Sardana pool device management

Tiago Coutinho, Emmanuel Taurel

August 9, 2006

Contents

1 Introduction **1**

2 Creating device **1**
2.1 Retrieving object parameter value 2
2.2 Defining the object parameters 2
2.3 Retrieving the object parameter list and informations 3
2.4 Device creation 3
2.5 Work to be done 4

3 Extending the device pool **4**
3.1 Dynamic pool extension 4
3.2 Adding/Removing a new Tango class to the device pool 5
3.3 Problems to be solved and investigation to be done 5

1 Introduction

This paper describes

- A possible way to implement Tango device classes embedded within the Sardana pool in order to have an easy and user friendly graphical interface to create new devices.
- A possible way to dynamically extend the set of devices hosted by the Sardana pool.

2 Creating device

The principle described here is very similar to what already exists for the Tango wizard available from Jive. It is a not a precise definition of what need to be implemented. It is rather a description of the philosophy used during a pool managed device creation.

Every entity hosted by the pool can be configured by what will be called **parameters** later on in this document. For simple Tango class, these parameters are the classical Tango properties. For objects like motor controller or the pseudo motor Tango device, it can not be classical Tango property because a motor controller is not a Tango device and even if a pseudo motor is a Tango device, each pseudo motor implementation could be different and therefore could request different number/types of parameters. Nevertheless, every motor controller belongs to a C++ or Python class and every pseudo Motor also belongs to a Python class (which is not the Tango class). Therefore, it is possible to have a way to define these parameters very close to the way used by classical Tango property.

2.1 Retrieving object parameter value

Parameter value can be defined at the class level and/or re-defined at instance level. It is also possible to define parameter default value at class/instance level. These default value are stored within the class code. If the default value is not adapted to specific object instance, it is possible to define a new parameter value which will be stored in the Tango database. Tango database allows storing data which are not Tango device property. In Jive, these type of data are under the "Property" leave in the main tree. This storage could be seen simply as a couple name/value. Naming convention for this kind of storage could be defined as

class->param: value or class/instance->param: value

The calls necessary to retrieve/insert/update these values from/to the database already exist in the Tango core. The algorithm used to retrieve a parameter value is the following:

```
/IF/ Parameter has a default value defined at class level
    - Parameter value = class default value
/ENDIF/

/IF/ Parameter has a value defined in db at class level
    - Parameter value = class db value
/ENDIF/

/IF/ Parameter has a default value defined at instance level
    - Parameter value = instance default value
/ENDIF/

/IF/ Parameter has a value defined in db at instance level
    - Parameter value = instance db value
/ENDIF/
```

Using an example, the following array summarizes the result of this algorithm. The example is for an IcePap controller and the parameter is the port number (called port_number)

	case 1	case 2	case 3	case 4	case 5	case 6
class default	5000	5000	5000	5000	5000	5000
class in DB				5150	5150	5150
inst. default			5100		5200	5200
inst. in DB		5200				5250
Parameter value	5000	5200	5100	5150	5200	5250

Case 2 : The IcePap controller is created with an instance name "My_IcePap". The property IcePap/My_IcePap->port_number has been set to 5200 in db
Case 4: The hard coded value of 5000 for port number does not fulfill the need. A property called IcePap->port_number set to 5150 is defined in db.
Case 6: We have one instance of IcePap called "My_IcePap". This case need the definition of 2 properties in db

1. IcePap->port_number set to 5150
2. IcePap/My_IcePap->port_number set to 5250

2.2 Defining the object parameters

In order to have a user-friendly interface, all the parameters used by the object class has to have informations hard-coded into the object class code. We need at least three informations and sometimes four for each parameter. They are:

1. The parameter name
2. The parameter data type restricted to the data type available for Tango property
3. The parameter description
4. The parameter default value (if defined)

With these informations, a graphical user interface is able to build at object creation time a panel with the list of all the needed parameters, their descriptions and eventually their default value. The user then have the possibility to re-define parameter value if the default one is not valid for his usage. This is the rule of the graphical panel to store the new value into the Tango database. For Python classes (Python controller class and pseudo motor class), it is possible to define these parameter informations using a Python dictionary with a well defined name. The parameter name is the dictionary element key. The dictionary element value is a list with two or three members which are the parameter data type followed by its description and an optional default value. For instance, for our IcePap port number parameter, this dictionary could be

cl_param_dict = {"port_number":[PyTango.DevLong,"Port on which the IcePap software server is listening","5000"]}

Concerning instance level parameter definition, a second similar dictionary has to be used. Something similar has to be implemented for C++ controller. For classical tango classes, these kind of informations are stored in the Tango wizard which is initialized by some code automatically generated by Pogo when the class is created.

