**Solaris Synchrotron Control Program Extensions: Device Group Panel**

User Manual

|  |  |
| --- | --- |
| Revision: |  |
| Status: | Released |
| Repository: | acc |
| Project: |  |
| Folder: |  |
| Document ID: |  |
| File: |  |
| Owner: | Vid Juvan |
| Last modification: | June 17, 2015 |
| Created: | April 28, 2014 |

|  |
| --- |
| **Prepared by** |
| Vid Juvan |
|  |

Document History



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Revision | Date | Changed/reviewed | Section(s) | Modification |
| 1.0 | 19-6-2014 | Vid Juvan |  | Initial version |

Confidentiality

This document is classified as a public document. As such, it or parts thereof are openly accessible to anyone listed in the Audience section, either in electronic or in any other form.

Scope

This document provides a description and instructions how to use the Solaris Synchrotron Control Program.

Table of Contents

[1. Introduction 4](#_Toc422489099)

[2. Device Group Panel 5](#_Toc422489100)

[2.1. Device Group Panel configuration 5](#_Toc422489101)

[2.1.1. Device configuration 5](#_Toc422489102)

[2.1.2. Group configuration 7](#_Toc422489103)

[2.2. Device Group Panel start procedure 8](#_Toc422489104)

[2.3. Directory and file structure 9](#_Toc422489105)

[2.4. Device Group Panel structure 10](#_Toc422489106)

[2.4.1. Runner script 10](#_Toc422489107)

[2.4.2. Main Device Group Panel file 11](#_Toc422489108)

[2.4.3. Main QDialog UI file 13](#_Toc422489109)

[2.4.4. Widget UI file 20](#_Toc422489110)

Figures

[Figure 1: Device Group Panel template: runner script 12](#_Toc422489246)

[Figure 2: Device Group Panel template: main file 1 13](#_Toc422489247)

[Figure 3: Device Group Panel template: main file 2 14](#_Toc422489248)

[Figure 4: Device Group Panel template: QDialog UI file 2 15](#_Toc422489249)

[Figure 5: Device Group Panel template: QDialog UI file 2 16](#_Toc422489250)

[Figure 6: Device Group Panel template: QDialog UI file 2 17](#_Toc422489251)

[Figure 7: Device Group Panel template: QDialog UI file 3 17](#_Toc422489252)

[Figure 8: Device Group Panel template: QDialog UI file 4 19](#_Toc422489253)

[Figure 9: Device Group Panel template: Widget UI file: 1 22](#_Toc422489254)

[Figure 10: Device Group Panel template: Widget UI file: 2 23](#_Toc422489255)

Tables

[Table 1: Device Configuration Format 6](#_Toc422489211)

[Table 2: Group Configuration Format 7](#_Toc422489212)

Typography

This document uses the following styles:

|  |  |
| --- | --- |
|  | A box like this contains important information. |

|  |  |
| --- | --- |
|  | A box like this would contain sidebar text. |

|  |  |  |
| --- | --- | --- |
|  | Warning!  A box like this provides information, which should not be disregarded! |  |

Glossary of Terms

|  |  |
| --- | --- |
| Tango | TANGO is an object oriented distributed control system (http://www.tango-controls.org/) |
|  |  |
|  |  |
|  |  |

References

1. Tango-related documentation: <http://www.tango-controls.org/>
2. Solaris Synchrotron Control Program: <http://git.m.cps.uj.edu.pl/controlroomsoftware/app-cosylab-controlprogram>
3. Device Group Panel template: <http://git.m.cps.uj.edu.pl/controlroomsoftware/app-cosylab-templategroupgui>
4. PyQt4 class reference: <http://pyqt.sourceforge.net/Docs/PyQt4/classes.html>
5. Taurus: <http://www.taurus-scada.org/en/latest/docs.html>
6. Cosywidgets library: <http://git.m.cps.uj.edu.pl/controlroomsoftware/lib-cosylab-cosywidgets>

# Introduction

Solaris Synchrotron Control Program provides a high-level overview of the operation of the synchrotron devices and enables access to the device engineering screens. It is instantiated according to the provided main device CSV file and connects to the corresponding Tango database of the control system instance.

