ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA148

CUMBIA-TELEGRAM-BOT: USE CUMBIA AND TELEGRAM TO READ, MONITOR AND RECEIVE ALERTS FROM THE CONTROL SYSTEMS

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

Abstract

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 user signs in the cumbia-telegram-bot to chat with a Tango or EPICS control system from everywhere. 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, intuitive. A phone number to register with telegram and a client are the necessary ingredients. On the server side, cumbia-telegram-bot provides the administrator with full control over the allocation of resources, the network load and the clients authorized to chat with the bot. Additionally, the access to the systems is read only. On the client side, the bot has been meticulously crafted to make interaction easy and fast: history, bookmarks and alias plugins pare texting down to the bone. Preferred and most frequent operations are accessible by simple taps on special command links. The bot relies on modules and plugins, that make the application extensible.

TECHNOLOGY

The server is written in C++and employs the Telegram BOT API[1], that is an HTTP-based interface for building *bots* for Telegram. Qt [1] is used to manage network connections, support JavaScript integration, store user data into a local database (*SQLite*) and draw graphs of spectrum variables (by means of the *Qwt* library). Figure 1 shows the whole infrastructure.

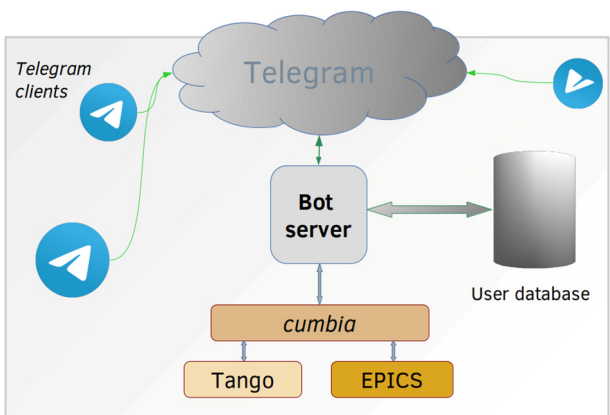

Figure 1: The whole *cumbia-telegram bot* architecture.

After joining the *bot*, clients must be authorized before messaging. Operations on the control systems are read only and may be subject to limitations (e.g. upper limit on active monitors) according to the account privileges that can be set up on a user basis. Further restrictions curb the access to the engines: the maximum read rate is established by the administrator and the bandwidth is divided across the connected users. Those monitoring more than one source in *polling* mode will face major restraint. On the other hand, event based updates are not limited at the moment, due to their theoretical low impact on the control system and network. The application is organised in modules. Some of them are built in, while additional ones can loaded as plugins, that register with the server, receive and process messages. Bookmark, search and command components are examples of plugins. The *modules* command returns a list of currently loaded modules and plugins, with their description.

FEATURES

Overcoming Messaging App Limitations

Telegram is an advanced cloud-based mobile and desktop messaging application. The design of the *bot* had to face the limitations of a text centred interface. Source names are not easy to remember and keyboard dictionaries cannot help writing often complex patterns. Frequent operations must be accessed easily, and handwriting must be replaced by *tap* operations as much as possible. *Tapping* is possible on commands either configured through a special bot named *The BotFather* or nested in the message (and identified by a leading / character, like */help*). Those manually configured with *The BotFather* are available through a button at the right of the message input text area.

The following components have been implemented to address these limits and make the user interaction effective, intuitive and funny:

- *history*, to keep a record of recent successful operations;
- shortcut to repeat the *last* action;
- *bookmarks*, where preferred commands can be saved and sent over again with a *tap*;
- *alias*. An alias can be associated to any sort of instruction, so that it can be quickly recalled.

Supported Engines

At the moment, the Tango [2] and EPICS [3] control systems are seamlessly integrated in the application. Since the *bot* is based on *cumbia*, more engines can be added with little effort.

