ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPR01

CUMBIA: GRAPHICAL LIBRARIES AND FORMULA PLUGIN TO COMBINE AND DISPLAY DATA FROM TANGO, EPICS AND MORE

G. Strangolino, Elettra-Sincrotrone Trieste S.C.p.A., Basovizza, Trieste, Italy

Abstract

Cumbia libraries offer the next generation core (C++) and graphical (Qt) software to write complete and lightweight applications that provide a unified user interface, regardless of the underlying engine (Tango, EPICS, WebSocket, ...) With the new formula plugin, results can be manipulated and combined by JavaScript functions and displayed in the appropriate widget. Qt has a deep JavaScript integration that allows efficient introduction of program logic into the application. Using the Qt + QML technologies, apps can be designed for the desktop and mobile devices. Switching between the two targets is an immediate operation. A WebSocket based service has been used to test Qt + QML mobile applications on portable devices. It makes it possible to connect to Tango and EPICS without their installation. A new tool called *la-cumparsita* lets non-programmers use the Qt designer to realize complete applications ready to communicate with the control system in use: Tango, EPICS or any other abstraction framework (e.g. WebSocket). These apps seamlessly integrate with the desktop. Most demanding users can integrate JavaScript functions and use them as data sources for the GUI elements.

STRUCTURE OF THE FRAMEWORK

The *cumbia* library is made up of several modules. The core and the engine specific ones are written in pure C++, while those providing graphical elements employ the *Qt* framework [1]. Figure 1 outlines the relationship between the main modules that compose the software.

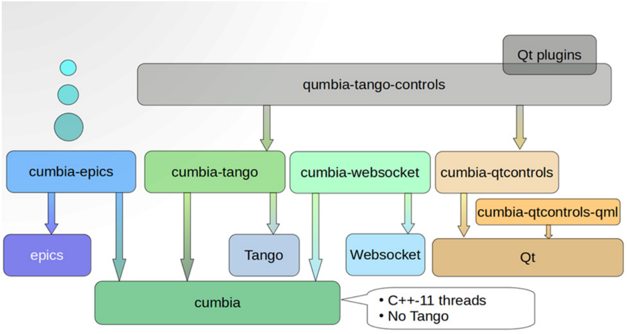


Figure 1: Relationship between *cumbia* main modules.

The next paragraphs describe each component in more detail.

MODULES

Cumbia Base Module

Cumbia is a component that offers a carefree approach to multi thread application design and implementation. 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 those with identical tokens are run in the same thread. Work is done inside the *init*, *execute* and *exit* methods. The library guarantees that they are always called in the activity thread. From within *init*, *execute* and *exit*, computed results can be forwarded to the main execution thread, where they can be used to update a graphical interface. Data is exchanged by means of a dedicated key/value bundle, named *CuData*.

Cumbia-tango

Cumbia-tango integrates *cumbia* with the Tango [2] control system framework, providing specialised *activities* to read, write attributes and impart commands. Readings are accomplished through either a poller or the Tango event system, for those attributes suitably configured. Write operations are always executed in an asynchronous thread and the result is delivered later in the main thread. *Cumbia activities* are employed by the module to setup the connection, access the database, subscribe to events or carry out periodic readings. Progress and result events are delivered to the main thread from the background *activity*. As stated in the previous section, activities identified by the same *token* belong to the same thread. Here, the *token* is the Tango device name. Applications that connect to the Tango control system will typically instantiate a *CumbiaTango* object that defines which kind of threads will be used (e.g. *Qt*'s for graphical interfaces) and thereafter parametrizes each reader or writer. Several modern design patterns have been exploited to provide a flexible and scalable architecture. Singletons have been completely replaced by *service providers* in order to offer services For graphical applications. The component provides helpful classes that can be used from outside an activity to access devices, fetch database properties or interpret exceptions raised from within the engine. Aside from these utilities, one would not normally employ this module directly. *Cumbia-qtcontrols* and *qumbia-tango-controls* is where to look for when the integration between the control system and the user interface is the objective.