The main purpose of the application is to provide the user with easy access to various visual control panels of the specified devices. Three types of panels can be accessed, namely Standard Device Panel, Custom Panel and Device Group Panel. This documentation regards the latter.

The concern of this documentation is strictly just the Device Group Panel for the Solaris Synchrotron Control Program. The documentation regarding the Control Program itself can be found in the dedicated repository [2]. In the scope of this documentation, we will only describe the functionality of the Control Program that directly concerns the implementation of the Device Group Panels.

In the documentation, we assume that the user is familiar with the PyQt4 [4] and Taurus [5] libraries.

# Device Group Panel

The Device Group Panel is a GUI that exposes multiple devices within one GUI. All of them are custom made, but foreseen to be dynamic to a certain extent. Usually, multiple instances of the same Device Group Panel are used within the control program, differing only in the configuration and the name.

For example, the following Device Group GUI instances are created within the Control Program:

* VAC\_K00
* VAC\_K01
* VAC\_K02
* VAC\_K03
* …

They all use the same Device Group Panel implementation, but feed it with different configuration.

Device Group Panels are generally built on a Control Program Device Group template. The Device Group Panel template is provided in the dedicated repository [3]. Henceforth, we will consider the provided template as a base for Device Group Panel development.

## Device Group Panel configuration

Device Group GUI receives the entire configuration from the Control Program in a form of the input parameters. However, in order for the user to fully understand the configuration procedure to tackle the development of Device Group GUIs, one needs to be familiar with how the Control Program reads and forwards the configuration.

Two sources are used for configuring Device Group Panels. The former is referred to as the main device CSV configuration file. The latter is referred to as the group configuration file. In the following chapters we explain the syntax and provide the description for both.

### Device configuration

The device configuration CSV is the main configuration file for the Control Program. It is device oriented. Below is provided a device configuration BNF format. Here we only provide the details of the configuration that is relevant for the Device Group Panels. For more information, refer to the Control Program documentation [2].

|  |  |
| --- | --- |
| **Device Configuration BNF Format** | |
| CSV configuration: | 1\*(DEVICE\_SPECIFICATION) |
| DEVICE\_SPECIFICATION | ELEMENT\_NAME, “,”,  TYPE, “,”,  L, “,”, S, “,”, X, “,”, Y, “,”, Z, “,”, SECTION, “,”,  SUBSYSTEM, “,”,  MANAGED\_IN\_CS, “,”,  DEVICE\_SERVER\_NAME, “,”, DEVICE\_SERVER\_INSTANCE, “,”,  DEVICE\_CLASS, “,”,  FULL\_TANGO\_DEVICE\_NAME, “,”,  DEVICE\_ALIAS, “,”,  TRIGGERED\_BY\_TTL, “,”,  CUSTOM\_GUI, “,”,  DEVICE\_GROUP\_GUIS, “,”,  DESCRIPTION, “,”,  COMMENT; |
| DEVICE\_GROUP\_GUIS | “” | (DEVICE\_GROUP\_GUI, 0\*(“|”, DEVICE\_GROUP\_GUI)); |
| DEVICE\_GROUP\_GUI | DEVICE\_GROUP\_INSTANCE, “-”, DEVICE\_GROUP\_ID; |
| DEVICE\_GROUP\_INSTANCE | SCRIPT\_NAME, INTEGER\_SUFFIX; |

Table 1: Device Configuration Format

Upon the device configuration, the Control Program parses all different Device Group Panel instances and all devices that concern it. The Control Program assembles a configuration array that is sent to the Device Group Panel runner function upon opening the panel. The process is described below on a simple example.

