Giacomo Strangolino
Elettra – Sincrotrone Trieste

QTangoCore

A multi threaded framework to develop *Tango* applications

mailto: giacomo.strangolino@elettra.trieste.it

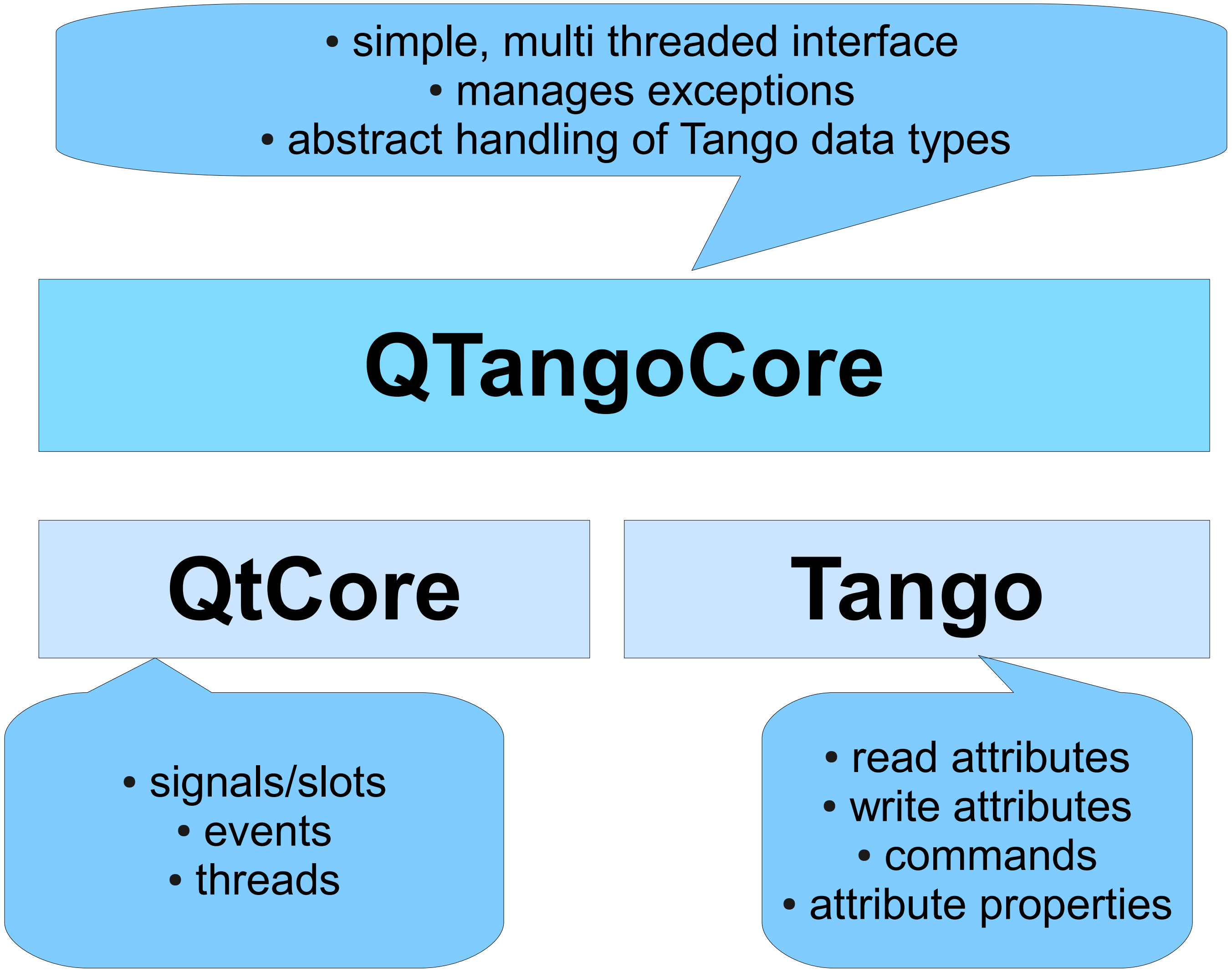
Part I

QtangoCore architecture overview

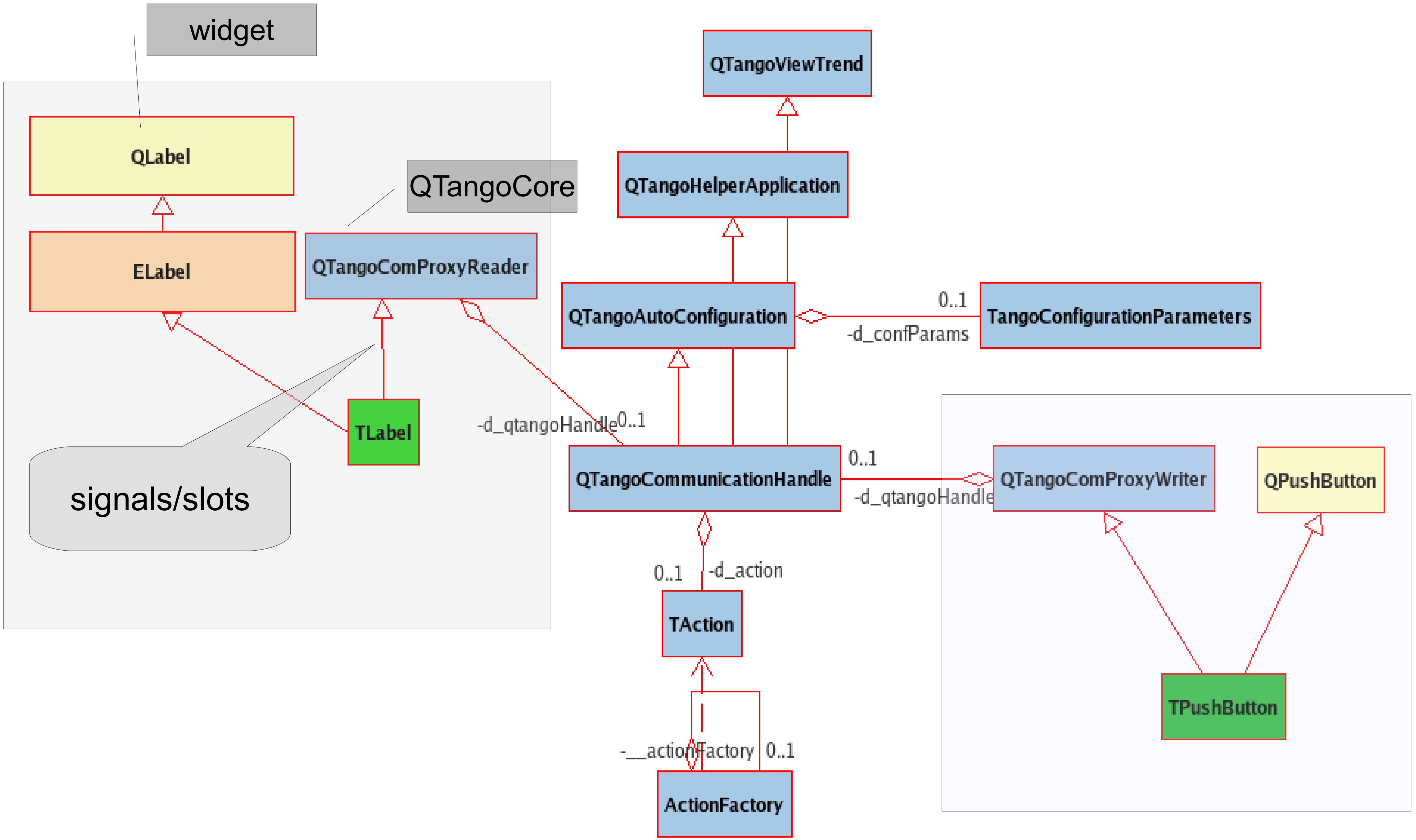
Overview (II)

