *refresh*()
- the *refresh*() method has a *TVariant* as argument. It contains the data read from the *Tango* layer.
- *connect() reader's qTangoCommunicationHandle newData() signal to the refresh() slot*
- (Optional) inherit from *QTangoWidgetCommon* in order to obtain a common behavior among QTango widgets (copy source, view trend, helper application). No methods shall be reimplemented.

auto configuration

- The Tango attribute must be configured into the database with its *minimum and maximum values* (also warning and alarm thresholds, if desired);
- Must connect the reader's *handle signal* *attributeAutoConfigured(const TangoConfigurationParameters *)* to your configuration *slot;*
- If *Tango events* are available, you may receive *attribute configuration events* via the connected *slot*



To disable attribute configuration change events:

export QTANGO_ATTR_CONF_CHANGE_DISABLED=1

Connection: configuration

QObject!

```
connect(myReader->qtangoComHandle(),

SIGNAL(attributeAutoConfigured(const
TangoConfigurationParameters *)),

        this,

SLOT(readerConfigured(const
TangoConfigurationParameters *)));
```

QObject!

Connection: configuration (II)

TangoConfigurationParameters

- *double maxValue() const { return mxValue; }*
- *double minValue() const { return mValue; }*
- *double maxWarning() const { return mxWarning; }*
- *double maxError() const { return mxError; } […]*
- *bool maxIsSet() const { return d_maxIsSet; }*
- *bool minIsSet() const { return d_minIsSet; }*
- *bool MErrIsSet() const { return d_MErrIsSet; }*
- *bool mWarnIsSet() const { return d_mWarnIsSet; } […]*
- *QString description() const { return d_desc; }*
- *QString label() const { return d_label; }*
- *QString stdUnit() const { return d_stdUnit; }*
- *QString displayUnit() const { return d_displayUnit; }*
- *QString format() const { return d_format; }*
- *TVariant currentValue()*

First available read value

Connection: new data signal

QObject!

```
connect(myReader->qtangoComHandle(),
SIGNAL(newData(const TVariant&)),
   this,
SLOT(refresh(const TVariant&) ));
```

QTangoCore variant data type

Inside refresh(), extract the data

refresh() (in a reader)

- Using TVariant, test the attribute quality;
- see if *canConvert()* to the required type;
- if yes, convert it into the desired type
- do whatever you like with the extracted data

Can convert to a certain data type?

- *bool canConvertToState() const;*
- *bool canConvertToString() const;*
- *bool canConvertToInt() const;*
- *bool canConvertToUInt() const;*
- *bool canConvertToDouble() const;*
- *bool canConvertToBool() const;*
- *bool canConvertToStringVector() const;*
- *bool canConvertToIntVector() const;*
- *bool canConvertToDoubleVector() const;*
- *bool canConvertToBoolVector() const;*

TVariant (II)

Yes, can convert

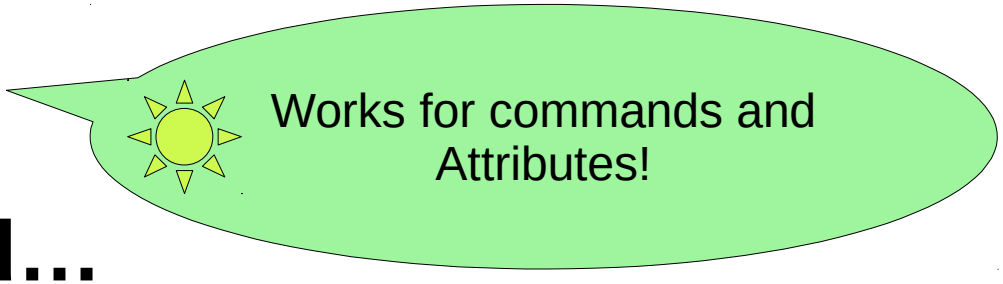
DevState	*toState() const;*
QString	*toString(bool = true) const;*
int	*toInt(bool = true) const;*
unsigned int	*toUInt(bool = true) const;*
double	*toDouble(bool = true) const;*
bool	*toBool(bool = true) const;*
QVector<QString>	*toStringVector(bool = true) const;*
QVector<int>	*toIntVector(bool = true) const;*
QVector<unsigned int>	*toUIntVector(bool = true) const;*
QVector<double>	*toDoubleVector(bool = true) const;*
QVector<bool>	*toBoolVector(bool = true) const;*

...

