ACCESING HARDWARE
Commands and Channels

Antonia Beteva (ESRF)

ACCESSING HARDWARE – COMMADS AND CHANNELS

Scope and usage:

- Provide standard API to access the hardware that makes an abstraction of the underlying control system or communication protocol.
- Supported systems/protocols: Exporter, Tango, Sardana, Epics, Tine, Taco, SPEC.
- Commands are “actions”:
    - used when an action takes place (by default not instantaneous)
    - there is a need to specify timeout.
    - more than one value as input parameter.
    - have at least __call__ method
    - emit signals "commandBeginWaitReply“, "commandFailed“, "commandReplyArrived“
- Channels:
    - have similar usage as the python @property – get/set a value.
    - have at least get_value and update methods.
    - emit signal “update” when value changed.

ACCESSING HARDWARE – COMMADS AND CHANNELS

Implementation:

- CommandContainer class has add_command and add_channel methods with footprint (dictionary, string).
- the **type** key of the dictionary defines the protocol (control system). The rest of the parameters depend on the specific control system implementation. The string parameter is the command or channel name.
- Specific control system implements its Command and Channel classes, which inherit from CommandObject and ChannelObject. Common methods (like get_value) can be implemented.
- Optional Client class to handle events and protocol internals.
- All is grouped in the Command directory.
- MockupCommand and MockupChannel classes for testing.
- HardwareObject class inherits from CommandContainer, so any hardware object has add_command and add_channel methods.

ACCESSING HARDWARE – COMMADS AND CHANNELS

Implementation – Expporter Protocol example:

- Exporter is a protocol developed at EMBL (Grenoble) to interface their equipment control application. It is a socket connection to Java Application. The application provides commands and properties.
- Exporter Command and ExporterChannel are the two classes, which translate the exporter commands and properties to mxcubecore commands and channels.
- Exporter class is the interface that allows to connect to the application so there is only one connection. It defines the execute method to be used by the ExpoterCommand and get/set_property to be used by the ExporterChannel. Also defines method to register to callbacks and another to translate the exporter binary data to python values.
- Exporter class inherits from ExporterClient. ExporterClient is the class where the low level methods are defined. It deals with the input/output to the application, the errors and data handlin and the evnts to subscribe to.
- From a GardwareObject it can be used two ways – commands and channels (like in Microdiff) of instantiating the Export class and use its methods (like in ExporterNState or ExporterMotor).

ACCESSING HARDWARE – COMMADS AND CHANNELS

Exporter and Tango examples – 2 ways of doing the same:

- Add exporter command “startSetPhase”

```
_exporter_addrress = self.get_property(“exporter_address“)
self.move_phase = self.add_command(
     {"type": "exporter", "exporter_address": _exporter_addrress,  "name“: "move_to_phase",},
     "startSetPhase",)
```

- Add exporter channel “AperturePosition”

```
value_channel = self.get_property("value_channel_name")
self.value_channel = self.add_channel(
     {"type": "exporter", "exporter_address": _exporter_address, "name": value_channel.lower(),},
     value_channel,)
```

- Configuration in xml file

```
<exporter_address>microdiff:9001</exporter_address>
<value_channel_name>AperturePosition</value_channel_name>
```

- Add reading of a tango channel “Current”

```
_chan = self.get_channel_object("Current")
_chan.connect_signal("update", self.value_changed)
value = _chan.get_value()  -> methog get_value defined in the TangoChannel class
```

- Configuration in xml file

```
<tangoname>fe/id/23</tangoname>  -> tangoname defined in TangoChannel class
<channel type="tango" name="Current" polling="2000">SR_Current</channel>
```