- **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;*
  - *Threads are grouped by device to optimize their number*

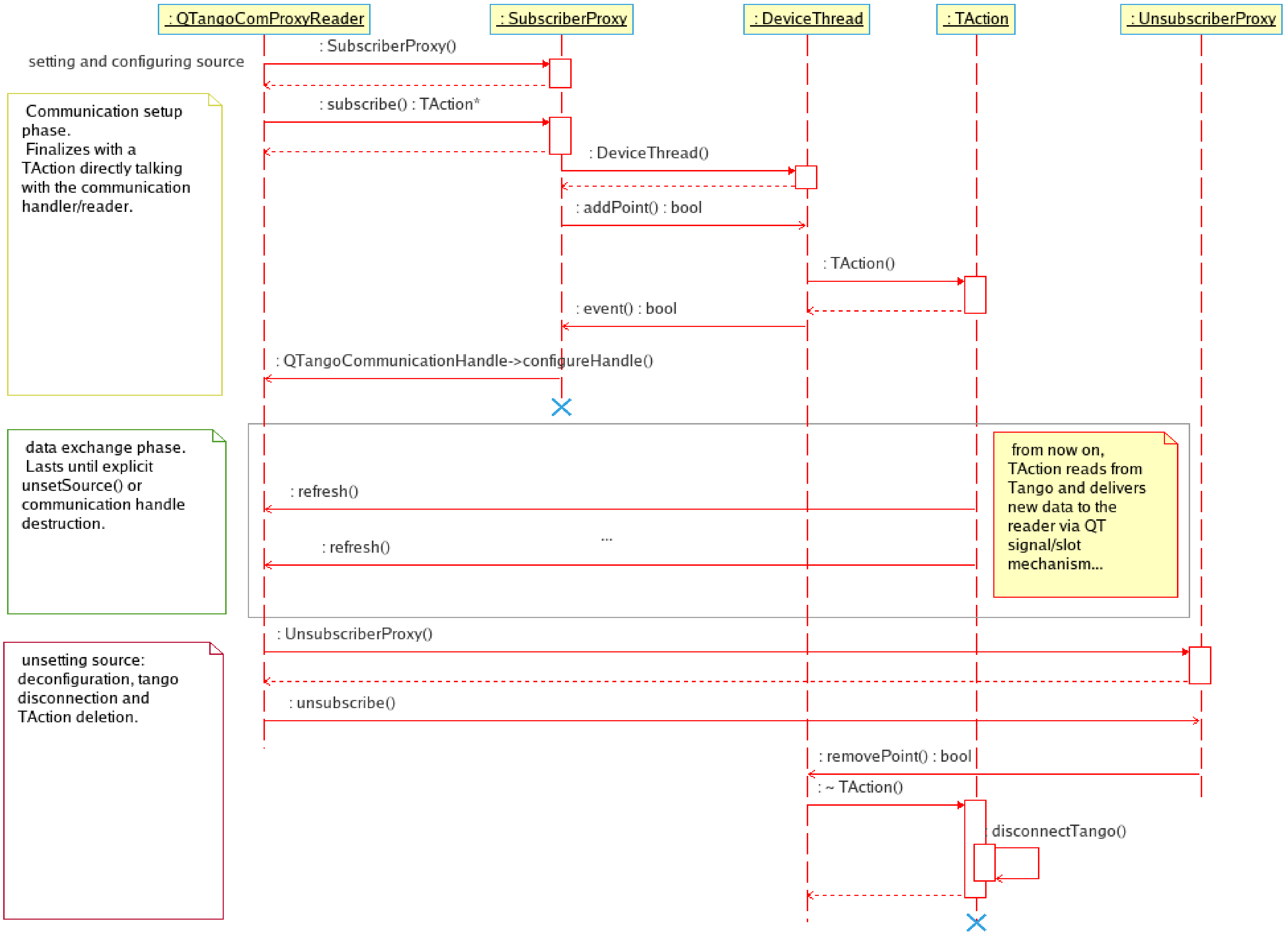
Overview (II)


• simple, multi threaded interface
• manages exceptions
• abstract handling of Tango data types
QTangoCore
QtCore
Tango
• signals/slots
• events
• threads
• read attributes
• write attributes
• commands
• attribute properties

QtangoCore class Diagram with two client widgets



QTangoCore objects lifetime sequence diagram



Communication setup phase.
Finalizes with a TAction directly talking with the communication handler/reader.

data exchange phase.
Lasts until explicit unsetSource() or communication handle destruction.

from now on, TAction reads from Tango and delivers new data to the reader via QT signal/slot mechanism...

unsetting source: deconfiguration, tango disconnection and TAction deletion.