TVariant (III)

Get Tango data structures

- *AttributeInfo getAttributeInfo () const*
- *CommandInfo getCommandInfo () const*
- *CmdArgType type() const*
- *AttrQuality quality() const*



And…

- *QString message() const*
- *const struct timeval *timeReadRef() const*
- *struct timeval timeRead() const*
- *QString tangoPoint() const*

QTangoWidgetCommon

Provides a common set of functionalities for QTango widgets

- View trend
- Helper application
- Copy source
- Connection state

```
#include <qtango_wcommon.h>
#include <com_proxy_reader.h>
#include <QLabel>

class ReadLabel : public QLabel,
        public QTangoComProxyReader,
        public QTangoWidgetCommon
 {
 }
```

No method shall be reimplemented from QTangoWidgetCommon

Example: reader implementation

The reader will be able to:

- *read an attribute*;
- *auto configure* itself to be aware of warning and alarm thresholds;
- associate a *helper application* to the connected *source*.

Writer

inherits *QTangoComProxyWriter*

- auto configuration available (see considerations done for the reader)
- write is performed inside *QTangoComProxyWriter*'s *execute()* implementation

Simple Data Proxy

- provides **input data** for your **writers**;
- any QWidget displaying something can be used to implement a simple data proxy:
  - QLabel
  - Q[Double]SpinBox
  - QTextEdit/QTextBrowser
  - QComboBox
  - QLineEdit
  - …

just implement the virtual slot getData, returning a string representation of the data displayed by the widget

Simple Data Proxy (II)

- inherit from *SimpleDataProxy*;
- implement the pure *virtual QString getData()* method
- example: *QTango TLineEdit*

Elettra
Sincrotrone
Trieste

TANGO

QTWatcher and QTWriter classes

Elettra Sincrotrone Trieste

QTWatcher

TANGO

- Reads Tango variables using QTango;
- QObject or base types can be *attached()*;
- on new data, a SLOT can be invoked on the QObject;
- the *data type* is guessed from the QObject SLOT input parameter
- *auto configuration* possible if QObject has suitable slots (e.g. *QProgressBar setMinimum()*)
- On read error, *slots aren't invoked and variables aren't updated*!

QTWatcher with QObjects


```
QProgressBar *pbar = new QProgressBar(this);
QTWatcher *pbarWatcher = new QTWatcher(this);

pbarWatcher->attach(pbar, SLOT(setValue(int)));

// configure maximum and minimum values when available
pbarWatcher->setAutoConfSlot(QTWatcher::Min, SLOT(setMinimum(int)));
pbarWatcher->setAutoConfSlot(QTWatcher::Max, SLOT(setMaximum(int)));

pbarWatcher->setSource("$1/short_scalar_ro");
```

QTWatcher with simple data types

```
short int var;
QTWatcher *intWatcher = new QTWatcher(this);

 pbarWatcher->attach(&var);

 pbarWatcher->setSource("$1/short_scalar_ro");
```

- var is always up to date;
- tango reads are performed in another thread;
- it is safe to access var in any moment inside your thread.

QTWatcher: signals

- attributeAutoConfigured(const TangoConfigurationParameters *);
- connectionFailed();
- connectionOk(bool);
- connectionErrorMessage(const QString &);
- readOk(bool);
- newData(int), newData(double), … , newData(const QString&).

QTWriter

Elettra Sincrotrone Trieste

TANGO

- Write an attribute or give a command from any QObject or Qwidget;
- a *signal* of the QObject is connected to a compatible *execute()* method implemented in QTWriter;
- a *set point slot* can be provided to initialize the object with the current value at auto configuration time;
- data type automatically detected from the *signal* specified!