Assume the following configuration:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ELEMENT\_NAME** | **…** | **FULL\_TANGO\_DEVICE\_NAME** | **…** | **DEVICE\_GROUP\_GUIS** | **…** |
| SOME\_VALVE1 |  | SOME/VALVE/1 |  | VAC\_K00-VGM\_R|VAC\_K01-VGM\_L |  |
| SOME\_ION\_PUMP1 |  | SOME/IONPUMP/1 |  | VAC\_K00 -IPC1 |  |
| SOME\_ION\_PUMP2 |  | SOME/IONPUMP/2 |  | VAC\_K01 -IPC1 |  |

The Control Program would define the following:

|  |  |  |
| --- | --- | --- |
| **Device Group Panel instance** | **Devices** | **Device IDs** |
| VAC\_K00 | SOME/VALVE/1 | VGM\_R |
| SOME/IONPUMP/1 | IPC1 |
| VAC\_K01 | SOME/VALVE/1 | VGM\_L |
| SOME/IONPUMP/2 | IPC1 |

For every Device Group Panel instance, the Control Program prepares its configuration. The configuration is in a form of an array. When a user opens an instance of the Device Group Panel, this configuration is passed to the “getGuiWidget” function of the corresponding python file (refer to the next chapters).

The Device Group Panel configuration array is assembled as follows:

* A configuration array contains the flag for label (“--LAB”), followed by the Device Group Panel instance name.
  + E.g. CONF\_ARRAY = [“--LAB”, “VAC\_K00”, …]
* For every device that corresponds to the Device Group Panel instance, three items are added, namely the device ID flag (--DEVICE\_ID), full Tango device name and the device description. They are added consecutive, in the above order.
  + E.g. CONF\_ARRAY = [… , “—VGM\_R”, “SOME/VALVE/1”, DESCRIPTION1, …]

For the above example of device configuration, Control Program would generate the following Device Group Panel configuration arrays:

|  |  |
| --- | --- |
| **Device Group Panel instance** | **Configuration array** |
| VAC\_K00 | [“--LAB”, “VAC\_K00”,  “—VGM\_R”, “SOME/VALVE/1”, DESCRIPTION1  “—IPC1”, “SOME/IONPUMP/1”, DESCRIPTION2] |
| VAC\_K01 | [“--LAB”, “VAC\_K01”,  “—VGM\_L”, “SOME/VALVE/1”, DESCRIPTION1  “—IPC1”, “SOME/IONPUMP/2”, DESCRIPTION3] |

### Group configuration

Group configuration is a separate configuration CSV file, used only for configuring the Device Group Panels. Three configuration elements can be placed for every Device Group Panel instance, namely a set of succeeding Device Group Panel instances, a set of preceding instances, and an additional input parameter. Bellow we provide a BNF format of this configuration file.

|  |  |
| --- | --- |
| **Group Configuration BNF Format** | |
| CSV configuration: | 1\*(DEVICE\_GROUP\_SPECIFICATION) |
| DEVICE\_GROUP\_SPECIFICATION | DEVICE\_GROUP\_PANEL\_INST\_NAME, “;”,  PRECEEDING\_INSTANCES, “;”,  SUCEEDING INSTANCES, “;”,  ADDITIONAL\_PARAMETER, “\n” |
| PRECEDING\_INSTANCES | “” | (PRE\_DEV\_GROUP\_PANEL\_INST\_NAME, 0\*(“|”, GROUP\_GUI\_NAME)) |
| SUCEEDING\_INSTANCES | “” | (SUC\_DEV\_GROUP\_PANEL\_INST\_NAME, 0\*(“|”, GROUP\_GUI\_NAME)) |
| ADDITIONAL\_PARAMETER | Any string. It can contain “\n”, if entire string is enclosed in quotes. |

Table 2: Group Configuration Format

#### Additional parameter

The Control Program, according to the configuration, updates the configuration array for every Device Group Panel instance as follows:

* If an additional parameter is not empty:
  + A flag for additional parameter (“--ADD”) is added to the configuration array, followed by the actual additional parameter.
    - CONF\_ARRAY += [“--ADD”, ADDITIONAL\_PARAMETER]

With this, we achieve a way of sending additional information to the Device Group Panel, separately for every instance. It is up to the Device Group Panel implementation to handle this additional input. A simple example when this would be useful is when PLC signals showing temperatures have to be specified for each instance of the vacuum Device Group Panel.

