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

User Manual

|  |  |
| --- | --- |
| Revision: |  |
| Status: | Released |
| Repository: |  |
| Project: |  |
| Folder: |  |
| Document ID: |  |
| File: |  |
| Owner: | Vid Juvan |
| Last modification: | April 20, 2015 |
| Created: | April 28, 2014 |

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

Document History



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Revision | Date | Changed/reviewed | Section(s) | Modification |
| 1.0 | 23-4-2014 | 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](#_Toc387924571)

[2. Software installation 5](#_Toc387924572)

[2.1. Prerequisites 5](#_Toc387924573)

[2.2. Facility Configuration Software Installation 5](#_Toc387924574)

[2.3. Solaris Synchrotron Control Program Installation 5](#_Toc387924575)

[2.4. Running the application 5](#_Toc387924576)

[2.4.1. Manual start 6](#_Toc387924577)

[2.4.2. Runner application 6](#_Toc387924578)

[3. Configuration 7](#_Toc387924579)

[4. Operation 9](#_Toc387924580)

[4.1. Device Tree 9](#_Toc387924581)

[4.1.1. Device Filtering 9](#_Toc387924582)

[4.1.2. Standard Device Panel 9](#_Toc387924583)

[4.1.3. Custom Panel 10](#_Toc387924584)

[4.2. Device Group Tree 10](#_Toc387924585)

[4.2.1. Device Group Panel 11](#_Toc387924586)

Figures

**No table of figures entries found.**

Tables

[Table 1: Configuration Format 7](#_Toc387924567)

[Table 2: Configuration Specifications 8](#_Toc387924568)

[Table 3: Standard Device Panel Script Input Parameters 10](#_Toc387924569)

[Table 4: Device Group Script Input Parameters 11](#_Toc387924570)

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>

# 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 [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.

# 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.

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 [3]. Henceforth, we will consider the provided template as a base for Device Group Panel development.

## 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 Control Program look for a command called “getGuiWidget” within the python file. If the function exists:
  + Control Porgram runs the function and passes the entire configuration for the Device Group Panel to the function in a form of an array. For more derail on the configuration, refer to the next chapters.
  + Control Program expects an instance of the QDialog in return. The QDialog is then shown under the control of the Control Program. In such a way, various optimizations are enabled.
* 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.

## 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 a 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. TODO

## Additional Device Group Panel functionality

To enhance the Device Group Panel, the following additions were implemented:

* Chaining Device Group Panel instances
  + A chain of Device Group Panel instances can be defined in the configuration.
  + For every preceding or acceding element in the chain, the Control Program can call a function of the Device Group Panel and provide a callback for opening the preceding/acceding instance.
  + A simple example would be buttons on the left and the right side of the Device Group Panel. When a user would click on a button, the Control Program would open a preceding/acceding instance of the Device Group Panel, and close the current instance.
* Communication between Device Group Panel and the Control Program
  + The Control Program can provide an instance of itself to the Device Group Panel. In such a way, a Device Group Panel has access to the functionality of the Control Program.

### Chaining Device Group Panel instances

### Communication between Device Group Panel and the Control Program

## 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. In the following chapters we explain the syntax and provide the description for both.

Device Group Panels are custom made panels, built on a ControlProgram Group template. They provide access to a group of devices. The content of each Device Group Panel is dynamically generated, according to the input. Another feature that is supported within the scope of the Device Group Panels is a support for chaining the panels. For every instance of the Device Group Panel, one can define a set of preceding and succeeding instances of the Device Group Panels. Upon this information, buttons are generated on the left and right side of the respective Device Group Panel, which provide easy access to the specified instances of panels. Moreover, certain Device Group Panels require additional specific configuration that exceeds the scope of the main device CSV file.