QTWriter

Elettra
Sincrotrone
Trieste

TANGO

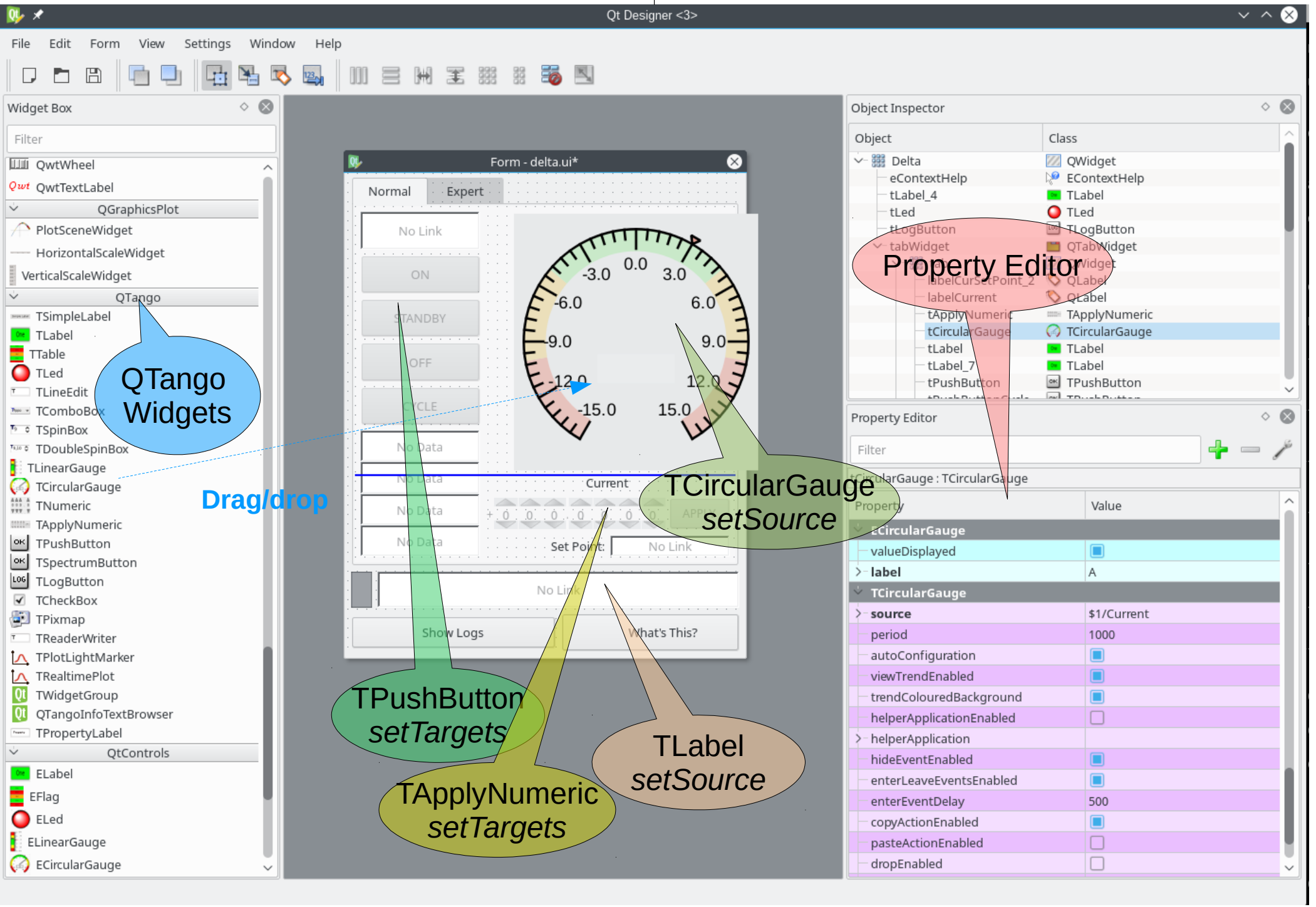
```
QLineEdit *lineEdit = new QlineEdit(this);
QTWriter *lineEditWriter = new QTWriter(this);

lineEditWriter->attach(lineEdit,
        SIGNAL(textChanged(const QString&)),
        SLOT(setText(const Qstring&)));

lineEditWriter->setTargets("test/device/1/string_scalar");
```