[1] https://core.telegram.org/bots/api

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA148

Formulas

The application is equipped with the *cumbia* formula plugin to make its features available to the user:

- sources from any supported engine can be combined into formulas;
- simple operators can be employed in expressions as well as more complex JavaScript functions.

FUNCTIONS

Read

The most basic function of the *cumbia-telegram-bot* is to read values from the control systems. The reply to a message with the name of the source contains either its value or an error message, if the procedure fails. Figure 2 represents these cases. A Tango value from the test device is requested in the first message. The response contains the value, its set point, if available (in square brackets), the measurement units and the timestamp. The final section indicates which Tango database has been used. The second message is the name of a quantity that is not defined, and an error is reported. The third source identifies an EPICS variable. Its value is provided together with the measurement units ("Counts") and the quality (error severity) of the value, in italics.

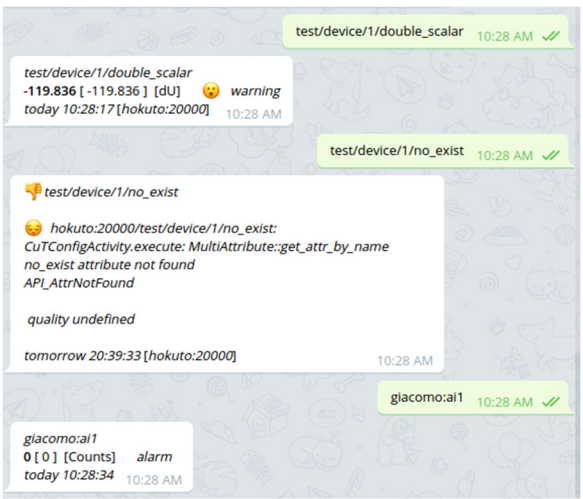

Figure 2: Single read operations.

Monitor

A variable can be monitored over time. The *monitor* command placed before the source name starts the operation. The frequency of the updates is determined by how they are supplied by the underlying system. For example, EPICS variable monitoring and Tango events will produce an update whenever a new value is pushed from the engine. On the other hand, if *cumbia* is in charge of polling the quantity, the period is defined both by a limit on the operation rate set on the *bot* by the administrator and the number of sources being read with this method. The maximum allowed bandwidth is divided among the users currently polling the control system. This is a security feature that prevents the *bot* from overloading the control system regardless of the number of users connected. Notifications sent by a *monitor* are *silent*. In effect Telegram API offers this option, although the actual behaviour may be device dependent.

The logic of the *monitor* command is as follows. Simple variables or their combination into formulas determine how a new message is delivered: there must be a change in the value in the first case; in the formula evaluation result in the second. By default, the most recent monitor messages are *updated* rather than new ones issued. If users prefer to apply the evaluation principles of the *monitor* directive but still receive every update in a separate message (and the corresponding silent notification), they must use the *warn* instruction instead of *monitor*. Finally, if sound notifications are desired for the events herein described, the *configure* command must be sent and the appropriate option enabled[2]. An ongoing operation can be stopped either through a shortcut in the list produced by the */monitors* command or from within a message itself. The */stop* directive stops monitors and alerts altogether.

Alert

Alerts enable the user to receive a notification either when a source changes its quality (error severity) or the evaluation of a formula switches from false to true. Unlike monitors, alert notifications are not silent and are either *quality* or *formula evaluation* centred. Consider the following examples:

```
alert test/device/1/long_scalar
alert test/device/1/double_scalar > -60 &&
      test/device/1/double_scalar <  60
```

In the first case, you will be notified when *long_scalar* quality changes from normal to warning to alarm and vice versa. In the second case, one alert is sent each time the value enters the range (-60, 60). Figures 3 and 4 show the trend of the *double_scalar* variable and the corresponding alert messages on the Telegram application.

