MACRO PACKAGE BASED ENHANCEMENT OF SPEC CONTROLLED EXPERIMENTAL SETUPS

Thomas Spangenberg*), Karlheinz Cerff, Wolfgang Mexner
Institut for Synchrotron radiation, ISS, ANKA, KIT-Campus North, Karlsruhe, Germany

Abstract

Certified Scientific Software's program package spec [1] for X-Ray diffraction and data acquisition provides reliable instrument control to scientists at synchrotrons and other facilities worldwide. It's very flexible C-like macro language provides a large number of degrees of freedom for experiment control as advantage and as big disadvantage at the same time. A large number of programmers with their own ideas and naming conventions are contributing to the growth of functionality. At the same time the risk of collateral damage by accidentally overriding already existing functions and variables grows constantly. To solve this dilemma a new object oriented like software development concept for spec is proposed. A few naming rules plus a macro package in combination with a single client-server-application expand the manageability and options to control experiments considerably. As main goal spec gets an object-like handling and a standardized user interface of newly introduced devices. A generic server-client based interface allows a smooth integration of spec in more complex control environments via TANGO [2].

INTRODUCTION

Most of the physical and logical devices provides the opportunity to operate them in a simplified model as a set of independent properties which are offered by a certain remote interface. Therefore it becomes possible to integrate them rapidly into its own measurement setup either by direct driver support or by some macro integration.

As an example, the software package SPEC with its flexible macro language and various interfaces offers a number of paths to implement additional hardware into an experiment.

It will be shown that the risk of interfering solutions can be avoided for the device integration by introducing a few design rules in combination with a macro package. Additionally the client server based export possibilities of the integrated devices will be increased significantly.

MACRO PACKAGE AND DATA STRUCTURING

The basic idea of that macro package is to organize and handle devices object like although SPEC's pure macro based programming language definition doesn't support objects directly. But the provided data structures permit with a few limitations an object like structuring of data and a macro supported creation of specific functions to manipulate them.

Starting from the abovementioned simplified device model the representation of the device properties is stored into SPEC's associative arrays (see Fig. 1) which yields three advantages.

- First, all objects of one class are stored in only one array variable. It is evident, that a naming conflict can be prevented by using a single identifier per class.
- Second, due to SPEC's data type definition any type of data can be stored into this array.
- Third, the two dimensional index organized by strings is well suited to store data differentiated into 'objects', their properties, and their methods.

The data organization of the macro package is basically funded to associated arrays and is introducing a naming convention to their indices. SPEC defines associative arrays as a string indexed data object which stores any type of information. The first dimension of the two dimensional index is used for the device name. The name is usually chosen as an acronym which describes the device function in the experiment (e.g. vc1 for vacuum controller 1, see fig. 1).

The second part of the index string is primary subjected to the device property. Additionally the first character is used to transport the minimal necessary information about the represented property which is used for the automatically generation of the user interface. The implemented scheme is as follows:

- '$' indicates internal variables. There are no user functions provided.
- '*' indicates read only properties or variables. Read functions are provided.
- '!' indicates a command. A command function will be available.
- no special character indicates a read/write property. Read and write functions are provided.

The formal initialization overhead due to the macro package is very small. There are 2 functions for the whole macro package and only 3 additional steps are needed to implement a new device. There are:

- The formal declaration of the device instance by name and device type, followed by
- The initialization and declaration of start-up values and finished by
- Initializing the device or synchronizing the stored information.

The macro package evaluates the stored data and creates automatically the functions to manipulate them obeying the fixed naming scheme. Thereby the whole user functionality will be generated.

*) thomas.spangenberg@kit.edu

EXAMPLE IMPLEMENTATION

An example may demonstrate the situation. Assuming a hypothetical vacuum pump controller (similar as shown in fig. 1) device vc1 which may store its data into the associative array VPC. The declaration of that structure is done by global VPC while loading the macro package for that type of controller.

private:	address
r/o:	current
r/w:	voltage
command:	on/off

Figure 1: an example vacuum pump controller and its properties

As it is common to object orientated approaches a set of 'class-functions' needs to be provided by the controllers macro package. The first argument of all functions is the device name. Other arguments are regulated and straight forward connected to the idea of the object like access and the simplified device model approach.

The value initialization is done by VPC_standardvalues(device, [arg1 [,arg2...]]). This special function (see fig. 2) doesn't have strong naming rule because it will never be used automatically by the package and depends of course from the device which is to be implemented.

The task of the function is comparable to a constructor of an object. It has to pre initialize all instance variables and at the same time it is declaring the user interface functionality due to the fixed naming scheme abovementioned.

```
def VPC_standardvalues(device,address) '{
VPC[device]["$adress"] = address
VPC[device]["*current"] = 0
VPC[device]["voltage"] = 0
VPC[device]["!on"] = "VPC_poweron"
VPC[device]["!off"] = "VPC_poweroff" }'
```

Figure 2: example device value initialization

Furthermore current implementations of the macro package expecting the functions VPC_init(device) which drops all pre setted values into the device, VPC_sync(device) to synchronize the object to the device otherwise, and VPC_state(device) which is printing the read device state onto the screen.