QTangoCore implementation

- Only one thread per device;
- *TActions* shared among readers with the same source;
- *TActions* living in the *Device Thread* and so, as it was in *Qtango2*, managing *tango* data transfer outside the main application thread;
- QTango 3 TActions allow obtaining the return values from the commands.

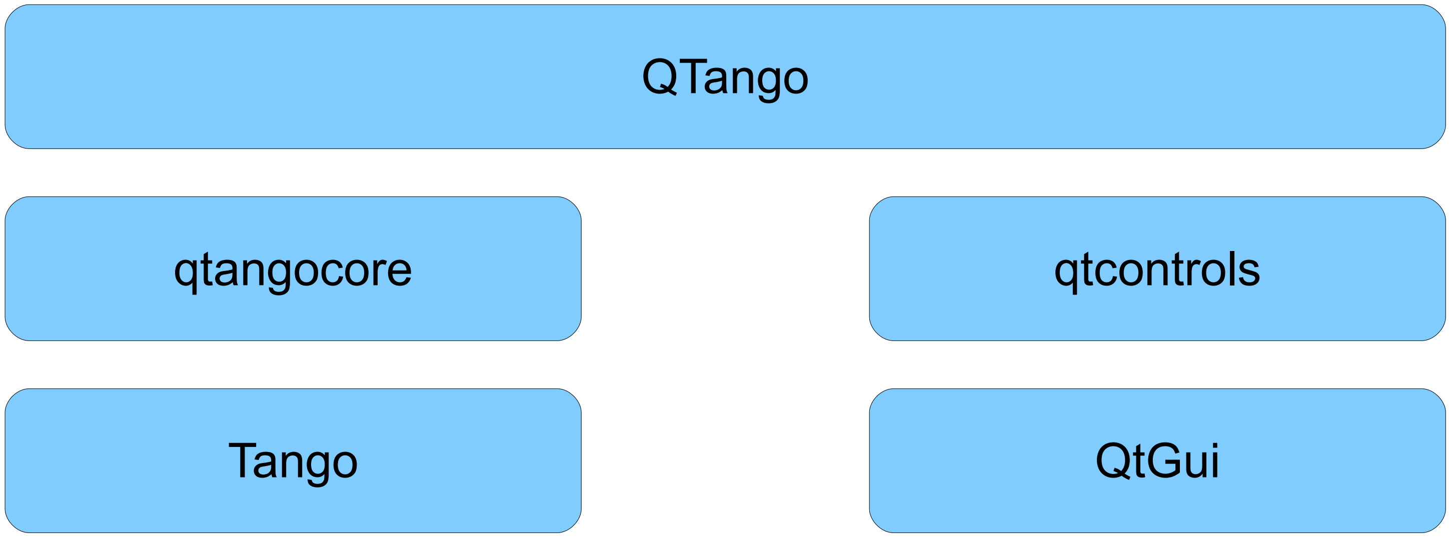
Part II

QTango