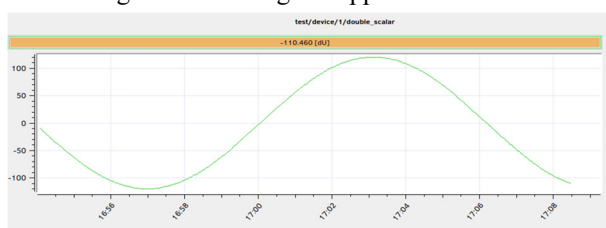

Figure 3: Trend of a variable over time.

An alert can be stopped either through a shortcut in the list produced by the */alerts* command or from within an alert message itself. The */stop* command stops monitors and alerts altogether.

Monitors and alerts have a time to live whose span depends on the user rights. In the example in Fig. 4 the lifetime is one day. After that period, the instruction must be repeated to restart the process.

[2] The watch and configure commands are still under development at the moment of writing this paper.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA148

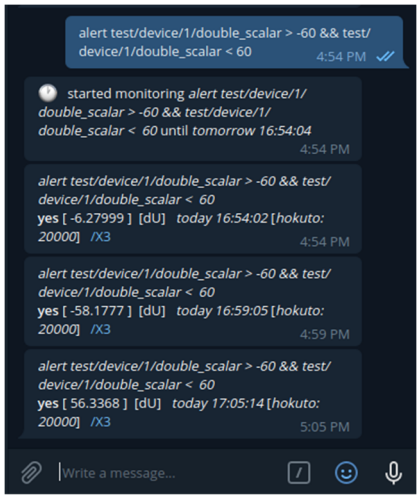

Figure 4: Alerts associated to the trend in Fig. 3.

UTILITIES

As pointed out in the Features section, *cumbia-telegram-bot* design target is to guarantee an immediate, pleasant and even funny user experience. The main objective is to ensure that most operations are triggered by *tap* actions after an initial configuration where at least the preferred commands and names of the sources must be manually introduced. The Telegram Desktop version can be a very versatile tool to accomplish this initial setup. This chapter introduces the most relevant features that enhance the user interaction.

Repeat Last Command

If an operation is successful, it can be sent over again with the */last* command. Failed readings are not saved.

History

The *bot* keeps a history of successful tasks. The number of its records is decided by the administrator. */alerts*, */monitors* and */reads* commands provide access to the respective history buffers. From the reply, it is possible to start the operations over again as well as stop the running ones individually.

Bookmarks

A bookmark can be assigned to the successful operation. The list can be later retrieved and the instructions can be wherefrom imparted again.

Alias

Remembering and writing complicated source names or formulas can be a nuisance, especially when they must be written on a messaging application, without the support of a dictionary. *Cumbia-telegram-bot* addresses this issue with a module that allows the user to assign an *alias* to practically anything. Figure 5 shows a typical usage of aliases.

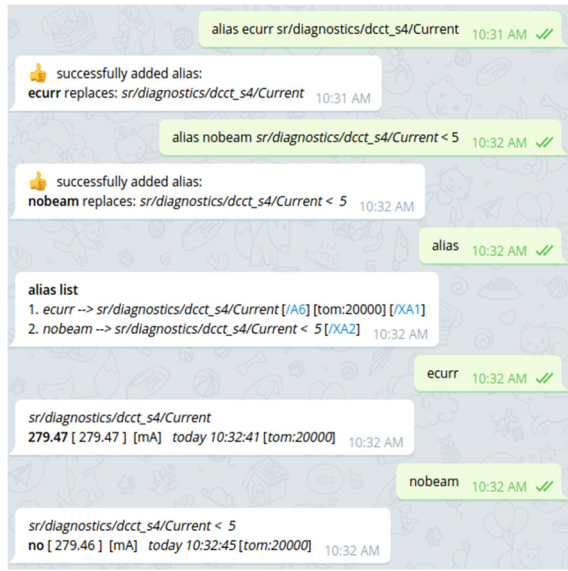

Figure 5: Usage of *alias*. One can see the shortcuts to remove an entry from the list, [/XA1] and [/XA2].