Cumbia-epics

Cumbia-epics integrates the Experimental Physics and Industrial Control System (EPICS) [3] control system with *cumbia*. The interaction with the lower level *cumbia* base component and the interface offered to clients is equivalent to the *cumbia-tango*'s. Configuration, monitor and *put* operations are currently implemented. Data is exchanged through the same aforementioned key/value structure (*CuData*). Differences between the EPICS and Tango engines are concealed and utmost effort has been taken to unify the representation of the results. For example, Tango *Max value* database attribute property and EPICS *upper_disp_limit* from *dbr_ctrl* data are both stored into the

CuData value associated to the *max* key. *Cumbia-tango* and *cumbia-epics* clients are thus enabled to represent data in a way that is independent of the source. Further extensions to the *cumbia* framework operating on additional engines should commit to this sort of *contract* pledging homogeneous data representation across diverse control systems or software architectures (*cumbia-websocket* is another example). Table 1 describes some relevant keys with their data type stored by a typical *CuData* carrying a result.

Table 1: Example of *CuData* Contents

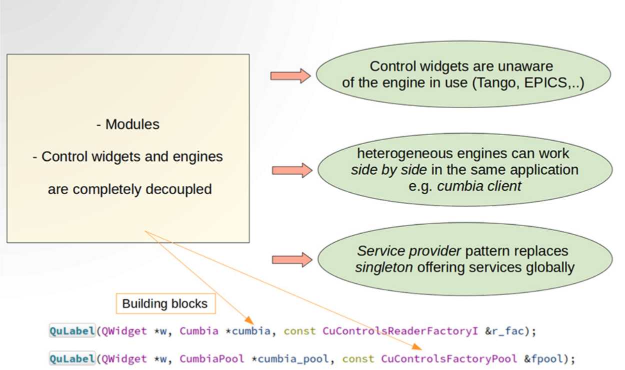
KEY	TYPE	VALUE	DESCRIPTION
type	string	“property”	Identifies a configuration content
src	string	-	The source name as configured with *setSource*
value	CuVariant	-	The value read by the engine
display_unit	String	-	The unit for displayed data
min	String	-	Minimum value (convert with *toDouble*)
max	String	-	Maximum value (convert with *toDouble*)
err	Bool	true|false	True if an error occurs
Msg	String	-	Operation/error message
timestamp_ms	time_t + suseconds_t	-	Timestamp from *struct_timeval*: tiv.tv_sec * 1000 + tiv.tv_usec / 1000; Convert with *toLongInt()*

Cumbia-qtcontrols Module

This module combines *cumbia* with the *Qt* cross platform software framework, offering graphical control system components. Labels, gauges, thermometers and advanced graphs are supplied, as well as buttons, spinners combo and text boxes to set values. Components are unaware of the engine to which they are connected. In order to display real data on the controls, different building blocks must be combined when they are set up. When data is available from the background (i.e. from the control system), it is delivered to the component in the main application thread. Control elements need to implement the *CuDataListener* interface. Figure 2 represents some of the aspects hitherto described. Readers and writers must adhere to interfaces that declare how to set and remove sources and targets of execution, as well as how to send and receive messages to and from the background *activities*. A report on the health of the link between the object and the control system is available through the context menu. A dialog shows information concerning the application and author, errors and connection statistics. It is possible to start a fresh live reader, inspect received data structures (*CuData*) and see a graph of the value over time for scalar data types. In case of malfunction, error messages are reported as well.

Qumbia-tango-controls

Qumbia-tango-controls is written in *Qt* so as to blend the *cumbia-qtcontrols* and the *cumbia-tango* modules together. It provides a higher level interface to use graphical elements and *QObjects* from the *Qt* library and link them to the tango control system. Factories instantiate Tango readers and writers. They represent the second building block used to instantiate engine-independent objects. The first is *CumbiaTango,* mentioned in the namesake section.


- Modules
- Control widgets and engines
are completely decoupled
Control widgets are unaware of the engine in use (Tango, EPICS,..)
heterogeneous engines can work *side by side* in the same application e.g. *cumbia client*
Service provider pattern replaces *singleton* offering services globally
Building blocks

```
QuLabel(QWidget *w, Cumbia *cumbia, const CuControlsReaderFactoryI &r_fac);
QuLabel(QWidget *w, CumbiaPool *cumbia_pool, const CuControlsFactoryPool &fpool);
```

Figure 2: Building blocks allow component decoupling.

Qumbia-epics-controls

Qumbia-epics-controls is the equivalent of the Tango counterpart described in the previous section. *Cumbia-qtcontrols* items represent data uniformly no matter what control system they are linked to. Figure 3 shows an application with mixed sources from Tango and EPICS. At the bottom of Figure 2 one can see how a *cumbia* object is generally instantiated: either by means of an engine specific *Cumbia* object and reader (writer) factory or through *Cumbia* and factory *pools*. Available control systems register to the *pools* and the *pools* at runtime guess which one each source belongs to, according to characteristic name patterns.