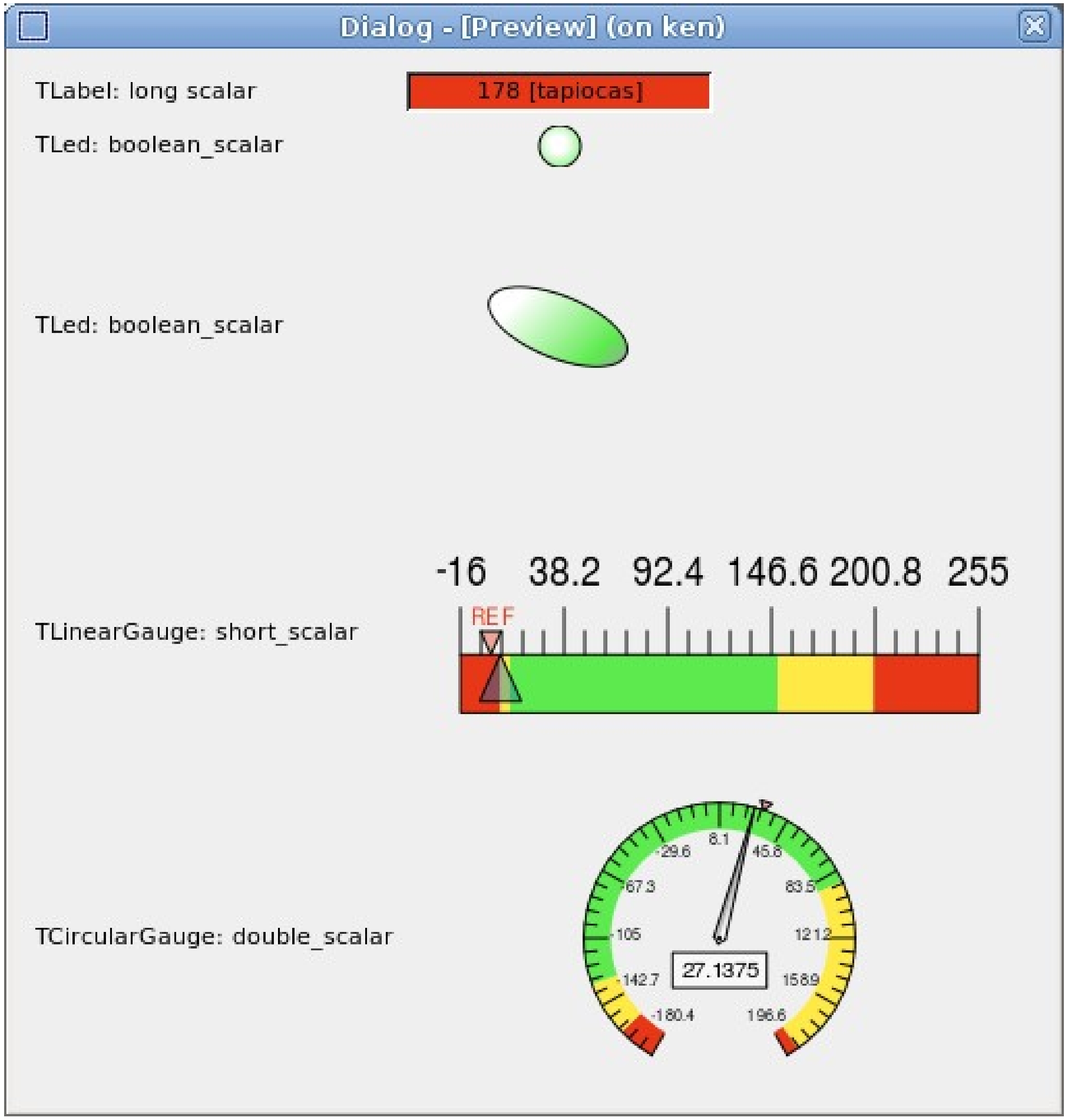
a set of Qt widgets integrated

with QTangoCore

QTango 3 infrastructure



Readers


Dialog - [Preview] (on ken)

TLabel: long scalar
178 [tapiocas]

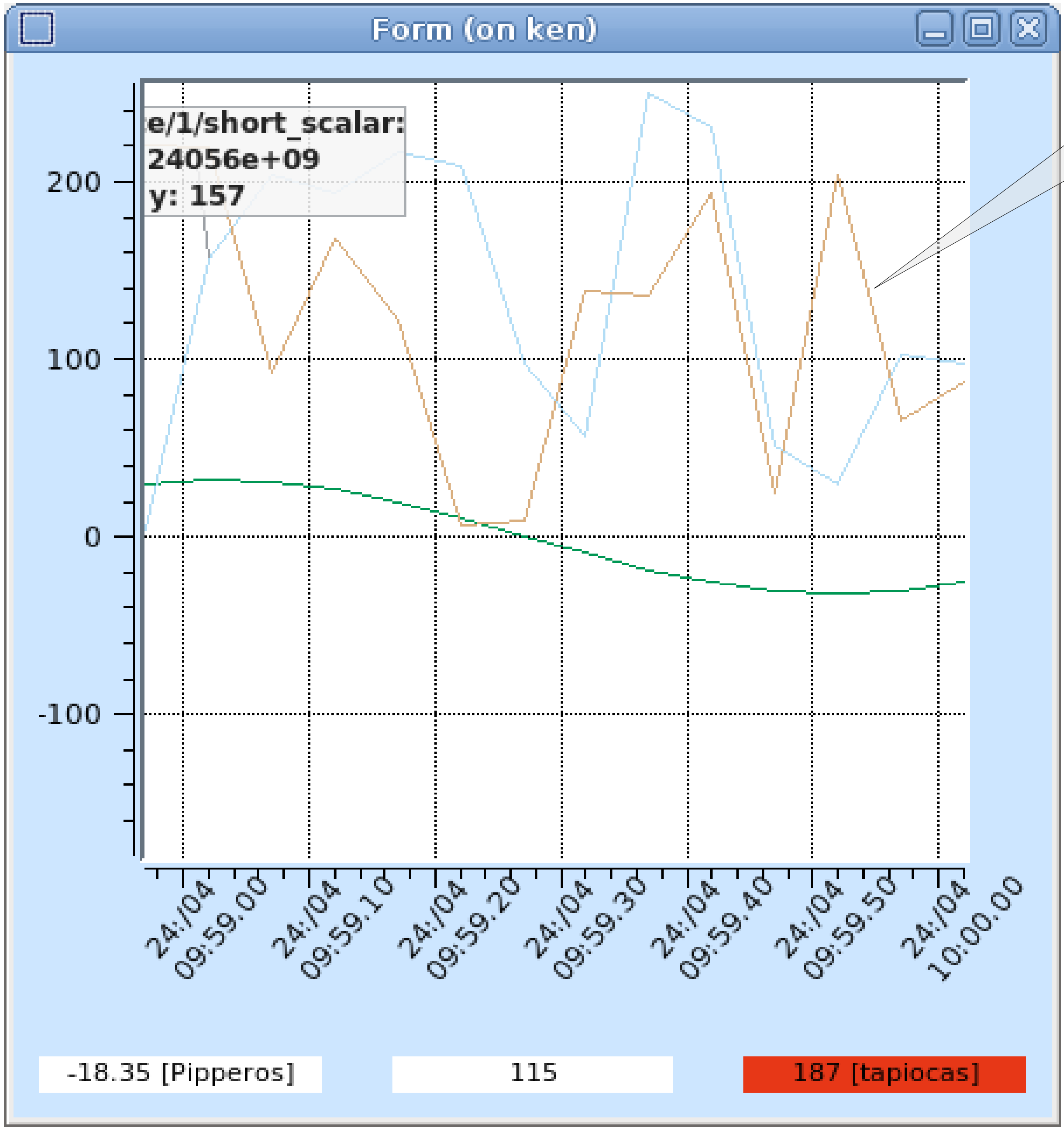
TLed: boolean_scalar

TLed: boolean_scalar

TLinearGauge: short_scalar

TCircularGauge: double_scalar

Readers (II)



-18.35 [Pipperos]   115   187 [tapiocas]