Declared commands are realized by any function which has to handle 2 arguments. The first is the device and the second is the optional user argument. An example may be VPC_poweron(device,option), which name was stored into the device property '!on'.

For reading and setting the property XYZ the functions VPC_readXYZ(device,..) and VPC_setXYZ(device,..) need to be defined.

The read and set functions have to provide some other arguments which will be discussed following.

FUNCTION ARGUMENTS

The macro package requires from all 'class-functions' a strict organization of all arguments concerning their order and the meaning. The first argument is always the device name.

Reading and setting functions are already differentiated by the second argument which is for reading functions an integer indicating the verbosity of it. The complete declaration of the example read function is as follows VPC_readXYZ(device,verbose).

The argument verbose regulates the verbosity which can be switched on or off.

Setting functions using as a second argument an integer which lets them operate quiet. In that case the third argument represents the value to be set. The declaration is therefore

VPC_setXYZ(device, quiet, value)

It is quiet clear that these 'driver class functions' needs to be programmed with respect to the device and are therefore similar to other approaches in relation to the necessary programming effort. The goal are the generated user interface and the export capabilities.

USER INTERFACE

Just offering to the user device view orientated functions doesn't satisfy the users view to an experiment, which is usually more orientated to the job that needs to done than to a certain device.

The macro package evaluates automatically the array stored information (the index names and '$*!') and builds the whole set of corresponding functions and macros which are representing the user interface for any device. Even different user custom is satisfied by creating a function based access and a macro based access as well at the same time.

The created set for the example is shown in table 1:

Table 1: corresponding set of device functions and generated user functions

device function	**user function / macro**
VCP_state("vc1")	blstate_vc1 blstate vc1
VCP_readXYZ("vc1",...)	blread_vc1_XYZ(...) blread vc1.XYZ
VCP_setXYZ("vc1",...)	blset_vc1_XYZ(...) blset vc1.XYZ
VCP_poweron("vc1")	blcmd_vc1_poweron blcmd vc1.poweron

It is obviously that the hardware specific part VCP is eliminated from the user functions or macro calls. Therefore any device with similar options maybe exchanged without big incidence for the user.

The argument structure may appear rather complicated but the macro package derived functions offering a verbose interface to the user as well as a quiet device interface for further macro programming at the same time. Furthermore increases the clear and straight forward command structure for any implemented device the user acceptance and comprehension.

In case of reading a certain property the user may type blread_vc1_XYZ(1) or blread vc1.XYZ to get a print out of the current value. Otherwise any macro may use blread_vc1_XYZ([0]) to obtain the value of the property returned silently. The 0 is optionally because a not set argument is implicitly set as 0.

On the other hand blset_vc1_XYZ(1,3) sets the value 3 silently to the device property and the use of the argumentless version blset_vc1_XYZ() indicates the request for a user dialog. The macro versions of the same functions are blset vc1.XYZ 3 and blset vc1.XYZ respectively.

DEVICE EXPORT

SPEC supports among other things the export of variables and arrays and furthermore the remote execution of code by a socket connection. This server functionality is well developed but isn't SPEC's main goal. Some care is advisable concerning the bandwidth of a single socket connection and therefore the strategy for data exchange influences the benefit.

The internal structure of the devices organized by the macro package, as stated before, is concentrated in two arrays which stores the basic set of information about a device. The name and the name of the device class array can be obtained and therefore the whole information set maybe derived in a second step. Observing and exporting these two arrays into a client application offers the option to derive the complete state information about all macro package managed devices if additionally the device class arrays are obtained as well.

This approach minimizes the total number of variables to be observed by the client and the run-time influence of the steady client-server connection. Only 2 + N variables needs to be tracked.

The realized client itself is designed as a TANGO server. The first one offers a generic access to all macro package devices too.

Due to the strict data organization the TANGO-server can offer a generic and complete interface to access any property for reading and writing (if applicable). The generic functions are string based and schematically (the original TANGO calls are a bit less instructive) defined as follows:

- string SPECgetdevices();
- string SPECgetproperties(string device);
- string SPECblread(string device, string property);
- void SPECblset(string device, string property , string value);
- void SPECblcmd(string device, string command);

All reading interface functions are operating with a buffered and automatically updated data base. Settings and commands are scheduled into SPEC's command queue

CONCLUSIONS

The introduced macro package in combination with a few naming rules offers a straight forward approach to an object like access for device implementations with SPEC's macro language. Unwanted variable cross talking is maximally avoided and a systematic macro generated user interface can be provided at the same time.

The whole functionality can be exported into a socket client which offers itself a TANGO server for the SPEC macro package managed devices and permits a remote control of them by other programs

REFERENCES

[1] http://www.certif.com
[2] http://www.tango-controls.org