#### Preceding and succeeding instances

For every Device Group Panel instance, the Control Program notes all the preceding and succeeding instances. These are not added to the configuration array. They are handled upon opening the Device Group Panel instance (refer to the next chapters).

## Device Group Panel start procedure

The Control Program parses instances of the Device Group Panels from the configuration. When running any instance of the panel, the following process takes place:

* The Control Program looks for the name of the python file, corresponding to the Device Group Panel instance that we want to open.
  + The name of the python file is the same as the name of the Device Group Panel instance, omitting a possible integer suffix. For example, Device Group Panel instance with a name “VAC\_K00” would correspond to a python file “VAC\_K.py”.
  + The file must be located in a dedicated directory, which is provided to the ControlProgram when it is started, Refer to the ControlProgram documentation [2] for more information.
* The Control Program look for a command called “getGuiWidget” within the python file:
  + If the function does not exist, the Control Program runs the python file in a separate sub-process. This is useful for integrating certain external applications into the Control Program. However, we will assume that the function exists.
  + Control Program runs the function “getGuiWidget” and passes the configuration array of that particular Device Group Panel instance to the function. For more detail on the configuration array, refer to the previous chapter.
  + Control Program expects an instance of the QDialog in return.
* An instance of QDialog is obtained. The following procedure takes place:
  + The Control Program calls (if it exists) a function of the QDialog called “setManagerInstance”. It provides a Control Program manager as an argument to the function. The Device Group Panel is in this way provided the means of communicating with the Control Program.
  + For every succeeding Device Group Panel instance in the chain (refer to the configuration chapter), the Control Program calls a function of the returned QDialog called “addNext” (if it exists). As arguments to the function, the Control Program first provides the callback function, which has to be called when a current Device Group Panel instance wants to open that particular succeeding instance. As the second argument, the control program provides an index, which has to be passed to the callback function. As the third argument, the Control Program provides a short description of the succeeding instance.
  + For every preceding Device Group Panel instance in the chain (refer to the configuration chapter), the Control Program calls a function of the returned QDialog called “addPrev” (if it exists). The arguments are the same as for the “addNext” function.
* The QDialog is shown under the control of the Control Program. In this way, various optimizations are enabled.

## Directory and file structure

The Control Program uses a dedicated GUI directory for custom and device group GUIs. The python files used for running the Device Group Panels must be located in that directory, on the first level.

For more transparent development, each Device Group Panel is kept in a separate GIT repository. They are included in the dedicated GUI directory as GIT submodules, each in a separate directory. For every Device Group Panel, a dedicated runner script is created and placed in the GUI directory, alongside the Device Group Panel submodules. The runner script is a small python file that is being called by the Control Program. The correct naming of runner scripts is essential. For every Device Group Panel, multiple runner scripts can be created, supporting different names of the Device Group instances within the Control Program. When the Control Program starts the Device Group Panel, it calls a function “getGuiWidget” in the corresponding runner script. The script then imports code from the Device Group Panel submodule, and runs a function “getGuiWidget” there. The QDialog is returned to the Control Program. The runner scripts serve only for forwarding the calls of the Control Program and are regarded as part of the Control Program, and not as a part of Device Group Panels.

For example, the file structure for the vacuum Device Group Panel:

* Synchrotron\_GUIs (dedicated GUI directory of the Control Program)
  + VAC\_K.py (runner script, contains “getGuiWidget” function)
  + VAC\_R1\_.py (another runner script)
  + app-cosylab-vacuumgroupgui (Device Group Panel repository as submodule)
    - VacuumGroupGUI.py (contains “getGuiWidget” function)
    - VacuumGroup.py
    - IonPumpWidget.py
    - …

## Device Group Panel structure

Device Group Panel structure can be realized in a desired way. The only limitations are:

* A function called “getGuiWidget” must be implemented in one of the files.
* Function “getGuiWidget” accepts an array of configuration parameters.
* Function “getGuiWidget” returns an instance of QDialog.
* Device Group Panel must not initialize a new instance of the Taurus application.