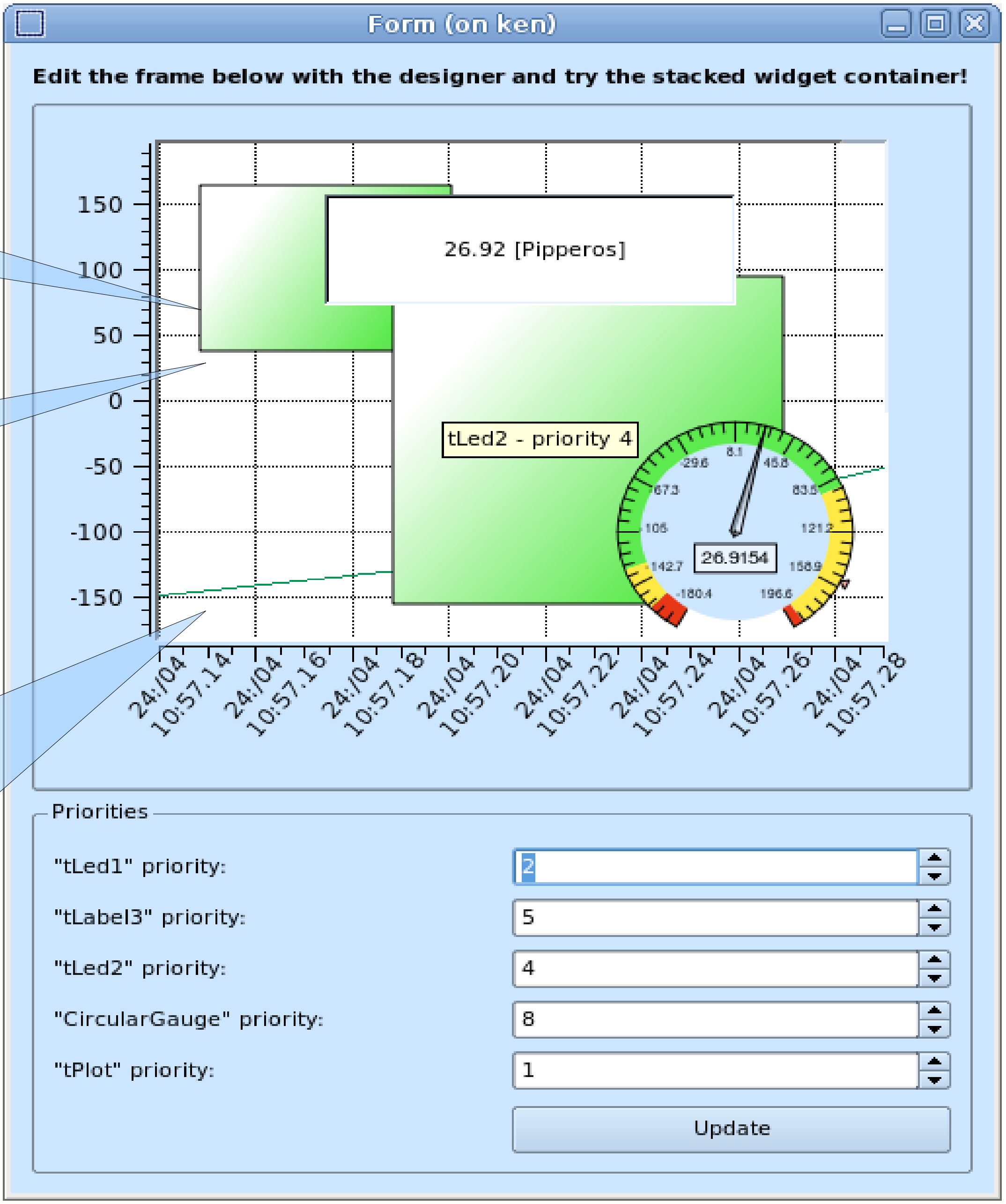
TPLotLight

Overlapping widgets

Overlapped widgets with a z axis defining their stack position

May be useful in synoptic design

<u>Use</u>:
- introduce an EstackedWidgetContainer in the designer
- place QTango widgets inside
- add each widget to the container with its `*z axis*` priority

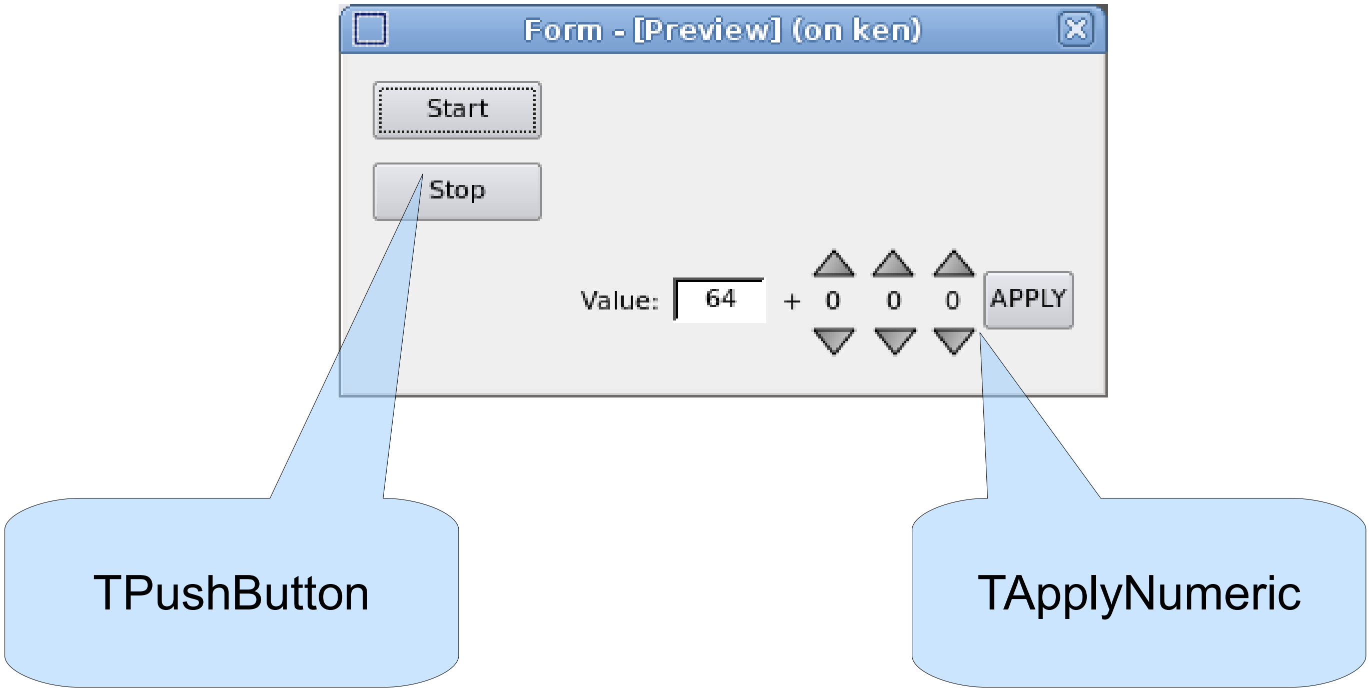


Edit the frame below with the designer and try the stacked widget container!

