Sardana Configuration User Guide

May 7, 2008

Contents

1 Introduction

This guide is intended to give an overview of all the different components of the Sardana framework from the point of view of configuration. We briefly explain each part of the framework, and highlight those interesting concepts to take into account.

1.1 Sardana framework

The Sardana framework is designed to hide some internal control of a group of devices, while providing a wide, rich, and extensible set of operations that a user may need. It is composed of:

- a Device Pool Server = the server containing all the devices,
- a MacroServer = the server that asks the Device Pool Server to perform actions on devices,
- Doors = the user access points to the system.

1.2 Device Pool Server

The Device Pool Server is a DeviceServer that provides access to groups (pools) of generic hardware devices as motors or counters. The Pool is configured as any other Tango Device Server, so the configuration is stored in the Tango database. All devices within the Pool are controlled with specific "Pool controllers", so the stored Pool configuration contains also all the parameters for those controllers which are loaded automatically when the pool starts. The Pool can contain groups of devices and guarantees operations like synchronized movement of motors, synchronized data acquisition, or motor limit control. The Device Pool Server is event based.

The Device Pool Server core is developed in C++. The pool controllers can be written in Python or C++ and they will be loaded at pool start-up. The Pool is able to upload new versions of these controllers without restarting.

SardanaGUI is a tool used for the creation and configuration of the Device Pool Server. With this graphical user interface, you can add the desired controllers and devices to the pool. You can also configure the different groups of experiment channels of data acquisition for future use.

1.3 MacroServer

The MacroServer can be connected to different pools. It provides a group of macros (functions) to interact with the hardware through the Pool. The MacroServer communicates with the pool using TANGO. It uses events as well. The MacroServer is also defined as a device in the Tango database.

1.4 Door

A door is another device server that will be started by the MacroServer. The door allows users to connect to the MacroServer. The communication between the user application (client) and the MacroServer is based on TANGO and events. The door is an access point to the MacroServer so from the MacroServer point of view, there is no difference between a CLI (Command Line Interface) and a GUI (Graphical User Interface).

Spock is an application that offers a way to connect to a MacroServer using a door. It is just a Python script that starts an IPython session and offers the macros defined in the MacroServer. Spock needs a profile with the values of the door device server and the MacroServer device server.

2 Getting the code

The code is in the svn repository. It is also accessible via the blissinstaller packages. It has been already installed on controls01.

2.1 Source code

You can checkout the code from the svn server:

svn+ssh://user@controls01/siciliarep/svn/trunk/tango_ds/Miscellaneous/pool
svn+ssh://user@controls01/siciliarep/svn/tags/POOL_DEVEL_RELEASE/pool

2.2 SardanaGUI

The SardanaGUI is currently developed in java and the code is at:

svn+ssh://user@controls01/siciliarep/svn/trunk/tango_client/sardanaGUI

2.3 Dependencies

If you want to compile the pool, you will need:

OmniORB >= 4.1.0
OmniNotify >= 2.1
Tango >= 6.0.0
PyTango >= 3.0.3
Python >= 2.4
gcc >= 3.4
automake >= 2.6.1

libtool (installed by default)

2.4 Default controllers

There are some pool controllers already available as blissinstaller packages, for example an Icepap motor controller or the National Instruments 6602 DAQ Counter/Timer card.

3 Creating the Device Pool Server (SardanaGUI)

The SardanaGUI application can be launched with the 'java -jar sardana.jar' command, or installing the sardana blissinstaller package. It uses the TANGO_HOST environment variable to connect to the Tango database and retrieve all defined Device Pool Servers. SardanaGUI is an application under development and it still has a lot of features to be implemented. There are also known bugs that have to be fixed. All the actions that can be performed using the SardanaGUI are also available from Jive.

