CUMBIA: A NEW LIBRARY FOR MULTI-THREADED APPLICATION DESIGN AND IMPLEMENTATION

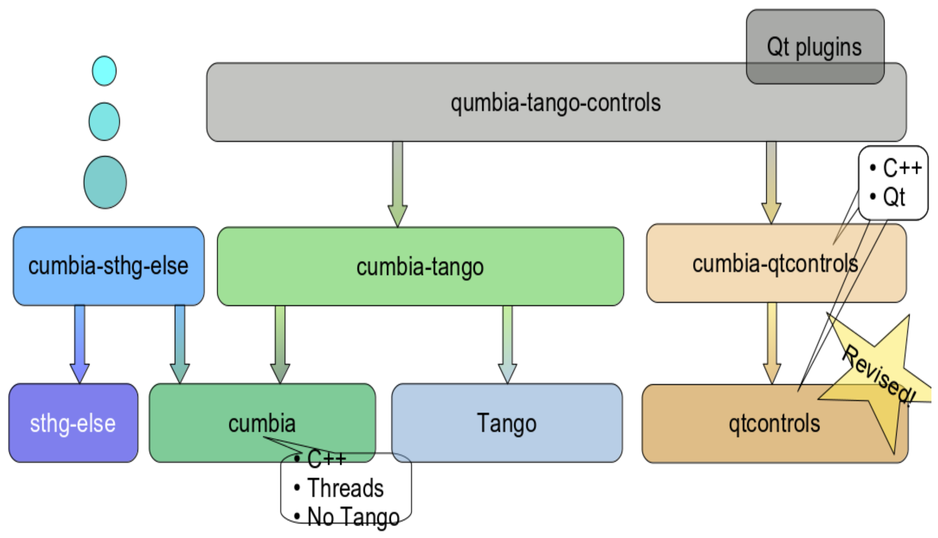
Giacomo Strangolino, Elettra, Trieste, Italy

Cumbia is a new library that offers a carefree approach to multi-threaded application design and implementation. It can be seen as the evolution of the *QTango* library. It offers a more flexible and object oriented multi-threaded programming style. Less concern about locking techniques and synchronization, and well defined design patterns stand for more focus on the work to be performed inside *cumbia activities* and reliable and reusable software as a result. The user writes *activities* and decides when their instances are started and to which thread they belong. A token is used to register an *activity*, and activities with the same token are run in the same thread. Computed results can be forwarded to the main execution thread, where a GUI can be updated. In conjunction with the *cumbia-tango* module, this framework serves the developer willing to connect an application to the TANGO control system. The integration is possible both on the client and the server side. An example of a TANGO device using *cumbia* to do work in background has already been developed, as well as simple QT graphical clients relying on the framework.

CUMBIA MODULES

Cumbia is a set of distinct modules; from lower to higher level:

- **cumbia**: defines the *Activities*, the multi thread implementation and the format of the data exchanged between them;
- **cumbia-tango**: integrates *cumbia* with the TANGO control system framework, providing specialised *Activities* to read, write attributes and impart commands;
- **cumbia-epics**: integrates *cumbia* with the EPICS control system framework. Currently, only variable monitoring is implemented;
- **cumbia-qtcontrols**: offers a set of QT control widgets to build graphical user interfaces inspired by the QTango's *qtcontrols* components. The module is aware of the *cumbia* data structures though not linked to any specific engine such as *cumbia-tango* or *cumbia-epics*.
- **qumbia-tango-controls**: written in QT, is the layer that sticks *cumbia-tango* together with *cumbia-qtcontrols*;
- **qumbia-epics-controls**: written in QT, the component pairs *cumbia-epics* to *cumbia-qtcontrols*.
- **qumbia-apps**: a set of applications written in QT that provide elementary tools to read and write values to the TANGO and EPICS control systems.

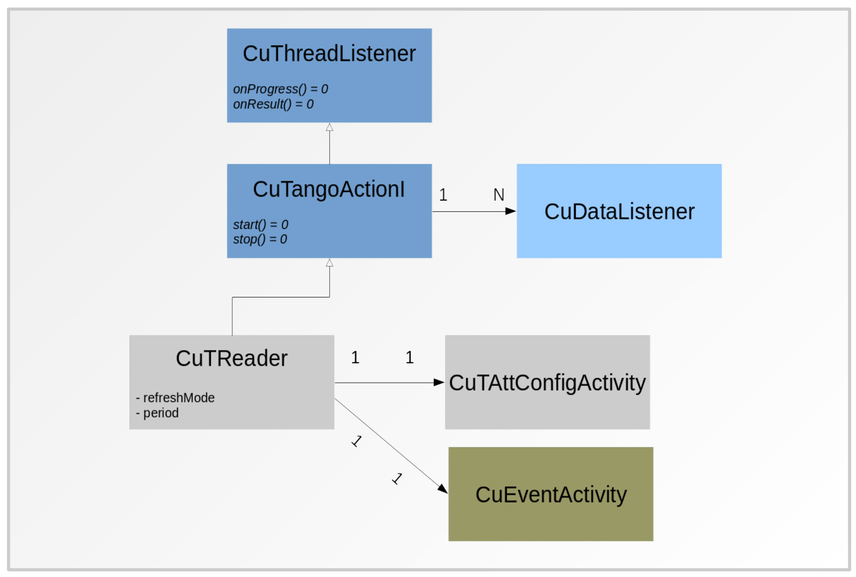


ACTIVITIES

CuActivity is an interface to let subclasses do work within three specific methods: *init*, *execute* and *onExit*. Therein, the code is run in a separate thread. The *publishProgress* and *publishResult* methods hand data to the main thread. To accomplish all this, an *event loop* must be running.