Priorities

"tLed1" priority:	2
"tLabel3" priority:	5
"tLed2" priority:	4
"CircularGauge" priority:	8
"tPlot" priority:	1

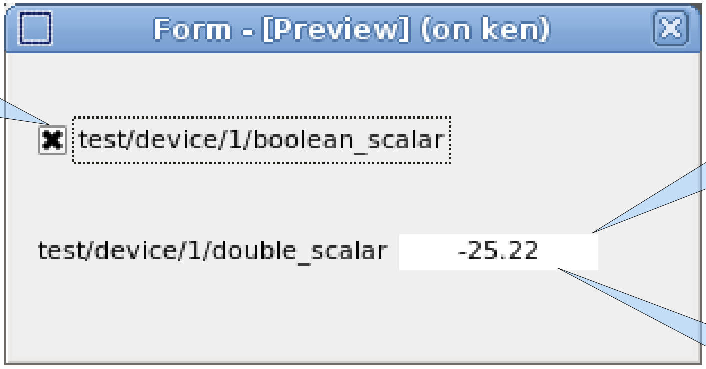
Writers


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

Readers *and* Writers

TCheckBox

TreaderWriter
- reads a value...

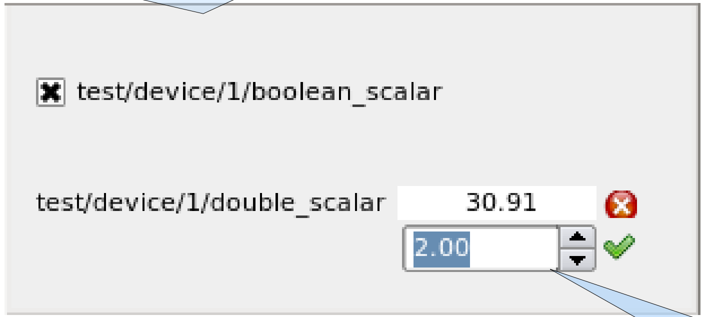

Form - [Preview] (on ken)

[✓] test/device/1/boolean_scalar

test/device/1/double_scalar -25.22

TreaderWriter
- move the mouse over...

TreaderWriter
- ideal for synoptics
- occupies the space of a label with a hidden writer