In this section, we will describe a structure of a standard implementation of the Device Group Panel. The Device Group Panel template provides such a structure.

A standard implementation of the Device Group Panels consists of the following files:

* Runner python script(s)
  + Officially a part of the Control Program
  + Includes a “getGuiWidget” function
  + E.g. VAC\_K.py
* Main Device Group Panel file
  + Used for parsing the input
  + Includes a “getGuiWidget” function
  + E.g. VacuumGroupGUI.py
* Main QDialog UI file
  + Excepts parsed input from the main file
  + Sets up the QDialog
  + E.g. VacuumGroup.py
* A set of widget UI files
  + A widget UI file for every type of the device in the Device Group Panel
  + E.g. IonPumpWidget.py

### Runner script

Runner scripts servers just for forwarding the function call from the Control Program to the Device Group Panel implementation.

Figure 1: Device Group Panel template: runner script



* The runner script adds a subfolder that belongs to the Device Group Panel to the python path.
* The runner script imports the main Device Group Panel file.
* The runner script calls a function “getGuiWidget” of the main Device Group Panel file, and forwards the returned QDialog.

### Main Device Group Panel file

Main Device Group Panel consists of one function called “getGuiWidget”. All the functionality is comprised in it.

Figure 2: Device Group Panel template: main file 1



* In the main Device Group Panel file, the configuration array is parsed into a map of devices (referred to as GUI groups), an array of additional arguments and a title. The devices are separated by their types, which are deduced from the device IDs.
  + A map of devices has a device type name as a key (ELEMENTX in figure 2).
    - E.g. “ION PUMPS”
  + A value of the device map is an array of device arrays that are of the same type. The first value of the device array is the ID of the device. The second value is a full Tango device proxy. The third value is a device description.
    - E.g. gui\_groups[“ION PUMPS”] = [ [“-IPC1”, “SOME/IONPUMP/1”, “desc”] , [[“-IPC2”, “SOME/IONPUMP/2”, “desc”] , …]

Figure 3: Device Group Panel template: main file 2



* It creates an instance of QDialog and an instance of the QDialog UI class, imported from the main QDialog UI file. It sends the QDialog, a map of devices and an additional arguments array to the main QDialog UI class to set up the QDialog. Additionally, it sets the title. Afterwards, it returns the QDialog instance.

### Main QDialog UI file

The Main QDialog UI file consists of one class, regarded as QDialog UI class.

The QDialog UI class file is in charge of setting up the entire QDialog. It makes use of widget UI classes, provided in separate files.

Figure 4: Device Group Panel template: QDialog UI file 2



* QDialog UI class first creates the layouts. On the left side, it creates a layout, intended for the buttons leading to the preceding Device Group Panel instances. In the middle, a main layout, holding the device groups and device widgets is created. On the right side, another layout is created, intended for the buttons leading to the succeeding Device Group Panel instances.

Figure 5: Device Group Panel template: QDialog UI file 2



* + For every group/type of devices in the device map, it creates a dedicated group layout for it. Above the group layout, a button is positioned, intended for hiding and showing the group layout. For every device in the group, a corresponding widget UI and a fresh instance of the QWidget are created. A special array is assembled, holding the full Tango device name and its description. The array and an instance of QWidget are sent to a widget UI. The widget UI constructs the widget. The widget is added to the group layout. The widget UI is added to the child widget UI list.

Figure 6: Device Group Panel template: QDialog UI file 2



* The QDialog UI class creates a new thread. This thread is in charge of performing heavy operations, mostly setting the model, for all widgets of the Device Group Panel. The widget UIs must already now the model, the thread merely calls a function of the widget UI to trigger the execution. This is implemented so that the heavy operations are not executed in the GUI thread. Instead, they are executed separately. The result of this is that the panel opens immediately in all cases. The background thread might take longer to complete the heavy operations. The thread can be stopped prematurely. When the QDialog is closed, the thread is stopped in a control manner. The function of the thread and a function for stopping the thread prematurely are described below.