In retrospect, for each instance of the Device Group Panel, an additional configuration is foreseen that includes the set of preceding and succeeding instances in the chain and an additional parameter for specific configuration. This configuration is kept in a group CSV configuration file, described below.

|  |  |
| --- | --- |
| **Group Configuration BNF Format** | |
| CSV configuration: | 1\*(GROUP\_SPECIFICATION) |
| GROUP\_SPECIFICATION | GROUP\_GUI\_NAME, “;”,  PRECEEDING\_INSTANCES, “;”,  SUCEEDING INSTANCES, “;”,  ADDITIONAL\_PARAMETER, “\n” |
| PRECEEDING\_INSTANCES | 0\*1(GROUP\_GUI\_NAME, 0\*(“|”, GROUP\_GUI\_NAME)) |
| SUCEEDING\_INSTANCES | 0\*1(GROUP\_GUI\_NAME, 0\*(“|”, GROUP\_GUI\_NAME)) |
| GROUP\_GUI\_NAME | Name of a desired Group GUI instance |
| ADDITIONAL\_PARAMETER | Any string. It can contain “\n”, if entire string is enclosed in quotes. |

Table 5: Group Configuration Format

It is not required that all instances of Group GUIs are specified in the Group configuration file. The instances are parsed from the device configuration. The group configuration provides additional configuration and is thus optional.

# Software installation

## Prerequisites

* Python 2.7
* Python modules
  + PyQt4 (version: 4.11.2)
  + PyTango (version: >= 8.1.5)
  + Taurus (version: >= 3.3.0)
  + Numpy (version: >= 1.8.1)
  + CosyWidgets (version: >= 0.1)
  + MaxWidgets (version: >= 0.8.3)
  + Csv, subprocess, argparse, threading, time, os, json, datetime, contextlib, base64, sys, pickle, re, math, signal, traceback, collections, copy, …

## Facility Configuration Software Installation

* Download source code
  + svn checkout https://internal.cosylab.com/svn/acc/projects/Solaris/Utils/FacilityConfiguration/
* Populate the Tango database
  + cd FACILITY\_CONFIGURATION
  + python FacilityConfoguration.py ./CSVFiles/synchrotron\_devices.csv ./CSVFiles/names-dictionary.csv
  + python FacilityConfoguration.py ./CSVFiles/BL-05ID.csv ./CSVFiles/names-dictionary.csv
  + python FacilityConfoguration.py ./CSVFiles/BL-04ID.csv ./CSVFiles/names-dictionary.csv

## Solaris Synchrotron Control Program Installation

### Using GUIRunner Application

When using a GUIRunner, this step can be omitted. GUIRunner will at the initialization download all source code and configuration files automatically. The use of GUIRunner application is advised, to ensure that the latest version of the source code and the CSV configuration files are being used.

### Manual Use

* Download source code from Cosylab SVN
  + svn checkout <https://internal.cosylab.com/svn/acc/projects/Solaris/Utils/ControlProgram/>
* Download source code from Solaris GIT
  + git clone –b cosylab git://git.cps.uj.edu.pl/controlroomsoftware/app-cosylab-controlprogram.git
* Create a directory for GUIs
  + mkdir SSCP\_GUI\_DIR
* Copy all custom GUIs for managing Tango devices and device groups into the created directory

## Running the application

The Solaris Synchrotron Control Program can either be run manually, or by using a provided GUIRunner application. As mentioned, the use of GUIRunner application is advised.

### Manual start

* cd SOLARIS\_SYNCHROTRON\_CONTROL\_PROGRAM
* python ControlProgram.py

--CSV FACILITY\_CONFIGURATION/CSVFiles/synchrotron\_devices.csv

--GCSV FACILITY\_CONFIGURATION/CSVFiles/synchrotron\_groups.csv

--GUI SSCP\_GUI\_DIR

--TITLE “Solaris Synchrotron Control Program”

* + After the CSV flag, you must provide a path to the main device CSV configuration file. The same configuration file ought to be used as for the Facility Configuration software.
  + After the GCSV flag, you must provide a path to the group CSV configuration file. This flag is optional.
  + After the GUI flag, you must provide a path to the directory where all custom GUIs are located.
  + After the TITLE flag, provide a title of the application. This flag is optional.