3.1 Using SardanaGUI to create a Device Pool Server

Right-click on the TANGO_HOST node to create a New Pool (you can also use the Create menu option). It asks you to fill in the Instance name (if you are doing tests, you can use your username for that instance name) and it automatically creates a Pool device name and an alias. As mentioned above, the Pool is meant to load controllers to access real hardware. In order to do so, it uses code taken from outside the core. In order to load the code the Pool needs the PoolPath property set with the directories where the code is located. If the Pool software has been installed in $POOL_HOME, then the directories to be added are:

$POOL_HOME/lib/pool (for the C++ code)

$POOL_HOME/lib/python2.5/site-packages (for the Python code)

Note:If you want to use the already installed Pool software in controls01, the C++ controllers directory is

/homelocal/sicilia/ds/suse102/pool/lib

and the Python controllers directory is

/homelocal/sicilia/ds/suse102/pool/python/site-packages

Note2: After creating the pool, use Jive to be sure that the path is only one line with directories separated by ':' instead of new lines.

4 Adding controllers (SardanaGUI)

Right-clicking on the controllers leaf of the pool node, you can create a new controller. Controllers must be created before hardware devices are added. The Controllers leaf has a tab called Controller Classes. Here you should be able to see all available controllers for the pool. If the controller you want to add is not

shown, check that the PoolPath variable includes the directory your controller is in.

4.1 Interfaces defined for controllers

The Pool recognizes the following types of controllers:

- Motor, Counter/Timer, and Communication as real hardware controllers
- PseudoMotor and PseudoCounter as software controllers.

For each type of controller, there is a specific interface developers have to implement. In this way, a hardware abstraction layer is provided to the Pool, which can access them as generic devices.

4.2 Controller properties

Each controller can define specific properties for its execution environment, as for example, a connection to a specific device server. It is also possible to define specific properties for each device added to the controller, for example specific values related to the hardware used.

4.3 Motor Controller

The motor controller must implement an interface for moving and for communicating the state of the motors it controls. For example, if more than one motor of the same controller must be moved at one time, the pool will notify the controller to move them both at the same time in case the hardware accepts a multiple move command.

4.4 CounterTimer Controller

The pool uses a CounterTimer controller in order to read data from counter cards. Pool clients must specify which hardware channels will be used for measurements. They do this by defining experiment channels in the controller.

An experiment channel is just a generic term the pool uses for data acquisition channels. For now, it can refer to a counter channel or a zeroD channel. The data is read from or written to experiment channels.

Experiment channels can be grouped in measurement groups for different measurement tasks. The measurement group must have a master channel defined, which can be either a timer or a monitor counter. The time to count or the number of pulses the monitor must count have to be set before starting the measurement.

Obs.: The information in this subsection might be incomplete. Please contact Tiago if you need more details!

4.5 Communication Channel Controller

Defining a communication channel controller allows you to have communication devices inside the pool for connecting controllers to other devices. The communication device has the open, write, read and close commands.

4.6 Pseudo Motor Controller

A pseudo motor controller provides motor devices that are not real.
The gap+offset example for the slits could be added here

4.7 Pseudo Counter Controller

The pseudo counter controllers add software data acquisition channels to the pool. These channels can be based on other devices. For example, if you have the counters CT1 and CT2, you can define a pseudo counter CT3 which value is CT1/CT2.

4.8 Adding hardware devices in the controllers (SardanaGUI)

Any hardware device that has to be available through the pool must be added to a controller. Some controllers may have declared specific properties for each device, so this information should be provided when you create the device. All the devices will be defined within the pool device server, and the user may use the MacroServer to operate with them.

5 Groups of Hardware (SardanaGUI)

The pool allows to create groups of hardware. For example, it is possible to create a group of motors to move them in a synchronized manner, or define a measurement group that includes a timer and some counters channels, so you can get the data from all the channels in the group at the same time.

6 Running the Pool

To run the pool you just need to execute the Pool binary followed by the instance name you want to use.

7 MacroServer

The MacroServer is an engine that offers client connections to perform actions on the Pool devices. The different actions are called macros, and should cover the average user needs to move and acquire data.

7.1 Creation (Jive)

To start a MacroServer you need to use Jive to create a new the device server. The ServerName is "MacroServer", you can use any instance name, the Class is also "MacroServer" and only one device is needed. Once the device is created, there are three properties that must be defined:

- MacroPath, which should point to the directory where the macros are stored (for now it only reads the standard.py macros file). In the distribution, the macro directory is $POOL_HOME/lib/python2.5/site-packages/macroserver/macros. Using the blissinstaller package, the path is /homelocal/sicilia/ds/suse102/macroserver/macros.
- MaxDoors is the number of doors that your MacroServer will accept.
- PoolNames is a list of pool device names that the MacroServer will connect to. At the time of writing this guide, the MacroServer can connect to only one pool.