The Qt designer

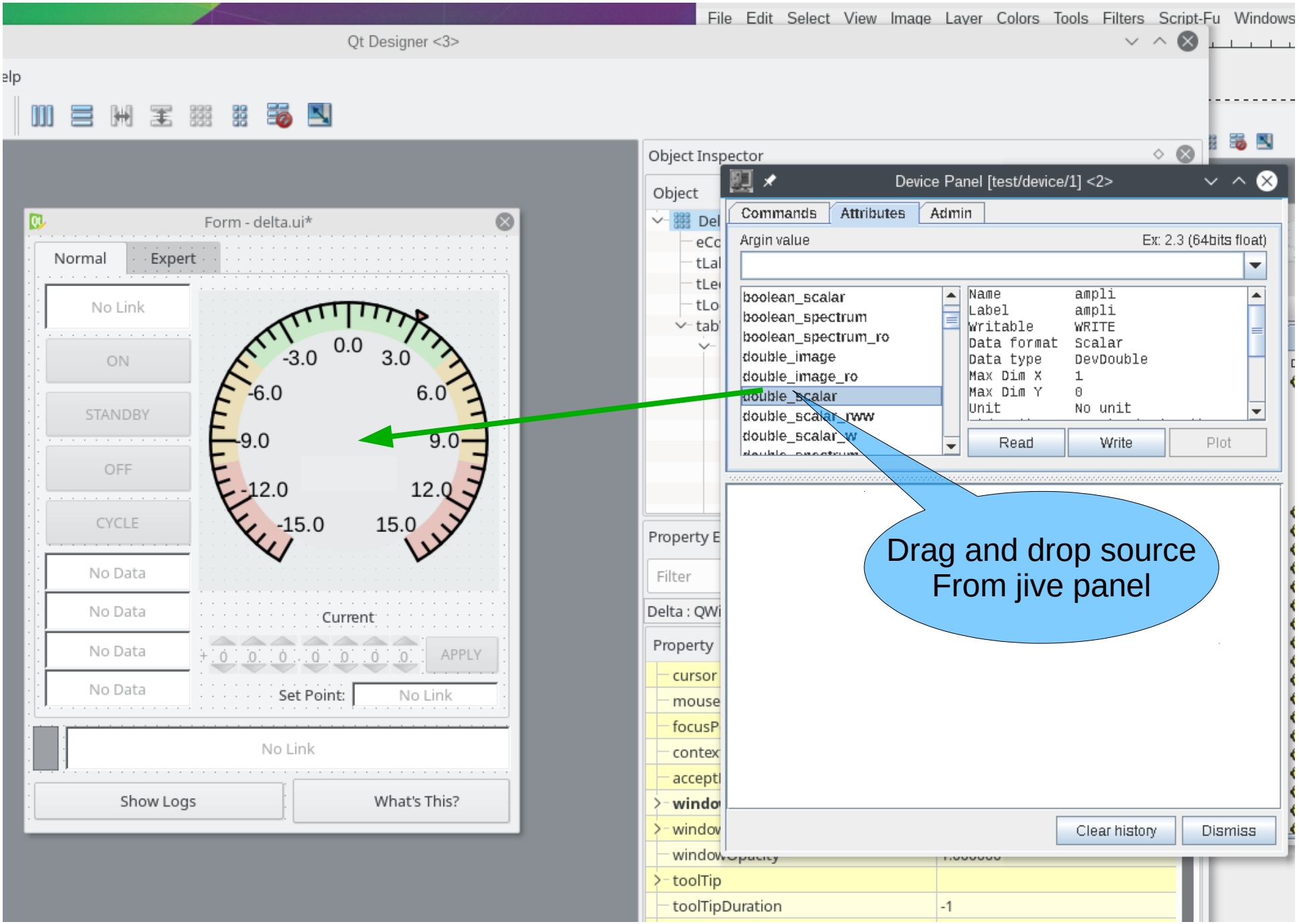
QTango plugins

QTango, *qtcontrols*, *QGraphicsPlot* and *TGraphicsPlot* plugins are available in the Qt designer.