### GUIRunner Application

GUIRunner application automatically downloads the source code of the Solaris Synchrotron Control Program, the CSV configuration files and the source code for developed custom GUIs from the dedicated GIT repositories for a selected control system instance. A repository for the Solaris Synchrotron control program is the same for all control system instances, but the location of the CSV files and custom GUIs differs. If the respective files already exist in the specified local directories, GUIRunner tries to update them to the newest version. By default, the following data structure is created:

|  |  |  |
| --- | --- | --- |
| **Linac and Storage Ring** | **Uarpess Beamline** | **Peem Beamline** |
| GUIrunner.py | GUIrunner.py | GUIrunner.py |
| ControlProgram   * ControlProgram.py * … | ControlProgram   * ControlProgram.py * … | ControlProgram   * ControlProgram.py * … |
| Synchrotron\_CSV   * synchrotron\_devices.csv * synchrotron\_groups.csv | UarpessBL\_CSV   * BL-05ID.csv | PeemBL\_CSV   * BL-04ID.csv |
| Synchrotron\_GUIs   * … | UarpessBL\_GUIs   * … | PeemBL\_GUIs   * … |

By default, the Linac and Storage Ring control system instance will be selected. A user can select the control system instance by providing a view input parameter flag specifying the desired control instance:

|  |
| --- |
| **Linac and Storage Ring** |
| python GUIrunner.py --view Synchrotron |
| **Uarpess Beamline** |
| python GUIrunner.py --view Uarpess |
| **Peem Beamline** |
| python GUIrunner.py --view Peem |

Altogether, GUIrunner supports the following optional input parameters for manual configuration:

|  |  |
| --- | --- |
| **GUIrunner input parameters** | |
| --view VIEW | VIEW must be of the elements: Synchrotron, Uarpess, Peem. With this parameter, a user can choose a control system instance that will be used. By default, if this flag is not present, a “Linac and Storage Ring” instance will be selected. |
| --noCheck | If this flag is present, GUIrunner will not try to clone or update the files from the repository. In this case, files must already be located in the corresponding locations on the local drive. |
| --repoCSV PATH | A user can manually provide a repository location of the corresponding CSV configuration file. |
| --localCSV PATH | A user can manually provide a location on local drive, where all the CSV configuration files will be/are stored. |
| --repoGUI PATH | A user can manually provide a repository location of the corresponding custom GUI scripts. |
| --localGUI PATH | A user can manually provide a location on local drive, where the custom GUI scripts will be stored. |
| --repoCP PATH | A user can manually provide a repository location of the Solaris Synchrotron Control Program. |
| --localCP PATH | A user can manually provide a location on local drive, where the Solaris Synchrotron Control Program will be stored. |

Table 1: GUIRunner input parameters

To run GUIrunner application, execute:

* cd GUI\_RUNNER\_DIRECTORY
* python GUIrunner.py INPUT\_PARAMETERS

# Configuration

Two different CSV files are used for configuring the ControlProgram. The former is referred to as the main device CSV configuration file. The latter is referred to as the group configuration file.

The CSV configuration files for Solaris Synchrotron are included with the Facility Configuration software.

## Device Configuration

All devices which are forming the synchrotron machine are described in a dedicated device CSV configuration file. This file is used as the master source for the FacilityConfiguration software to configure the control system database and for the Solaris Synchrotron Control Program. It is of the following format.

For more information refer to the Solaris signal list 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, “,”,  AGGREGATE\_GUI, “,”,  DESCRIPTION, “,”,  COMMENT; |
| MANAGED\_IN\_CS | Y | N; |
| TRIGGERED\_BY\_TTL | Y | N; |
| CUSTOM\_GUI | 0\*1(GUI\_SCRIPT\_NAME); |
| AGGREGATE\_GUI | 0\*1(SUBSYSTEM\_GROUP, “-”, DEVICE\_GROUP\_ID); |
| SUBSYSTEM\_GROUP | SUBSYSTEM\_NAME, SUBSYSTEM\_GROUP\_ID; |