2.3 Retrieving the object parameter list and informations

The pool already has a command called "GetPseudoMotorClassInfo" which from a pseudo motor class name returns miscellaneous informations related to this pseudo motor class. This command could be updated to return the parameter list with their informations. Nothing is defined yet for motor controller and this need to be added. For classical Tango classes, the Tango admin device already has 2 commands to retrieve Tango wizard data. These commands are called "QueryWizardClassProperty" and "QueryWizardDevProperty"

2.4 Device creation

As a Tango device server, the pool has a Tango admin device which already has a command called "QueryClass" which returns the list of all Tango classes embedded in the process. This command can be used as the starting point of the procedure used to create a device managed by the Sardana pool. The complete procedure could look like:

1. Query the list of Tango class embedded with the pool (QueryClass command)
2. Display this list to the user who select one of these class
3. If it is the first time a device of this class is created (a db call will give us this information)
   (a) If it is a Tango class, ask the wizard for ClassProperty info
   (b) If it is a Pseudo Motor, display the list of all pseudo motor classes available. The user selects one and with the GetPseudoMotorClassInfo pool command retrieve pseudo motor class parameters and information
   (c) If it is a motor, display the list of available motor controller. The user selects one and with the xxxxx pool command, retrieve controller class parameters and information
   (d) Display all the parameters defined at class level with their default value
   (e) If the user has modified one of the default value, store the new value in the Tango DB

4. Ask for instance level parameters
   (a) If it is a Tango class, ask the wizard for DevProperty info
   (b) Use the GetPseudoMotorClassInfo command result to get instance level parameters
   (c) If it is a motor, retrieve controller instance parameters with the xxxxx pool command
5. Display all the parameters defined at instance level with their default value
6. If the user has modified one of the default value, store the new value in the Tango DB
7. Create the device using the appropriate pool command

2.5 Work to be done

To implement this kind of behavior, we need to

- Implement the algorithm to retrieve parameter value for pseudo motor and motor controller
- Implement the way to define parameter information in Python class and in C++ class (for C++ controller)
- Modify the pseudo motor GetPseudoMotorClassInfo that it also returns the parameters list and informations
- Add to the pool a new command like GetPseudoMotorClassInfo but dedicated to motor controller
- Add a new pool command which returns the list of available motor controller class. Two possible ways (We need to decide which one to implement):
  1. Return only the list of class of the already created controllers
  2. Return a list of motor controller class (C++ and Python) found in a specific directory even if no controller of these class is created yet (similar to what is implemented for Pseudo-Motor)

3 Extending the device pool

This chapter is much more "experimental" than the previous one. It is simply a theoretical proposal which need to be tested and validated with small examples if it is judged as a desirable feature.

3.1 Dynamic pool extension

The Sardana device pool will embed a pre-defined list of Tango classes for classical devices found in a typical beam line hardware. Typically, the pool will embed classes for

- Motor
- Pseudo motor
- Motor group
- Serial line
- GPIB
- USB ?

and certainly some other. This defines the pool core. Nevertheless, from time to time it could be useful to extend this list in order to manage new device type which could be specific from one beam line to another. Using dynamic loader features, it should be possible to do this without need to compile/link the pool device server.

3.2 Adding/Removing a new Tango class to the device pool

We need to add three new pool commands to implement this feature. These commands are:

- AddClass: The input parameter is the class name (assuming the shared library file name is also the class name). The instance name associated to the specific instance will be the pool instance name. This command will automatically create two new commands and one attribute on the pool device. They are:
  - CreateXXX: This command will allow the user to create one device of the added Tango class
  - DeleteXXX: This command will allow the user to delete one device of the added Tango class
  - XXXList: One Read/Spectrum/String attribute which is the list of already created device of the added class
- RemoveClass: The input parameter is the class name. This command unload the specified class from the device pool. This is possible only if all the devices belonging to this class have been first deleted.
- ReloadClass: The input parameter is the class name. This command un-validate all the devices belonging to the class, unload the class, re-load it and validate class devices. This should be the equivalent of a device server process restart.

The information that a class has been dynamically added to the pool will be memorized and the class (and all its devices) will be part of the pool at the following pool restart.

3.3 Problems to be solved and investigation to be done

Some problems generated by this approach are mainly due to the fact that everything will be managed by the same operating system process. What will happens if a class dynamically added has a bug which generates a "core dump" ? Is it possible to make the pool resistant to such an event and to invalidate the class and all its devices if this kind of failure happens ? The same kind of problem could happens if a dynamically added class has a memory leak making the complete pool unavailable ?

Some investigation also need to be done in order to dynamically load Python Tango class. Is it possible and how ?