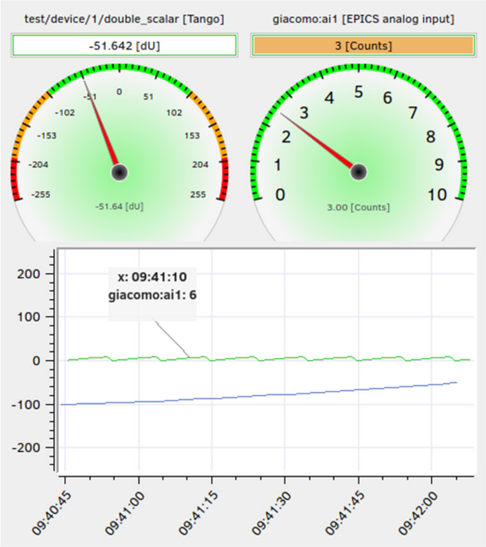


Figure 3: Tango and EPICS sources displayed by a *cumbia* application.

The Cumbia-qtcontrols-qml Module

The *QML* module employs the modern *QtQuick/QML* technology as an alternative to the classic *Qt widgets.* Amongst the advantages, we mention faster development thanks to the declarative language and the integration with the *Qt creator* and smooth realization of mobile applications. *Qt* for *Android* enables one to run *Qt 5* applications on such platform and supports native *Android* style with *Qt Quick Controls*. *Cumbia-qtcontrols-qml* integrates with *cumbia* (Tango, EPICS, *Websocket*) and offers a set of elements already included in the *Qt creator* library: labels, circular gauges, trend and spectrum charts (the latter based on *QtCharts QML*). Since Tango and EPICS do not build natively on *Android*, the test applications connect to the control system through *cumbia-websocket*. The last mentioned component gives access to the control system through the *websocket* technology in combination with the *canoned* server developed within the PWMA project [4]. In Fig. 4 one can see a *Qt QML* application running on an *Android* device.

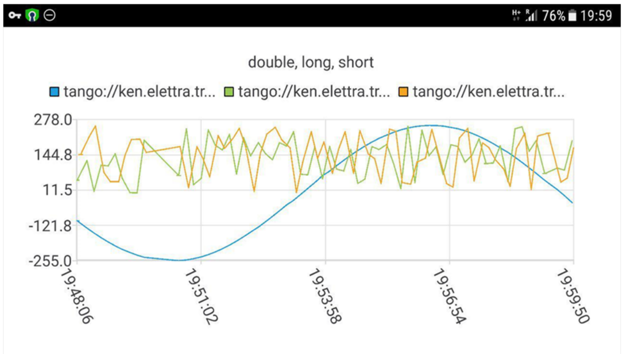


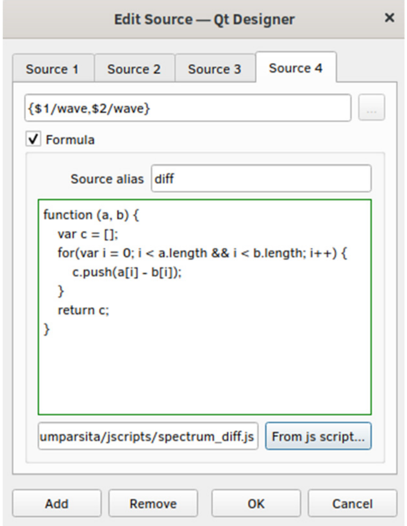
Figure 4. Three *Tango attributes* are read through the *websocket* interface provided by the *canoned* server (PWMA project).

PLUGINS

Cumbia library has been designed to be modular, fast and reliable. This objective is achieved more effectively if it is kept as small and basic as possible. Nonetheless, it can be expanded through plugins. New releases of the library will seldom introduce new features. Rather, they may introduce interfaces and loaders for additional plugins. A set of fundamental ones is included in the default *cumbia-libs* distribution available from *github.com*. They provide extensions to fetch properties from the Tango database, start helper applications, communicate through the *DBus* message bus, serialize reading of multiple sources, *Qt designer* integration, a set of context menu actions on the *cumbia-qtcontrols* widgets and support for formulas and JavaScript functions. More plugins can be downloaded from the ELETTRA *github* page [5]. The most relevant ones are discussed in the ensuing sections.

Formula Plugin

The formula plugin extends the base functionalities combining readings into formulas and functions. Sources of data can be written in the form of JavaScript functions rather then as simple variable names. Editing can be done from the *Qt designer* and the resulting application will understand formulas as soon as the plugin is loaded. Figure 5 shows the designer *Edit source* form and Figure 6 a spectrum plot representing two waveforms, their sum and difference.