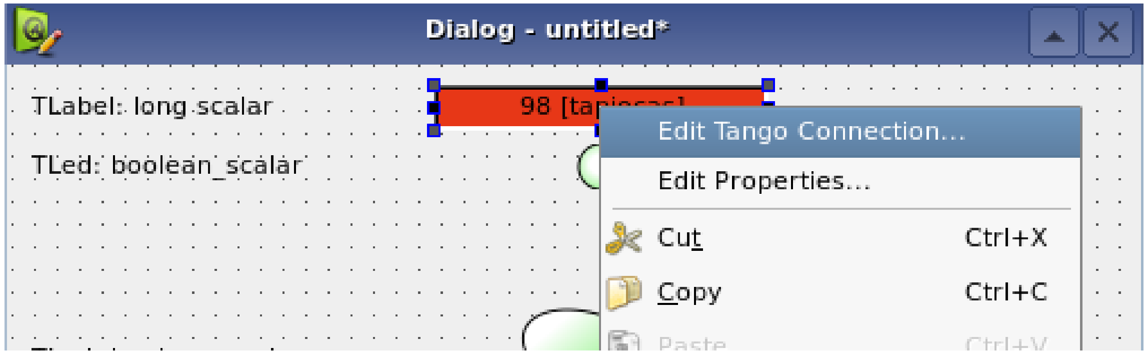

[✓] test/device/1/boolean_scalar

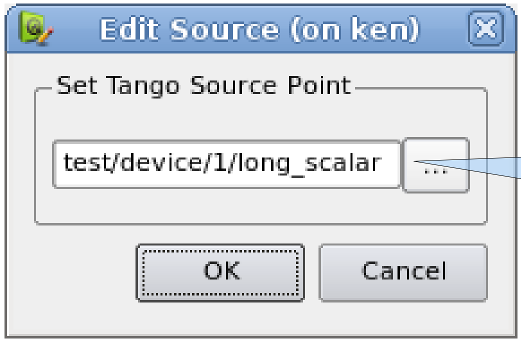
test/device/1/double_scalar 30.91
2.00

TreaderWriter
- a writer appears

Qt Designer integration

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





Edit Source dialog

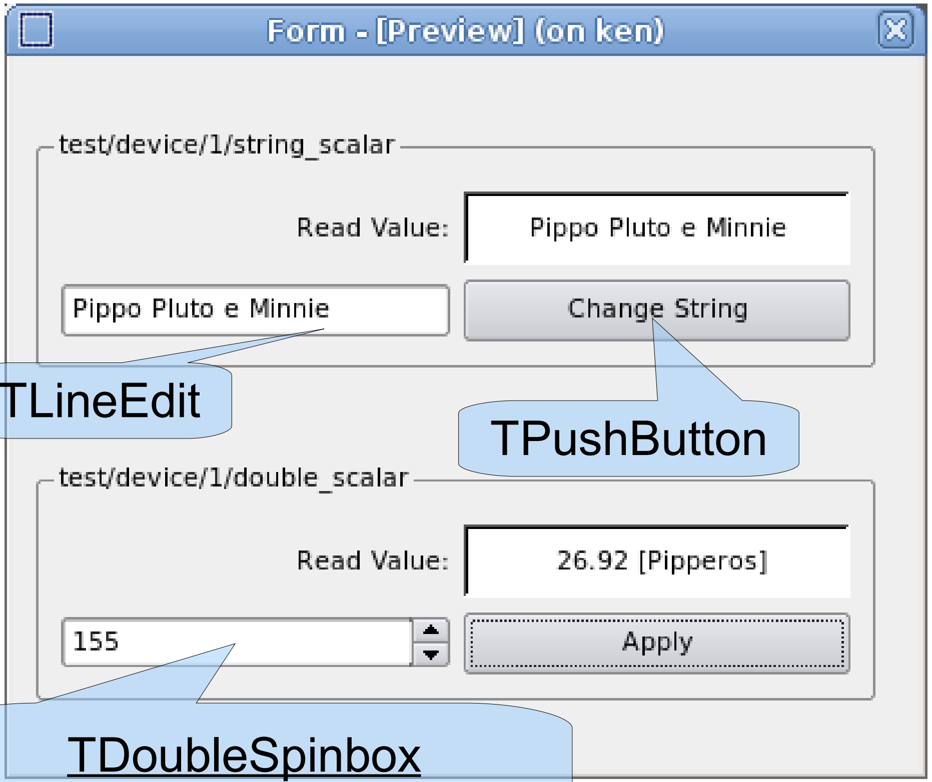
- test/device/instance/attribute_name
- test/device/instance->command_name(argin)

Design with SimpleDataProxy

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

Edit Targets Dialog

- test/device/1/double_scalar(&simpleDataProxyObjectName)



test/device/1/string_scalar

Read Value: Pippo Pluto e Minnie

Pippo Pluto e Minnie

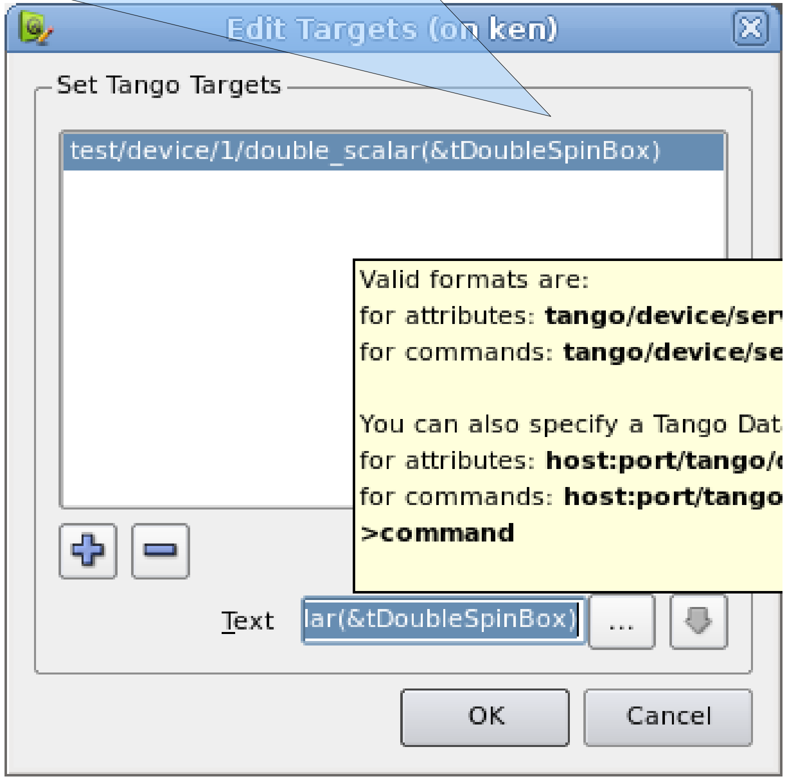
test/device/1/double_scalar

Read Value: 26.92 [Pipperos]

155

TDoubleSpinbox

- *with name "tDoubleSpinBox"*



test/device/1/double_scalar(&tDoubleSpinBox)

Valid formats are:
for attributes: **tango/device/ser**
for commands: **tango/device/se**

You can also specify a Tango Dat
for attributes: **host:port/tango/**
for commands: **host:port/tango**
>command

Text lar(&tDoubleSpinBox)

Part III

Programming with QtangoCore

Create a widget reading from and writing to a *Tango* device server

Reader

- Readers must inherit from QTangoComProxyReader
- readers must implement the *pure virtual* method *refresh*()
- the *refresh*() method has a <u>TVariant</u> as argument. It contains the data read from the *Tango* layer.
- *connect() reader's qTangoCommunicationHandle newData() signal to the refresh() slot*

Reader: TVariant

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;

Reader: 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;
...	

Reader: refresh()

- From the 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

Reader: attribute 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 call **setAutoConfiguration(true)** inside your reader – which inherits QtangoComProxyReader;

• 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*

Reader: attribute auto 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()

Example: reader implementation

The reader will be able to:

• *read an attribute*;

• disable readings when hidden;

• *auto configure* itself to notify warning and alarm values;

• have a *helper application* associated, started by the right mouse button click.

Example: reader implementation (II)

```
#include <com_proxy_reader.h>
#include <QLineEdit>

class MyReader : public QLineEdit, public QTangoComProxyReader
{
Q_OBJECT
        MyReader(QWidget *parent, Qt::WFlags f = 0);

protected slots:
        void refresh(const TVariant &);
        void init(const TangoConfigurationParameters *);

protected:
        void hideEvent(QHideEvent*);
        void showEvent(QShowEvent*);
        void mousePressEvent(QMouseEvent *e);

private: /* some variables for auto configuration... */
        double d_maxvalue, d_minvalue, d_minwarn, d_maxwarn;
        double d_minerr, d_maxerr;
        QString d_measurementUnit;
};
```

compulsory!

Auto configuration!