Message Titles

Figure 6 shows first a notification of an alert occurring when a test variable goes below a given threshold of 10.

Figure 6: Titles help identify the observed source.

It is not immediate to figure out what happened, especially if multiple monitors are active. When writing an expression, it is possible to add a title that will be placed at the beginning of the text, thus making the nature of the alert clearer at a glance, as can be seen in the second notification. Titles are specified after the formula and two "/" characters, like comments in many programming languages.

FORMULAS

Formulas can be used to configure specific alerts or warnings where the combination of multiple values is arranged to define the conditions to deliver them. The *bot* employs the *cumbia formula plugin* plus a processor (parser) that formats the expression of simple operations into the exact pattern required by the plugin. Formulas and

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA148

functions must be compatible with the JavaScript language. The application can be used as a simple calculator (Fig. 7) as well as a powerful tool to receive custom-tailored alarms (Fig. 8).

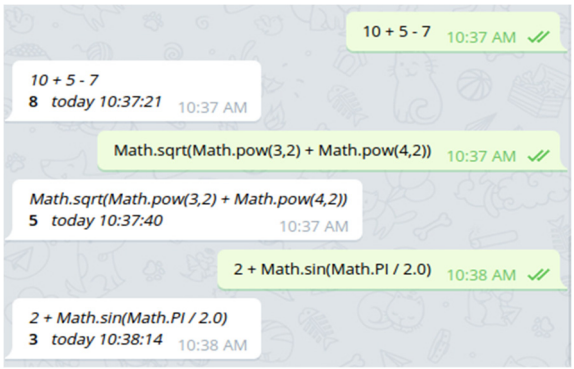

Figure 7: The *bot* used as a calculator.

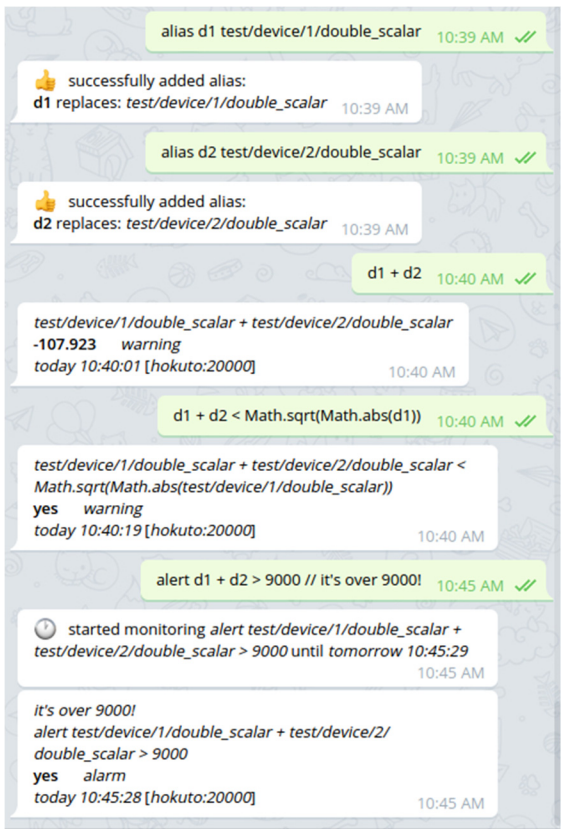

Figure 8: The *bot* used to combine sources into formulas and generate personalised alarms.

Sources from different engines can be mixed into formulas, as demonstrated in Fig. 9.

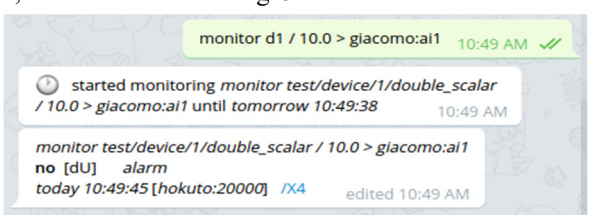

Figure 9: Monitor combinations of sources from different engines.