7.2 Running

To run the MacroServer, you just need to execute the MacroServer script followed by the instance name you want to use.

7.3 Macros (built-in functions)

The macros available from the macroserver allow users to interact with pool devices. The macro list will be always growing & users will be able to define their own macros. Here are some common macros the MacroServer offers for the time being:

- ascan - Do an absolute scan of the specified motor
- ascan_mot - Do an absolute scan of the specified motor appending motor positions as counters
- ct - Count for the specified time on the active measurement group
- defmeas - Create a new measurement group
- load - Load a macro file
- lsenv - Lists all current environment
- lsmeas - List existing measurement group
- mesh - Do an absolute mesh scan of the specified pair of motors
- motor_par - sets configuration parameters for motor 'motor'. Generic recognized values for the string 'par' follow.

  - "Step_per_unit" returns the current step-size parameter. The units are generally in steps per degree or steps per millimeter. If val is given, then the parameter is set to that value.
  - "Acceleration" (R/W) returns the value of the current acceleration parameter. The units of acceleration are the time in milliseconds for the motor to accelerate to full speed. If val is given, then the acceleration is set to that value.
  - "Decceleration" (R/W) returns the value of the current deceleration parameter. The units of deceleration are the time in milliseconds for the motor to stop. If val is given, then the deceleration is set to that value.
  - "Base_rate" (R/W) returns the current base-rate parameter. The units are steps per second. If val is given, then the base rate is set to that value.
  - "Velocity" (R/W) returns the current steady-state velocity parameter. The units are steps per second. If val is given, then the steady-state velocity is set to that value.
  - "Backlash" (R/W) returns the value of the backlash parameter. Its sign and magnitude determine the direction and extent of the motor's backlash correction. If val is given, then the backlash is set to that value. Setting the backlash to zero disables the backlash correction.
  - "Controller" (R) returns the name of the controller for the given motor.

- mv - Move motor(s) to the specified position(s)
- mvr - Move motor(s) relative to the current position(s)
- pwa - Show all motor positions in a pretty table
- pwm - Show the position of the specified motors in a pretty table
- senv - Sets the given environment variable to the given value
- set_lim - Sets the software limits on the specified motor
- set_lm - Sets the dial limits on the specified motor
- set_pos - Sets the position of the motor to the specified value
- settimer - Defines the timer channel for the active measurement group
- uct - Count on the active measurement group and update
- umv - Move motor(s) to the specified position(s) and update
- umvr - Move motor(s) relative to the current position(s) and update

- wa - Show all motor positions
- waenv - Displays the current environment
- wenv - Displays the current value of the given environment variable
- wm - Show the position of the specified motors

Note: For a more detailed description of each macro, you can retrieve the on-line documentation from the spock command prompt.

8 Doors

As the name explicitly says, a door a device server is used to access the MacroServer. The door allows the user to invoke actions on the pool devices. When the MacroServer starts, it starts all the doors defined under the device server.

8.1 Creation (Jive)

To create a door you must use Jive and define, under the same instance name of the MacroServer, the new door device. Proceed as follows:

- ServerName should be MacroServer,
- Instance name, whatever you want
- Class is Door
- add as many devices as you want.

Each Door device must have a property called MacroServerName defined. This is the device name of the MacroServer you want to use.

9 Spock

Spock is a script that acts as a Command Line Interface MacroServer client. It uses IPython to interact with the user. It allows the user to ask the MacroServer to perform the macros. These macros write the results to the Output attribute of the door, to a file, or in shared memory.

Using the Spock client you can execute macros, and in the future, create new macros or modify existing ones. There are some environment variables that allow you to select a measurement group from the list for a particular data acquisition session.

9.1 Profiles

Spock uses profiles (like the IPython profiles) to connect to different doors. These profiles are also user-related, so each profile can have a specific configuration. If you want to create a new profile just execute "spock -p myProfile", it will list the available doors and you can choose the one you want.

9.2 Execution

You can start the connection to a door by executing "spock -p myProfile". If no argument is given, the default profile "spock" will be used and if it is not available, one will be created.