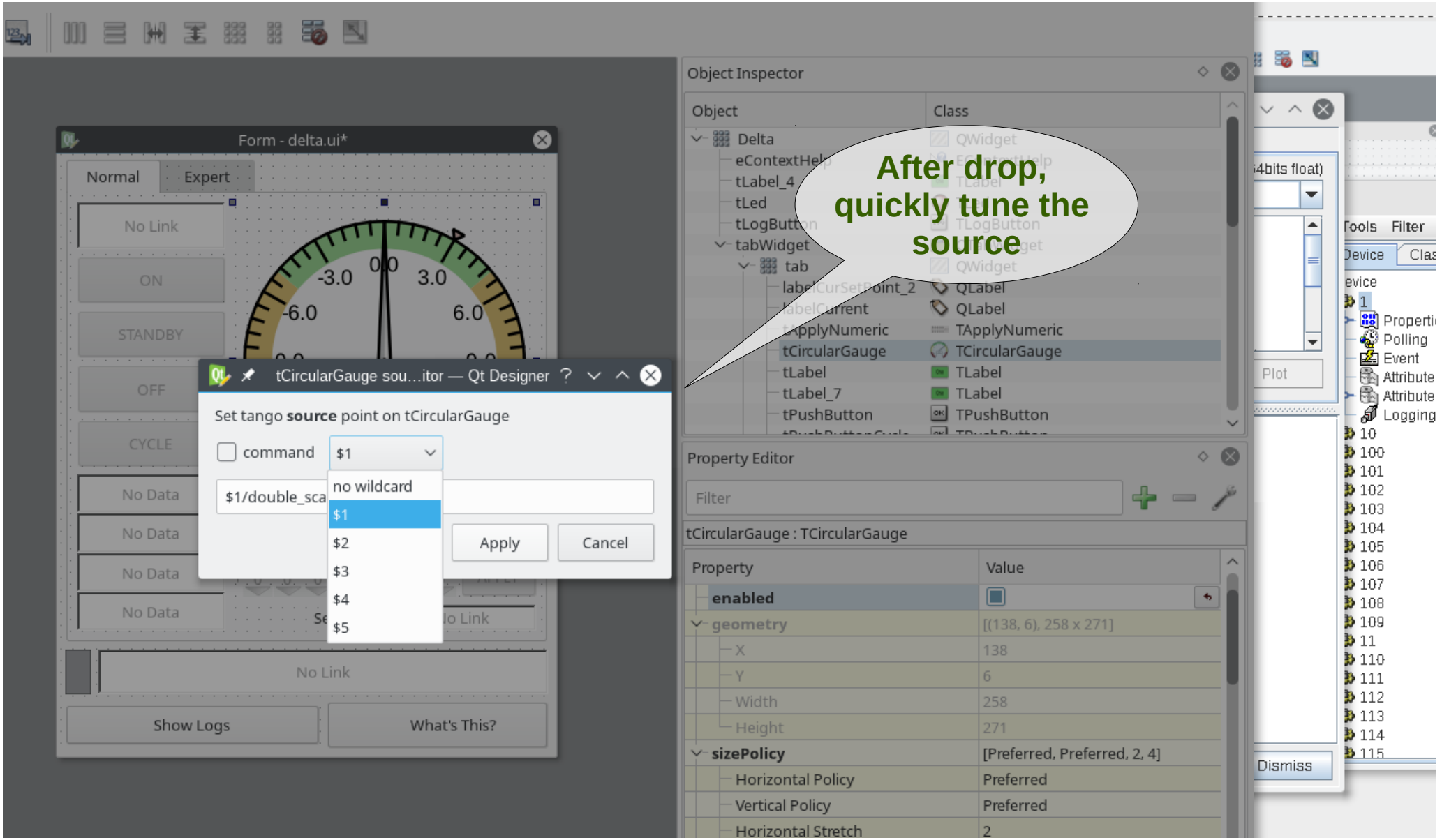
- Drag widgets from the *Widget Box* and drop them into the project widget
- Edit Qt, qtcontrols and QTango properties from the *Property Editor*;
- Right click on a widget to set the *QTango* source or *targets*