Example: how to write a reader (III)

The constructor

```
MyReader::MyReader(QWidget *parent, Qt::WFlags) :
      QLineEdit(parent),
      QTangoComProxyReader(this)
{
      setText("No Link");
      setHelperApplicationEnabled(true);
      connect(qtangoComHandle(), SIGNAL(newData(const
        TVariant&)), this, SLOT(refresh(const TVariant&)));

      connect(qtangoComHandle(),
            SIGNAL(attributeAutoConfigured(const
                  TangoConfigurationParameters *)),
                  this,
                  SLOT(init(const TangoConfigurationParameters *)));
      setAutoConfiguration(true);
}
```

Example: how to write a reader (IV)

The refresh() implementation

```
void MyReader::refresh(const Tvariant& v)
{
    switch(v.quality())
    {
        case ATTR_INVALID: /* … */
            break;
        case ATTR_VALID: /* … */
            break;
    }
    if(v.canConvertToDouble())
        setText(QString("%1 [%2]").arg(v.toDouble().
            arg(d_measurementUnit));
}
```

available through
auto configuration

Example: how to write a reader (V)

Helper application, show/hide events

```
void MyReader::hideEvent(QHideEvent *e)
{
      QTangoComProxyReader::hideEvent();
      QLineEdit::hideEvent(e);
}

void MyReader::showEvent(QShowEvent *e)
{
      QTangoComProxyReader::showEvent();
      QLineEdit::showEvent(e);
}

void MyReader::mousePressEvent(QMouseEvent *ev)
{
  QTangoComProxyReader::mousePressEvent(ev);
  QLineEdit::mousePressEvent(ev);
}
```

Example: reader implementation (VI)

Auto configuration

```
void MyReader::init(const TangoConfigurationParameters *cp)
{
    if(cp->maxIsSet())
      d_maxval = cp->maxValue();
    if(cp->minIsSet())
      d_minval = cp->minValue();
    if(cp->MWarnIsSet())
      d_maxwarn = cp->maxWarning();
    if(cp->mWarnIsSet())
      d_minwarn = cp->minWarning();
    if(cp->MErrIsSet())
      d_maxerr = cp->maxError();
    if(cp->mErrIsSet())
      d_minerr = cp->minError();

    d_measurementUnit = cp->displayUnit();
}
```

Example: reader implementation (VII)

Done!

- create your new reader,
- give it an object name and
- set source on it!

Writer

inherits **QTangoComProxyWriter**

- auto configuration available through *handle*'s **attributeAutoConfigured(const TangoConfigurationParameters *)**
- write execution available through *proxy writer*'s **execute()** method

Exercise: writer implementation

```
class MySpinBox : public QSpinBox,
    public QtangoComProxyWriter
{
    Q_OBJECT

    public:
        MySpinBox(QWidget *); /* constructor */

    protected slots:
      /* this is for auto configuration: put limits on the spin box */
      void configure(const TangoConfigurationParameters * );

     /* when changing the value on the spin box, write attribute */
     void myValueChanged(int);
};
```

Exercise: writer implementation (II)

```
MySpinBox::MySpinBox(QWidget *parent) :
    QspinBox(parent),
    QtangoComProxyWriter(this)
{
    connect(qtangoComHandle(),
        SIGNAL(attributeAutoConfigured(
        const TangoConfigurationParameters *)), this, SLOT
        (configure(const TangoConfigurationParameters *)));

    connect(this, SIGNAL(valueChanged(int)), this,
        SLOT(myValueChanged(int)));
}
```

Example: writer implementation (III)

```
void MySpinBox::MyValueChanged(int v)
{
    /* incapsulate v into a QVariant to pass to the
     * writer's execute() method
     */
    QList<TVariant> tl = execute(QVariant(v));

    /* do whatever you like with the list of TVariant */
}
```

Example: writer implementation (IV)

```
void MySpinBox::configure(const
        TangoConfigurationParameters * cp))
{
    /* attribute must be configured into the database
     * with its minimum and maximum values.
     */
    if(cp->maxIsSet() && cp->minIsSet())
    {
        setMinimum(cp->minValue());
        setMaximum(cp->maxValue());
    }
}
```

Example: writer implementation (V)

Done!
Now use your new writer

- instantiate your new writer,
- give it an object name and
- set target on it!

Simple Data Proxy

- provides **input arguments** for your **writers**;
- any QWidget displaying a value can be used to implement a simple data proxy:
    - QLabel
    - QSpinBox
    - QDoubleSpinBox
    - QTextEdit/QTextBrowser
    - QComboBox
    - QLineEdit
    - …

Simple Data Proxy (II)

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

Optimization

- Widget refresh is triggered by an external clock:
  - all widget refreshed at once
- global refresh trigger can be disabled:
  - ✗ globally;
  - ✗ *per* reader
  - ✗ *little cpu overhead if many widgets refreshing independently*

Part IV

Writing *QTango* - ready Tango servers

- Correctly shape the *Tango* server paying special attention to **command** and **attribute** modelling;
  - commands only when suitable to the device model;
  - please no commands with strings as *argin* and/or *argout*;
  - put logic on the server rather than in the panel, as much as possible;
  - consult a QTango “*expert* “ when in doubt ;-)

Documentation

- QTangoCore is *html*-documented
  - http://hokuto.elettra.trieste.it/documentation/qtangocore/doc/html/index.html
- QTango widgets are *html*-documented
  - http://hokuto.elettra.trieste.it/documentation/qtango/doc/html/index.html
- This presentation
  - http://hokuto.elettra.trieste.it/documentation/qtangocore/doc/QTangoCorePresentation.odt

Logging and bug reporting

- QTangoCore provides console logging via coloured messages:

* ***error*** *message*

* ***warning*** *message*

* ***ok*** *message*

Disable them exporting ***QTANGO_NOPRINT="yes"*** *on the terminal*

Logging and bug reporting (II)

- *Report bugs via Bugzilla*
- *http://ken.elettra.trieste.it/bugzilla/*
- provide full debug output from QTangoCore console messages
- if possible, provide steps to reproduce the problem

The End

- **Thanks for your attention**

mailto: giacomo.strangolino@elettra.trieste.it