{$1/wave,$2/wave}

☑ Formula

Source alias: diff

```
function (a, b) {
   var c = [];
   for(var i = 0; i < a.length && i < b.length; i++) {
      c.push(a[i] - b[i]);
   }
   return c;
}
```

umparsita/jscripts/spectrum_diff.js

Figure 5. *Qt designer Edit Source* form with a JavaScript *function*.

In the example above, the readings from the two sources in the brackets will replace the *a,b* input parameters to the JavaScript function. A third vector, named *c* in the function, is returned and used to provide data for the *Source 4*, that has been given the *diff* alias

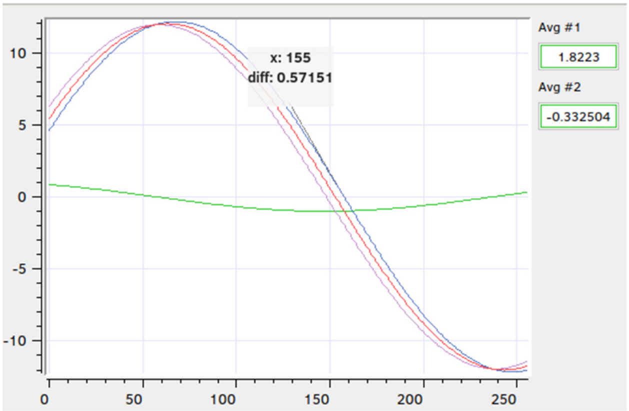


Figure 6. Two Tango waveforms, their sum and their difference.

Qt Designer Plugin

A *Qt designer* plugin lets the developer draw the graphical user interface and configure the sources and targets very quickly. The generated form can be either enriched by additional logic into the *C++* code or directly interpreted by the *la-cumparsita* application. The *QML* module integrates a library of elements directly into the *Qt creator*'s designer.

Extra Widgets Plugin

Custom widgets can be added to the basic set offered by the *cumbia* controls. One interface is defined for the plugin and one for the widgets. The latter is not a requisite because the *Qt property* system can be used to access methods and attributes (for example *source* and *target*). The *real time plot* is a graph that extends the base *cumbia spectrum plot* and adheres to a pattern that is common at ELETTRA: Tango devices expose commands with two input arguments that return arrays of data. This specific graph offers configuration options for such class of commands.

Extra widget plugins' name must match a given pattern, so that they can be easily identified and loaded. Each one offers a catalog of the components that are instantiated by one of the several flavours of the *create* method (both engine specific reader and writer allocation or *pool* factory approaches are available). When an application needs an object, it tries to find the supplier by class name through the extra widget plugin loader. On success, usage and access to properties are immediate.

APPLICATIONS

A set of utilities is included in the *cumbia* library distribution. They help the programmer develop new applications and migrate from QTango [6] projects. Additionally, they supply a generic client and a tool that allows setting up a control panel without using knowledge of coding.

Developing C++ Cumbia Applications

New software can be written in C++ with the same approach taken with QTango. A tool named *cumbia new project* can be used to create a skeleton project, that can be edited with *Qt creator* and *Qt designer.* The latter hosts a collection of base *cumbia* widgets that can connect to the available engines on the fly. To build the project, an instrument called *cumbia ui make* needs to be executed. When the *Qt* build system generates the *C++* code from the *ui* file created with the *designer*, default widget constructors are invoked. Since *cumbia* objects need to be parametrized at creation time, the *cumbia ui make* recognises and expands the constructors of the classes of the library as well as the custom ones that may have been added to the project. After a successful expansion, the workspace can be built. The command *cumbia new control* assists the programmer write a specialised reader or writer either for a specific project or as a supplementary component (e.g. part of the catalog of an extra widget plugin). Those familiar with QTango will immediately recognise that the two programming methodologies are much the same. For instance, the naming conventions for the sources and targets are identical.

Developing Codeless Cumbia Applications

La-cumparsita is a *ui* file interpreter. This means that simple GUIs can be composed with the *Qt designer* and then executed with the same look and feel and level of integration as any other *cumbia Qt* application. *La-cumparsita* supports the formula plugin, so that JavaScript functions can be mixed into elementary readers and writers. Figure 6 is actually a screenshot taken from *la-cumparsita* app.