Qt Designer <3>
File Edit Form View Settings Window Help
Widget Box
Filter
QwtWheel
QwtTextLabel
QGraphicsPlot
PlotSceneWidget
HorizontalScaleWidget
VerticalScaleWidget
QTango
TSimpleLabel
TLabel
TTable
TLed
TLineEdit
TComboBox
TSpinBox
TDoubleSpinBox
TLinearGauge
TCircularGauge
TNumeric
TApplyNumeric
TPushButton
TSpectrumButton
TLogButton
TCheckBox
TPixmap
TReaderWriter
TPlotLightMarker
TRealtimePlot
TWidgetGroup
QTangoInfoTextBrowser
TPropertyLabel
QtControls
ELabel
EFlag
ELed
ELinearGauge
ECircularGauge
QTango Widgets
Drag/drop
Form - delta.ui*
Normal
Expert
No Link
ON
STANDBY
OFF
CYCLE
No Data
No Data
No Data
No Data
Current
Set Point:
No Link
No Link
Show Logs
What's This?
TCircularGauge setSource
TPushButton setTargets
TApplyNumeric setTargets
TLabel setSource
Object Inspector
Object
Class
Delta
QWidget
eContextHelp
EContextHelp
tLabel_4
TLabel
tLed
TLed
tLogButton
TLogButton
tabWidget
QTabWidget
labelCurSetPoint_2
QLabel
labelCurrent
QLabel
tApplyNumeric
TApplyNumeric
tCircularGauge
TCircularGauge
tLabel
TLabel
tLabel_7
TLabel
tPushButton
TPushButton
Property Editor
Property Editor
Filter
tCircularGauge : TCircularGauge
Property
Value
ECircularGauge
valueDisplayed
label
A
TCircularGauge
source
$1/Current
period
1000
autoConfiguration
viewTrendEnabled
trendColouredBackground
helperApplicationEnabled
helperApplication
hideEventEnabled
enterLeaveEventsEnabled
enterEventDelay
500
copyActionEnabled
pasteActionEnabled
dropEnabled
Property Editor

Drag 'n drop from Jive



Drag 'n drop from Jive (II)