Figure 7: Device Group Panel template: QDialog UI file 3



* The QDialog UI class creates a context menu. When a user right-clicks anywhere on a screen, the QDialog presents him with three options: “Close”, “Show descriptions” and “Hide descriptions”. If a user selects the former, the QDialog closes. If a user selects “Show descriptions”, the QDialog will execute a dedicated function. The details of the function are explained below. Ultimately, the dedicated function shows the descriptions of all child widgets of the QDialog. If a user selects “Hide descriptions”, the QDialog will execute another dedicated function. The details of the function are explained below. Ultimately, this dedicated function hides the descriptions of all child widgets of the QDialog.

Figure 8: Device Group Panel template: QDialog UI file 4



* Certain additional functions are implemented:
  + setManagerInstance
    - This function is called by the Control Program. An instance of the Control Program manager is passed as an argument. The function stores the instance.
  + openEnginneringScreen
    - Helper function, which connects to the Control Program manager and requests to open an engineering screen for a certain device. A full tango device name must be provided as an argument to this function.
  + addNext
    - This function is called by the Control Program for every succeeding Device Group Panel instance. The function creates a new button in the right layout of the QDialog. It binds the button to call a callback function that was provided by the Control Program as an input argument to this function, when clicked. It passes an index to the callback function. The index is also provided as an argument. Moreover, a tooltip is added to the button, displaying the description of the succeeding instance. The description is provided by the Control Program as a third input parameter to this function.
  + addPrev
    - This function is called by the Control Program for every preceding Device Group Panel instance. The function creates a new button in the left layout of the QDialog. It binds the button to call a callback function that was provided by the Control Program as an input argument to this function, when clicked. It passes an index to the callback function. The index is also provided as an argument. Moreover, a tooltip is added to the button, displaying the description of the preceding instance. The description is provided by the Control Program as a third input parameter to this function.
  + closeHandler
    - This function stops the background thread, if it did not yet end.
  + setModelRun
    - This is the function of the background thread. It goes through the list of saved widget UIs, and calls its function for setting the model. It is essential that whenever a widget is added to the QDialog that the widget UI is added to the list of child widget UI list. As mentioned, the child widget UI must already know the model; this function merely triggers the execution. It calls a function called “setModel” of the widget UI, passing no arguments. Every widget UI in the list of child widget UIs must implement such a function. Any functionality that can be run in a separate thread can be added to the setModel function of the child widget UI.
    - The function performs a check before every call to the child widget UI. The check is implemented so that the main thread can request the background thread to stop prematurely.
  + showDescriptions
    - This function iterates through the list of child widget UIs, and calls its function “showDescription”. Every child widget UI of the QDialog must implement such a function. The “showDescription” function of a child widget UI is intended to show a special sub-widget within the child widget that displays the description.
  + hideDescriptions
    - This function iterates through the list of child widget UIs, and calls its function “hideDescription”. Every child widget UI of the QDialog must implement such a function. The “hideDescription” function of a child widget UI is intended to show a special sub-widget within the child widget that displays the description.
  + showHideNamedGroups
    - This function is a helper function for buttons that show and hide the group layouts.

### Widget UI file

Multiple widget UI files are generally used. Each contains the functionality for setting up a widget for a certain type of device.

As a part of a Device Group Panel template, we provide the base implementation of the widget UI class. Note that the template uses certain widget from the Cosywidgets [6] library.

Figure 9: Device Group Panel template: Widget UI file: 1



* The widget UI file must contain a UI\_Form class. The class contains one main function called “setupUI”. The function accepts a widget and device specifications array. It first stores the widget instance. It then sets up the widget, adding any desired sub widgets and layouts to it.

Figure 10: Device Group Panel template: Widget UI file: 2



* Apart from the “setupUI” function, the widget UI class must also contain the functions “showDescription” and “setModel”. Above is an example from the template.