EXTENSIONS

Cumbia-telegram-bot can be extended with plugins. They must implement an interface (named *CuBotModule*) and register to the application with a given priority. The priority (*type*) determines the order with which the application delivers the messages to the modules. For example the *alias* component has a very high priority (low *type* value) because in the first instance alias words must be searched and, if found, they must be replaced and the message sent onward down in the chain. A module can thus process a message or forward it to a lower priority module. Extensions can be used to implement custom functions, for example those specific to a given control system, as described in the next section.

Tango Database Search

The *search* plugin can be used to find a Tango device matching a given pattern. The availability of the plugin in Telegram can be checked with the command */modules*. A specific help section is dedicated to the users of the component. From the list of devices found according to the search pattern, a message with the attribute record can be obtained with a simple *tap*. From that place, any attribute can be read with one touch. Figure 10 shows an example of the module usage.

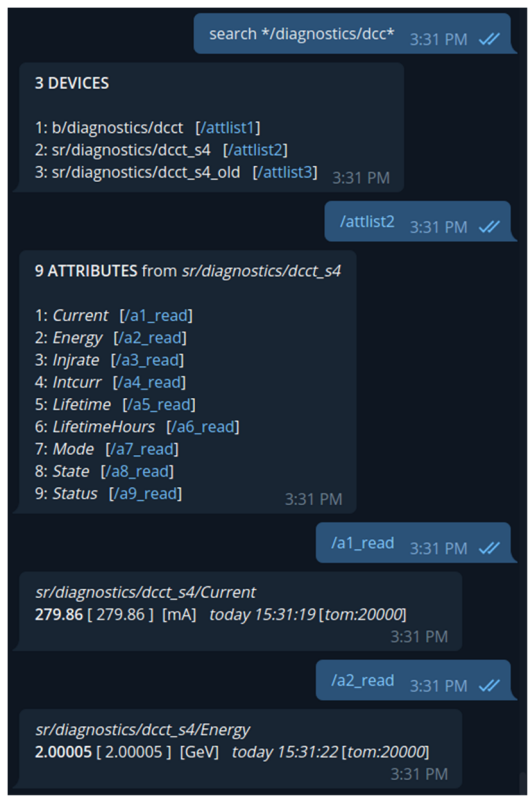

Figure 10. The *search* module in action.

The Tango Host

A characteristic of the Tango control system is the grouping of devices by databases that can be distributed across different hosts. To address a source, you need to specify on which host the corresponding device is exported. Normally

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA148

an environment variable accounts for this information, but the Telegram *bot* must be flexible enough to let the user switch across all the Tango hosts that are open to the *bot* server. The administrator populates the dedicated table in the small database used by the *bot* with the list of allowed hosts. The *host* module provides utilities to get the record of accessible Tango database names, print and change the one in use. The latter action can be accomplished with a single touch on the desired entry in the list. When dealing with engines relying on configuration databases, history and *alias* components associate them to the source names. This guarantees the correct invocation of recorded commands despite the host selection presently in use.

CONFIGURATION

A special Telegram *bot*, called *The BotFather*, is used to register custom *bots*, configure them and obtain a *token* that identifies them henceforth. *The BotFather* can be given instructions to define the name of the *bot*, a set of commands (e.g. */alias, /bookmarks, /monitors, /alerts*), a logo and so on. To start the *cumbia-telegram-bot*, the *token* is used as a command line parameter, along with the *SQLite* database file name. From this point on, clients will be able to join in and wait for the authorization from the administrator before messaging.

CONCLUSIONS

The *cumbia-telegram-bot* is a small tool that physicists and control room operators can put in their pocket to stay in touch with a machine, check its or one of its subsystems health and receive alerts when events require attention. On the control system side, the design objective centres on security and protection against network overload. In times where our lives are so involved in social media and messaging applications, the *cumbia-telegram-bot* is a gimmick that can put work in a little more entertaining perspective.

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/