QTango plugin components

readers

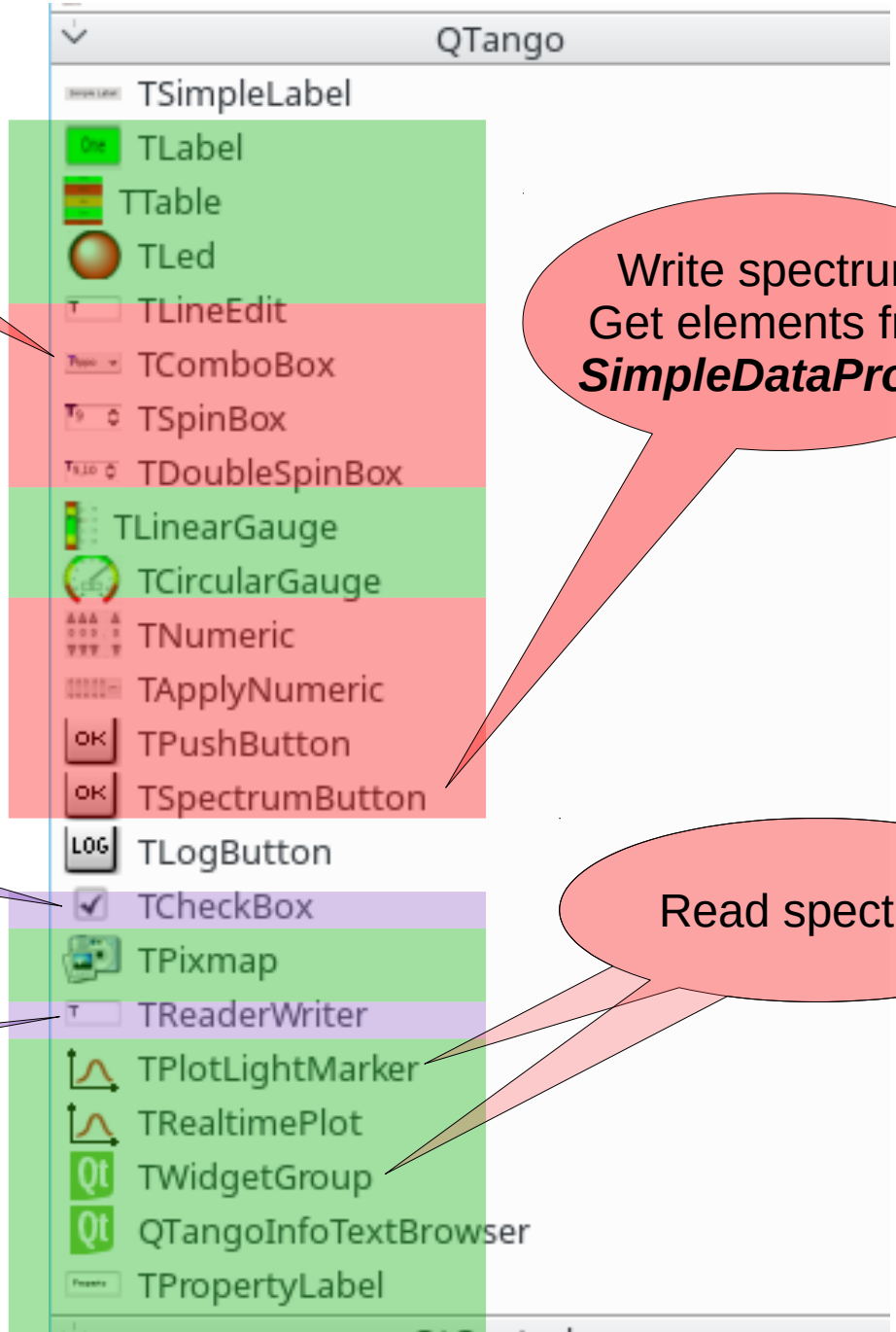
writers

Readers *and* writers

TComboBox:
Initialize with *values*
Attribute property!

NOTE: for historical reasons,
You must edit <u>both</u> source and
Targets

If no targets set,
targets = source



Write spectrum.
Get elements from
SimpleDataProxys

Read spectrum

QTango plots

TPlotLightMarker (Qwt) with a scalar attribute/command as source:

- Plots attribute value over time
- X axis is configured as a *time scale*

***source**:* semicolon separated list of sources

TPlotLightMarker (Qwt) with a spectrum attribute/command as source:

- Plots spectrum
- X axis is [0, 1, … , spectrum.size() - 1];
- Y axis is [spectrum[0], spectrum[1], … , [spectrum[spectrum.size() - 1]

TRealTimePlot (Qwt) (spectrum, command)

- Tailored for *GetSomething(N, M)* commands used for real time quantities.
- Only for commands that return a vector.
- Tested at Fermi with several curves refreshed at 10Hz.

QTango plots (II)