Table 2: Device Configuration Format

|  |  |
| --- | --- |
| **Configuration Specifications** | |
| **Item** | **Description** |
| TYPE | Type of the component. |
| L | Length of the component. |
| S | Width of the component. |
| X | Distance from the gun in X axis. |
| Y | Distance from the gun in Y axis. |
| Z | Distance from the gun in Z axis. |
| SECTION | System code + location code where the element is placed (I-K00, R1-SGA, …). |
| SUBSYSTEM | Code of the subsystem that the device belongs to (MAG, VAC, …). |
| MANAGED\_IN\_CS | Must be set to “Y”, if the device is controlled by a TANGO device server; “N” otherwise. |
| DEVICE\_SERVER\_NAME | Name of the TANGO device server (executable) controlling this device. |
| DEVICE\_SERVER\_INSTANCE | Device server instance name controlling this device. |
| DEVICE\_CLASS | Name of the TANGO class used to control this type of devices. |
| FULL\_TANGO\_DEVICE\_NAME | Full TANGO device name for this device instance. |
| DEVICE\_ALIAS | Desired alias for a tango device. If this field is set, it will be used as a display name for the device. If it is not set, a FULL\_TANGO\_DEVICE\_NAME will be used as a display name instead. |
| TRIGGERED\_BY\_TTL | Must be set to “Y”, if the device has dependency on the timing system, or “N” otherwise. |
| CUSTOM\_GUI | If this field is set, Custom GUI will be launched instead of TaurusDevicePanel as Standard Device Panel. This field needs to have the name of the file to run without the .py extension. |
| AGGREGATE\_GUI | This field must be set, if the device is a part of aggregated GUI display. This field has a special format describing the aggregate GUI that the device belongs to and the component it represents. |
| SUBSYSTEM\_GROUP | Name of the subsystem group. |
| SUBSYSTEM\_NAME | Name of the subsystem. A string without numbers. An executable with this name will be launched as Device Group Panel. |
| SUBSYSTEM\_GROUP\_ID | Id of the group, integer number. |
| DEVICE\_GROUP\_ID | Id of the device within the group. |
| DESCRIPTION | Short description of the device. |
| COMMENT | Comment. |

Table 3: Configuration Specifications

### TRIGGERED\_BY\_TTL explanation

If the device is triggered by EVR’s TTL port, this field is set to Y. There is no need to put any additional information about which receiver or port is used since the import script will find it in EVR’s properties file and write this information to TANGO database automatically for each device instance that uses timing.

### CUSTOM\_GUI explanation

In case the particular device instance has a custom GUI, this field has to be set with the name of the python script file, without the .py extension. Leave this field empty if the TaurusDevicePanel should be used. When opening a device panel for the particular device, if the CUSTOM\_GUI field is filled, the script will executed. The Tango device name will be passed as an input parameter to the script.

**Example**

Suppose we have a device called “dom1/fam1/device1”, and the CUSTOM\_GUI field is set to **SOME\_SCRIPT**. When opening a standard device panel, the following will be executed:

* python SOME\_SCRIPT.py dom1/fam1/device1

### AGGREGATE\_GUI explanation

This field allows linking of several device instances to a subsystem group overview GUI. Two pieces of information need to be set:

* Which subsystem GUI the device instance belongs to
* Which part of GUI is the device linked to

This is done using the following format:

* **SUBSYSTEM\_GROUP – DEVICE\_GROUP\_ID**

**SUBSYSTEM\_GROUP** is the name of the aggregate GUI and subsystem it represents. Since some will have more groups, the name must have an enumerator at the end.

**DEVICE\_GROUP\_ID** is used to determine which placeholders for models of Taurus widgets to replace with specific device instance proxy in the subsystem group GUI.

The SUBSYSTEM\_GROUP will be identified as a Group GUI instance. When opening a certain Group GUI from the ControlProgram, the ControlProgram will prepare the inputs. For every device that has the corresponding SUBSYSTEM\_GROUP element specified, its DEVICE\_NAME will be passed as an input argument under the flag: “—DEVICE\_GROUP\_ID”.

**Example**

Suppose that we split the Linac in 2 MAG subsystem groups. First part is named MAG1 and second MAG2. Each MAG subsystem group has 3 power supplies.