A *token* can be used to group several activities by a smaller number of threads. Activities with the same token run in the same thread.

Data transfer is realised with the aid of the *CuData* and *CuVariant* classes. The former is a bundle pairing keys to values. The latter memorises data and implements several methods to store, extract and convert it to different types and formats.



CUMBIA-TANGO

The *CumbiaTango* extends *Cumbia* base and manages the so called *actions*. An *action* represents a task associated to either a TANGO device attribute or a command (called *source*). Read, write, configure are the main sort of jobs an action can accomplish. *CuTangoActionI* defines the interface of an action. Operations include adding or removing data listeners, starting and stopping an action, sending and getting data to and from the underlying thread (for example retrieve or change the polling period of a *source*).

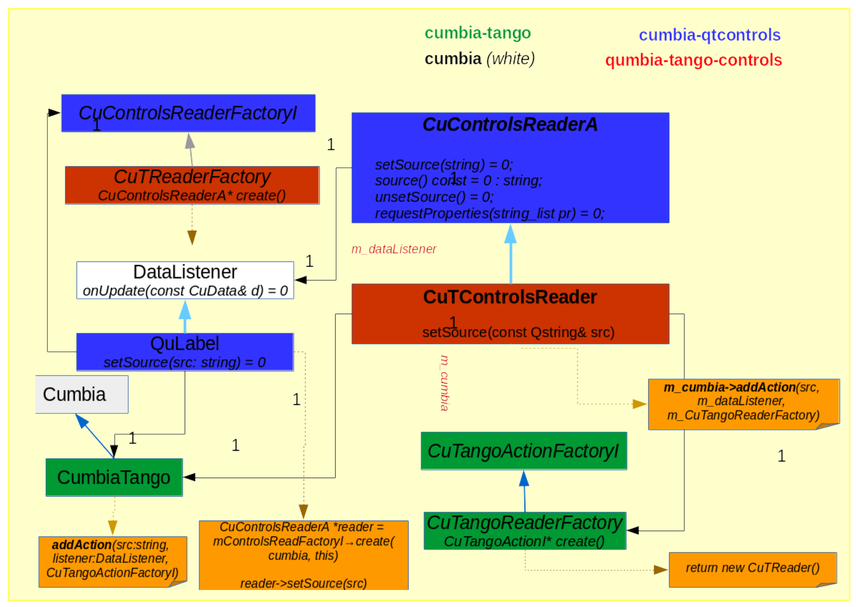
CuTReader implements *CuTangoActionI* and holds a reference to either an *activity* intended to receive events from TANGO or another one designed to poll a source.

Activities is where the TANGO connection is setup, database is accessed for configuration, events are subscribed, a poller is started or a write operation is performed. This is done inside the thread safe *init*, *execute* and *onExit* methods, invoked from another thread. Progress and results are forwarded by the *publishProgress* and *publishResult* methods in the activity and received in the *onProgress* and *onResult* implemented by the *action*. Therein, *CuDataListener*'s *onUpdate* method is invoked with the new data. Reception safely occurs in the main thread. *cumbia-tango* groups threads by TANGO device name.

CUMBIA-QTCONTROLS

Combining *cumbia* and the *QT* cross platform software framework, it offers graphical control system components. *Labels*, *gauges* and advanced *graphs* are supplied, as well as *buttons* and *boxes* to set values. Elementary data representation is provided: the components are unaware of the *cumbia* engine lying beneath. In order to display real data on the controls, you have to combine different building blocks at the moment of setting up each reader or writer.

CuControlsReaderA and *CuControlsWriterA* define an interface to readers and writers. They have to provide methods to set and remove *sources* and *targets* of execution, as well as means to send and receive messages to and from *actions*. They hold references to the currently active *Cumbia* and data listener instances.



CUMBIA-APPS

qumbia-apps module provides a set of base applications to perform elementary actions on *sources*, such as readings and writings. The *generic_client* tool is a graphical panel able to read and write from both TANGO and EPICS, using labels and plots to show the trend over time or the present values, if the format is a vector. The screenshot on the right represents the *generic_client* reading a TANGO scalar attribute, a TANGO spectrum attribute and an EPICS *analog input*.

QUMBIA-TANGO-CONTROLS

Written in *QT*, q*umbia-tango-controls* component combines *cumbia-tango* with *cumbia-qtcontrols*. *CuTControlsReader* and *CuTControlsWriter* are the implementors of the previously discussed *CuControlsReaderA* and *CuControlsWriterA* abstract classes. Their *sources* and *targets* are TANGO attribute and command names, written with the same syntax as that adopted by *QTango*. They operate on a *CumbiaTango* instance (*Cumbia*'s subclass), which is in charge of creating and registering *actions*, finding *actions* already in use and managing installation and removal of *CuDataListener*s.

WHY REPLACE QTANGO

- Lots of features not required daily
- Some useful features not easy to implement (*e.g.* multiple serialized readings)
- Tightly bound to *Tango*
- Has been stable for years, the architecture is somehow complicated
- Code is not modular nor reusable enough

Giacomo Strangolino, Elettra, Trieste, Italy
E-mail: giacomo.strangolino@elettra.eu