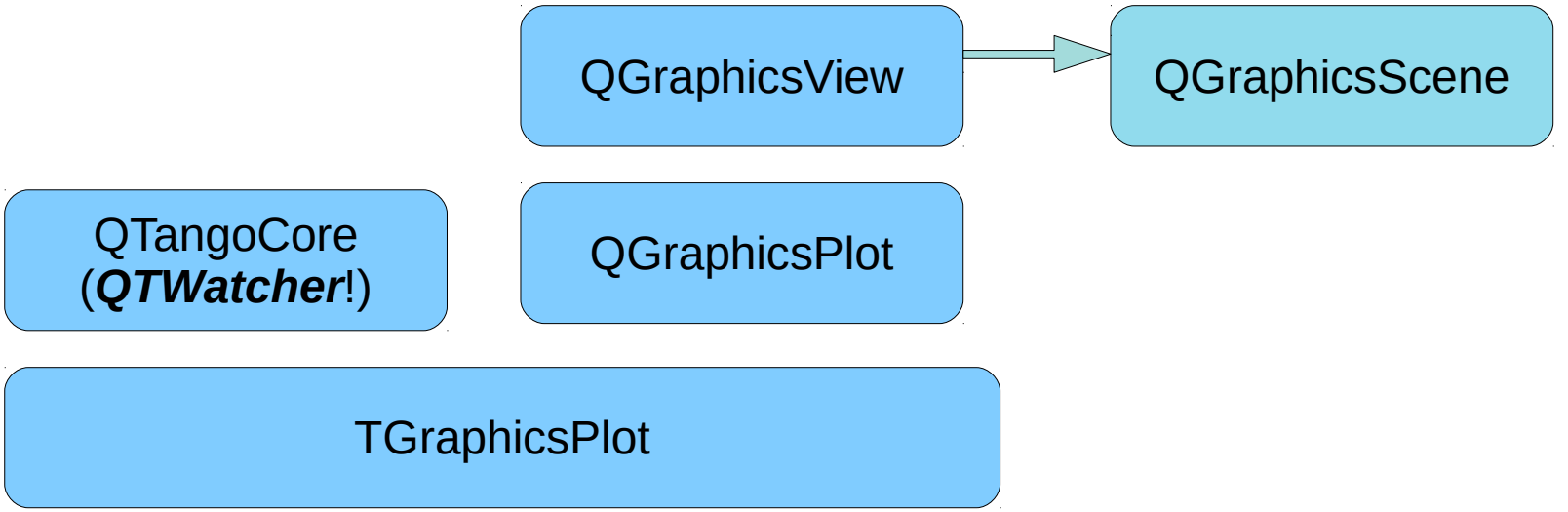
TGraphicsPlot

sourcesList:
QStringList of sources

QGraphicsView / QGraphicsScene technology

- a surface for managing and interacting with a large number of custom-made 2D graphical items
- a view widget for visualizing the items
- support for zooming and rotation

http://doc.qt.io/qt-4.8/graphicsview.html



Exercise

Qt designer

Using the Qt designer, make an application made up of QTango widgets only.
Configure the *test/device/1* server so that the attribute *short_scalar* has a property named *values* which is a list of strings (try with at least 4 elements).
Configure the *test/device/1* with a device property named *helperApplication* and value *xclock*.
A TabWidget with two tabs shows in the first page:

- A TLabel will read the *short_scalar*, an associated TComboBox allows to select one of the available *values*, and a *TPushButton* will write the attribute.
- A TLabel displays the state of the device.
- A TPushButton “SwitchStates” executes that command.
- A TReaderWriter is connected to *double_scalar.*
- A TReaderWriter reads *string_scalar.*
- A TPlotLightMarker reads *double_scalar* and *long_scalar*.
- A TGraphicsPlot reads *double_spectrum_ro and long_spectrum_ro.*

Exercise 4 (II)

Qt designer

In the second page:

- a TWidgetGroup reads the *double_spectrum*
- a set of 4 *TDoubleSpinBox* write the first 4 elements of the vector when an Apply *TSpectrumButton* is clicked.
- A plot of your choice connects to the same attribute.
- A TTable with 10 rows and 2 columns displays the *boolean_spectrum* attribute.

⇨ Set *double_spectrum* range in Tango database and verify that the double spinboxes are correctly configured.

⇨ If you configure *short_scalar values* property, remember to limit the range of the attribute accordingly. For example, if you put 6 string values, limit the attribute between 0 and 5 (Simply apply a value from the combo box list).

⇨ *IndexMode* property on *TComboBox* and the *configureEnumFromValuesProperty* on *TLabel* must be enabled.

Elettra Sincrotrone Trieste

Part IV

TANGO

Writing *QTango* - ready Tango servers

- Correctly shape the *Tango* server paying special attention to **command** and **attribute** modeling;
  - commands only when they suit the device model;
  - no commands with strings as *argin* and/or *argout*;
  - put logic on the server rather than in the panel, as much as possible

Documentation

- *QTango* documentation is installed inside the *share* folder under the root installation of qtango (see qtango.pri project file)
- QTango documentation is in the *html* format.

Logging

- QTangoCore provides console coloured messages:

* ***error*** *message*

* ***warning*** *message*

* ***ok*** *message*

Enable them exporting ***QTANGO_PRINT=1*** *in the terminal*

The End

- **Thanks for your attention**

mailto: giacomo.strangolino@elettra.trieste.it