First power supply of the MAG1 group will have the Aggregate GUI set to: **MAG1-PS1**

Second power supply of MAG1 group will have the Aggregate GUI set to: **MAG1-PS2**

Third power supply of the MAG1 group will have the Aggregate GUI set to: **MAG1-PS3**

First power supply of the MAG2 group will have the Aggregate GUI set to: **MAG2-PS1**

Second power supply of MAG2 group will have the Aggregate GUI set to: **MAG2-PS2**

Third power supply of the MAG2 group will have the Aggregate GUI set to: **MAG2-PS3**

Filename of the GUI in this case will be named: MAG.py

The script must implement a global function called “getGuiWidget” that returns a QDialog (please refer to the QT and PyQt4 documentation for information on QDialog).

The function will accept following arguments:

* --LAB, LABEL
* --PS1, DEVICE\_NAME1, DESCRIPTION1
* --PS2, DEVICE\_NAME2, DESCRIPTION2
* --PS3, DEVICE\_NAME3, DESCRIPTION3

Where LABEL is the label of the subsystem group (MAG1 or MAG2 in this example)

The “getGuiWidget” function will accept these arguments in a form of an array.

For some Group GUIs, additional configuration can be specified for every device.

In general, the following format was adopted:

* **SUBSYSTEM\_GROUP – DEVICE\_CONFIGURATION**

As in the previous example, the ControlProgram will assign the device to a specified instance of the Group GUI. When opening the GUI, the DEVICE\_NAME and DEVICE\_CONFIGURATION will be passed on to the Group GUI. However, opposed to the previous example, the DEVICE\_CONFIGURATION does not necessarily contain only the DEVICE\_GROUP\_ID, but can be any desired string. The format of the DEVICE\_CONFIGURATION depends on the implementation of the Group GUI. The ControlProgram does not try to parse it.

### Aggregated groups

A list of currently known subsystem groups is provided in order to facilitate the CSV file construction.

|  |  |  |
| --- | --- | --- |
| Aggregated codes | Device codes | Description |
| MAG# | PS# – Power Supply DS  THERM# - Thermocouple DS  MG# - Magnet DS | Magnet subsystems. # can be replaced by a number. |
| MAGG | PS# – Power Supply DS  THERM# - Thermocouple DS  MG# - Magnet DS | Magnet subsystem at the gun. # can be replaced by a number. |
| VAC# | VGM\_L - Left valve – valve at the start of the section  VGM\_R - Right valve – valve at the end of the section  IPC# - Ion Pump  VGC# - Vacuum Guage  PLC - PLC | Vacuum subsystem. # can be replaced by a number. |
| RF# | PLCB - PLC holding boolean values  PLCR - PLC holding real values  MOD# - Modulator  SCP# - Scope  SG# - Signal generator | RF subsystem. # can be replaced by a number. |
| DIA\_CAM | CCAM - Camera  SCRN – Yag screen | Diagnostic camera group. It aggregates only one camera and a corresponding yagscreen. |

Table 4: Aggregated subsystem and device codes

## Group configuration

Device Group Panels are custom made panels, built on a ControlProgram Group template. They provide access to a group of devices. The content of each Device Group Panel is dynamically generated, according to the input. Another feature that is supported within the scope of the Device Group Panels is a support for chaining the panels. For every instance of the Device Group Panel, one can define a set of preceding and succeeding instances of the Device Group Panels. Upon this information, buttons are generated on the left and right side of the respective Device Group Panel, which provide easy access to the specified instances of panels. Moreover, certain Device Group Panels require additional specific configuration that exceeds the scope of the main device CSV file.

In retrospect, for each instance of the Device Group Panel, an additional configuration is foreseen that includes the set of preceding and succeeding instances in the chain and an additional parameter for specific configuration. This configuration is kept in a group CSV configuration file, described below.

|  |  |
| --- | --- |
| **Group Configuration BNF Format** | |
| CSV configuration: | 1\*(GROUP\_SPECIFICATION) |
| GROUP\_SPECIFICATION | GROUP\_GUI\_NAME, “;”,  PRECEEDING\_INSTANCES, “;”,  SUCEEDING INSTANCES, “;”,  ADDITIONAL\_PARAMETER, “\n” |
| PRECEEDING\_INSTANCES | 0\*1(GROUP\_GUI\_NAME, 0\*(“|”, GROUP\_GUI\_NAME)) |
| SUCEEDING\_INSTANCES | 0\*1(GROUP\_GUI\_NAME, 0\*(“|”, GROUP\_GUI\_NAME)) |
| GROUP\_GUI\_NAME | Name of a desired Group GUI instance |
| ADDITIONAL\_PARAMETER | Any string. It can contain “\n”, if entire string is enclosed in quotes. |