Generic Client

The *cumbia client* command followed by a list of sources is a versatile tool to quickly connect, display and change field quantities. It works with most data types and constitutes an example of the seamless integration of distinct engines into one single application. Provide a mixed list of sources from distinct control systems (for instance Tango and EPICS) to experience this feature.

A Bot for the Telegram Messaging Application

As described in [4], Telegram is a cloud-based mobile and desktop messaging app focused on security and speed. It is available for Android, iPhone/iPad, Windows, macOS, Linux and as a web application. The *bot* is a *server* application that connects the control systems supported by cumbia to Telegram. One can read and monitor values, as well as receive alerts when something special happens. Simple source names or their combination into formulas can be sent to the bot. It replies and notifies results. It is simple, fast and intuitive. Refer to [7] for detailed information.

INSTALLATION AND UPDATES

The installation is automated by a shell script that guides throughout the whole process. After downloading the library from *github*[8], one must check the configuration in *scripts/config.sh* (to set the destination prefix and optional minor details) and finally execute *./scripts/cubuild.sh tango epics install* to build and copy the files into the system. Later on, the updates can be automatically accomplished with the *cumbia upgrade* command. It will prompt to choose the desired version, download it, rebuild and set it up automatically. Plugins may need to be manually rebuilt after a major version change. The documentation lists the prerequisites and dependencies and explains how to install every *cumbia* module by hand.

DOCUMENTATION

Special care has been taken in writing the documentation[9]. It is hosted by *github.io* and maintained in a dedicated *branch*, named *cumbia-libs-gh-pages*. Alongside class documentation, frequently asked questions and tutorial sections with code examples are available.

CONCLUSIONS

The *cumbia* libraries have been introduced into the ELETTRA control room workstations since last major Linux distribution upgrade. They stand side by side with QTango and the migration is going to be gradual. Early comparison tests between the core of the two frameworks show a very good performance of the new one. Figure 7 shows a graph of the CPU usage of two equivalent console applications performing readings of the same Tango attributes and commands.

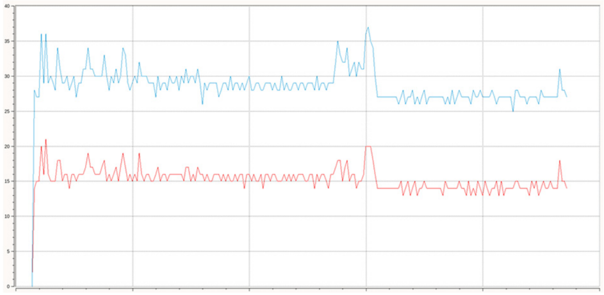


Figure 7. CPU usage over time of two equal command line applications, one relying on *cumbia* (red curve), the other on QTango (blue).

The design of the *cumbia* libraries focuses on lightness and simplicity, to promote extension and composition of its base elements to create more complex objects. Another central point is expanding the collection of items and functionalities through plugins. The design strategies will additionally ensure long life and flexible adaptation to increased demands in development and performance of control system applications. *La-cumparsita* is a tool that enables to realise graphical user interfaces without writing any code. Notwithstanding, JavaScript functions or the simple combination of results into formulas can broaden its field of application.

REFERENCES

[1] Qt cross platform software development for embedded & desktop, `https://www.qt.io/`

[2] Tango, `http://www.tango-controls.org`

[3] Experimental Physics and Industrial Control System, `https://epics.anl.gov/`

[4] L. Zambon, A. I. Bogani, S. Cleva, E. Coghetto, F. Lauro, and M. De Bernardi, "Web and multi-platform mobile app at ELETTRA", *in Proc. ICALEPCS'17,* Barcelona, Spain, October 2017, paper TUSH103.

[5] `https://github.com/ELETTRA-SincrotroneTrieste/`

[6] G. Strangolino, F. Asnicar, V. Forchì, and C. Scafuri, "QTango: a library for easy Tango based GUIs development", *in Proc. ICALEPCS'09,* Kobe, Japan, Oct. 2009, paper THP096. `https://github.com/ELETTRA-SincrotroneTrieste/qtango`

[7] G. Strangolino, "Cumbia-Telegram-Bot: Use Cumbia and Telegram to Read Monitor and Receive Alerts from the Control Systems", *in Proc. ICALEPCS'19,* New York, U.S.A, Oct. 2019, this conference, `https://github.com/ELETTRA-SincrotroneTrieste/cumbia-telegram`

[8] `https://github.com/ELETTRA-SincrotroneTrieste/cumbia-libs`

[9] `https://elettra-sincrotronetrieste.github.io/cumbia-libs/`