Table 5: Group Configuration Format

It is not required that all instances of Group GUIs are specified in the Group configuration file. The instances are parsed from the device configuration. The group configuration provides additional configuration and is thus optional.

## Forwarding configuration

# Operation

The Solaris Synchrotron Control Program reads all required configuration from the given CSV file. The same CSV file is used for populating the Tango database. The FacilityConfiguration application is used for such purposes. The Solaris Synchrotron Control Program exposes the devices from the configuration in a form of a device tree and a device group tree, along with the functionality for easy access and control of the regarded Tango devices. The program also displays the status of every device. This is accomplished by subscribing to a change in state of every device. Additionally, a dedicated thread is implemented to update a state of every device that was not updated for a significant amount of time. The state of the device will be properly updated, even if the device, at any point, is not accessible. However, if a device is not specified in the Tango database at the initialization phase of the Solaris Synchrotron Control Program, but placed in the database later, its state will not be properly updated. The Solaris Synchrotron Control Program would have to be restarted. Such a use case is not foreseen.

## Device Tree

The device tree is the main element of the Solaris Synchrotron Control Program. It exposes Tango devices to the user. The tree is created according to the configuration in the CSV file. It consists of three levels, namely “Section”, “Subsystem” and “Device”, in the respected order.

The order of the elements in the tree coincides with the order of the device specifications in the configuration file. Devices within each Subsystem are ordered according to the order in the CSV file. The device that is specified before another device of the same subsystem is in the tree positioned prior to it. Subsystems within each section are ordered according to the position of their devices in the configuration file. The subsystem, whose device is in the configuration file specified before any device of another subsystem, is in the device tree positioned prior to it. Sections are also ordered according to the position of their devices in the configuration file. The section, whose device in any of its subsystems, is in the configuration file specified before any device of another section, is in the device tree positioned prior to it.

The displayed names of the tree elements coincide with the names specified in the configuration. Additionally, if for a certain device an alias is provided in the configuration, it would be used to present a device, rather than a full Tango name.

The state of every device is represented by a color field, left of every device in the device tree. Additionally, a color legend is provided at the bottom of the application.

In the second column, a short description for every device is displayed. If the description in the configuration file for a certain device is not provided, the field would be left blank.

Basic tools are provided for viewing the tree, namely expanding, collapsing and the selection of all devices. The selection of all devices is here meant as selecting all devices currently visible in the tree.

### Device Filtering

The device tree can be filtered using the included widget. The device tree can be filtered by section, subsystem or by a device class. All different filtering keywords are provided. Additionally a user can manually input a filter keyword. Note that it suffices that the keyword of a filter is only included in the device specifications and does not need to match entirely. A device will be positioned in the tree, if parts of the section, subsystem and the class names match the filters.

### Standard Device Panel

The Standard Device Panel is a dedicated device GUI, intended for a full control of a certain device. A custom GUI python script can be provided for it. In this case, in the configuration file where the device is specified, in the CUSTOM\_GUI column, the name of the script must be provided. By the name of the script here we refer to the name of the script without the python file extension. The script must be located in the GUI folder, whose path was used as an input parameter for the application. If such a GUI does not exist, the corresponding column in the configuration file must be left blank. In this case a default GUI (TaurusDevicePanel) will be used instead. In both cases, a device name is used as an input parameter for running a device GUI script.

To open a Standard Device Panel, click on a device in the Device View and click “Open Standard Device Panel” button. If multiple devices are selected, Standard Device Panels for all selected devices will open, but prompting the user if the number of selected devices exceeds three.

#### Script Input Parameters

When opening the Standard device panel, a python script is executed, provided with the following input parameters.

|  |  |
| --- | --- |
| **Standard Device Panel Script Input Parameters** | |
| **Default GUI script** | |
| TaurusDevicePanel.py | FULL\_TANGO\_DEVICE\_NAME |
| **Custom GUI script** | |
| CUSTOM\_GUI.py | FULL\_TANGO\_DEVICE\_NAME |

Table 6: Standard Device Panel Script Input Parameters

### Custom Panel

Custom Panel is a GUI that exposes selected attributes of selected devices. It is automatically generated according to the user’s selection.

To open a Custom Panel, select a desired number of devices and click “Custom Panel” button. A dialog will pop up, prompting the user to select attributes of selected devices to display. Select attributes and click “Open Custom Panel” button. You can also double click a particular device in order for its panel to open.

## Device List

Device list is an alternative option to the device tree. It exposes the same devices. The difference here is that the devices are not combined by subsystem or section, but listed individually instead. All the functionality is identical to the device group tree, apart from the order of the devices. The devices are ordered according to the order of the device specifications in the configuration file. Device, specified before another device, is positioned prior to it in the device list. This view is intended to give the user a list of devices in the actual order as they are positioned in the Solaris synchrotron. The state of every device is represented with a color field on the left side of every device, similar as it is in the device tree.

## Device Group Tree

Device Group Tree is an additional view that the Solaris Synchrotron Control Program provides. It exposes device groups. The tree only has one level. The elements of the tree are in alphabetical order, independent from the order in the configuration file.

Device groups are groups of devices, where for every group a dedicated Device Group GUI ought to exist, exposing all devices of the group within one panel. The device groups are within the application constructed according to the configuration. If a device belongs to a certain group, the group should be specified under the AGGREGATE\_GUI column of the device specification in the configuration file. Refer to the configuration section for details.

### Device Group Panel

The Device Group Panel is a GUI that exposes multiple devices within one GUI. All of Device Group GUI scripts are custom made.

The GUI python script for every group must be located in the GUI folder. The name of the script must match the SUBSYSTEM\_NAME. The script must include a global function called “getGuiWidget”. The function must return the QDialog object. When opening the Device Group Panel, the “getGuiWidget” function will be called, where the input parameters for the GUI will be passed to this function in an array form. The retrieved dialog will then be shown.

Multiple groups can use the same device group GUI script, that is when their SUBSYSTEM\_NAME matches, but the SUBSYSTEM\_GROUP\_ID differs.

To open a Device Group Panel, click on a device group in the Device Group View and click “Open Device Group Panel” button. If multiple devices are selected, Device Group Panels for all selected device groups will open, but prompting the user if the number of selected groups exceeds three. You can also double click a particular device group in order for its panel to open.

#### Input Parameters

Input arguments for a device group, which are passed to the function called “getGuiWidget”, are structured using the following format:

|  |  |  |
| --- | --- | --- |
| **Device Group Input Parameters** | | |
| For every device that belongs to a group, add: | --DEVICE\_GROUP\_ID, DEVICE\_NAME, DEVICE\_DESCRIPTION | |
| **Example** | | |
| **Configuration** | | |
| **Device** | **AGGREGATE\_GUI** | **DESCRIPTION** |
| dom1/fam1/device1 | MAG1-PS1 | PowerSupply1 |
| dom1/fam1/device2 | MAG1-PS2 | PowerSupply2 |
| **Group Parameters** | | |
| Group name | MAG1 | |
| Script name | MAG.py | |
| Input parameters | --LAB, MAG1, --PS1, dom1/fam1/device1, PowerSupply1, --PS2, dom1/fam1/device2, PowerSupply2 | |

Table 7: Device Group Input Parameters

## Profile management

Solaris Synchrotron Control Program supports the basic functionality for saving and loading a profile. With profile, we refer to a set of opened GUI, consisting of Standard Device Panels, Custom Panels and Device Group Panels. When monitoring a commonly used set of panels within the Solaris Synchrotron Control Program, it is useful to save a profile, to spare time when trying to open the same set of panels.

To save a profile, click “file” in the top window toolbar and select “save profile”. To open a profile, select “load profile”.