J. Synchrotron Rad. (1998). **5**, 593–595

Object library for a new generation of experiment-controlling applications under the UNIX operating system

Yu. A. Gaponov,[a]* K. Ito[b] and Y. Amemiya[c]

[a]Siberian Synchrotron Radiation Center, Budker Institute of Nuclear Physics, Institute of Solid State Chemistry, Lavrentyeva 11, Novosibirsk-90, 630090 Russia, [b]Photon Factory, National Laboratory for High Energy Physics (KEK), 1-1 Oho, Tsukuba, Ibaraki 305, Japan, and [c]Engineering Research Institute, School of Engineering, The University of Tokyo, Yayoi, Bunkyo, Tokyo 113, Japan.
E-mail: gaponov@inp.nsk.su

(*Received 4 August 1997; accepted 3 December 1997*)

The Interface Object Library based on the Motif extension of the X Windows system and on the ESONE SVIC-VCC Library is presented. Some features of the applications for controlling a synchrotron radiation experiment are discussed. The Interface Object Library is written in the object-oriented C++ language. The library class-hierarchy structure is presented and discussed. Several interfaces were realized in the Interface Object Library: the Windows interface, the CAMAC interface and the interface for supporting the experiment. The behaviour of the objects describing the CAMAC crate and CAMAC block is discussed. The application of these protocols for controlling the fast one-coordinate position-sensitive X-ray detector OD3 is presented.

Keywords: controlling applications; object-oriented programming; C++; UNIX; data acquisition; graphical user interfaces.

1. Introduction

The programming of experimental control applications involves the solution of several different tasks. It is necessary to create an experimental-control-device interface to provide the connection and interaction of a computer with different modules needed to construct the experiment (step motors, detectors, monitors, temperature or pressure regulators). The display user interface is necessary to provide input and output of the experimental and control information (presentation of digital and graphical information on the screen, control of the keyboard and mouse for input of experimental parameters and control of the experiment operation). It is also very convenient to have some experimental procedure interface to provide simple programming of different experimental conditions.

The UNIX operating system is popular among programmers working with experiment control applications. Creation of the X Windows system, X Toolkit Intrinsics (Xt) and Motif (Xm) extensions has stimulated the writing of new extensions and specializations of such systems (see Kinder *et al.*, 1996; Skinner *et al.*, 1996; Heller, 1992). Such extensions are generally created to simplify user programming with defined hardware.

Very often, creation of a control application for new hardware forces the programmer to undertake system programming. From this point of view the C++ language is a very convenient and useful programming language (see Stroustrup, 1991). The main goal of the work presented in this paper is to create a class library (Interface Object Library) for the writing of experimental control applications in the object-oriented C++ language.

2. C++ as an instrument for object-oriented programming

Control applications always have different control elements (managed objects). The control application is linked with external control devices that may also be managed objects. Any experiment may be considered as several processes running simultaneously. Choosing the process as an object gives control of the process and organizes communication between different processes during the experiment. The existence of specialized yet quite simple objects allows the construction of the more complex and specialized objects necessary for the controlling application. In C++ language the object is described by the declaration and definition of the class. The class is a structure of data, functions (called method functions) and operations that may be executed with an object of the class. After creation, an object is initialized: its parameters and, if necessary, communications with other objects are defined. Finally, the method functions of the object are used to control it.

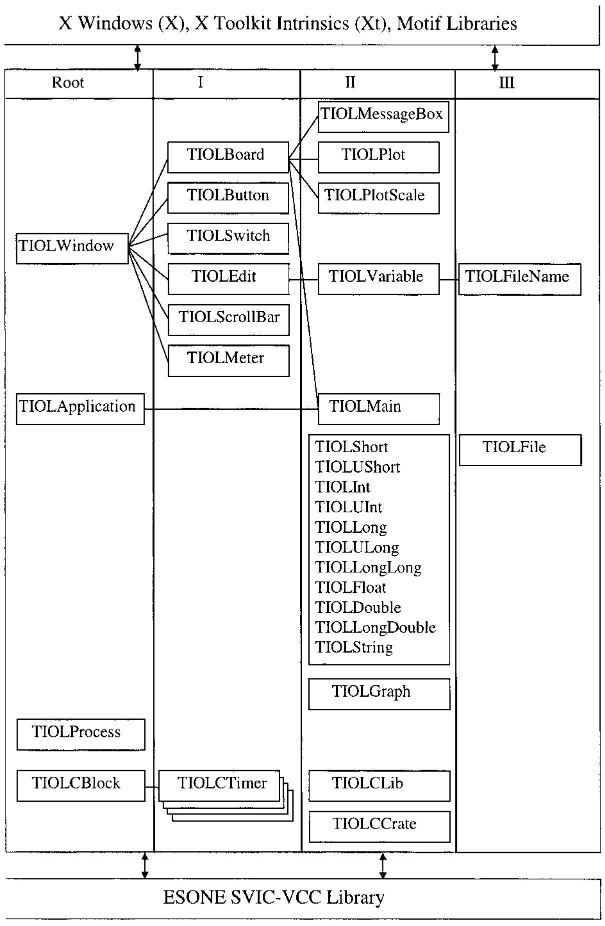


Figure 1
General overview of the Interface Object Library. There are four levels of hierarchy. TIOLMain is a derived class derived from both TIOLBoard and TIOLApplication. The classes TIOLShort, TIOLInt, ..., TIOLString, TIOLFile, TIOLGraph, TIOLCCrate and TIOLCBlock use classes from correspondent levels.

Synchrotron radiation experiments have particular characteristics that have to be considered when creating the controlling application. As a rule, synchrotron radiation beamline experimental stations consist of two types of devices and equipment: beamline optics and experimental X-ray equipment. Taking account of the large number of devices (mirrors, monochromators, shutters, slits, goniometers, detectors, temperature and pressure controllers, scanners), it is better to have one controlling application to ensure that all components of the complex system are under control. This is possible when writing an object library which hides many details of the operations of the devices. Experimental information from diffractometry, spectroscopy or other X-ray methods may need to be registered while the process is running. This means that one has to organize several synchronized processes for collecting, storing and visualizing the experimental data with the possibility of influencing and controlling each process without stopping the experiment. These requirements may be taken into account in C++ programming under the UNIX operating system. The object-oriented C++ language allows one to modify any object from a class library without modifying the library itself.

3. Interface Object Library

Fig. 1 shows a general overview of the Interface Object Library. The library has a hierarchical structure. The root classes TIOLWindow, TIOLApplication, TIOLCBlock and TIOLProcess define the main objects of the controlling application. All classes of the display user interface are constructed with the X Windows system (X), X Toolkit Intrinsics (Xt) and Motif (Xm) Libraries. Classes for CAMAC access are based on the ESONE SVIC-VCC Library.

The object of the TIOLMain class is as an application with a main application window. After creation of this object the main window of the application appears on the screen. One may define the main menu bar (with sub-menus) by using the corresponding method functions of this object.

Different controls are represented by objects from the TIOLEdit, TIOLButton, TIOLSwitch and TIOLScrollBar classes of the library. The completion function is realized in the Interface Object Library. When any control is made active, the defined completion function is called. It is not necessary to create a completion function for every control. As a parameter of this function the library sends the object pointer that allows one to identify the control that is active. The object of the TIOLEdit class may check the type of input character.

The diagnostic indicator is represented by the object of the TIOLMeter class. This is a bar with a different colour bar of a variable size inside. This indicator represents an application variable changing within some limits.

A very important class of the library is TIOLVariable. Objects of this class represent the different variables of the application. The pointer of the application variable is passed to the object of this class during creation. During input operations in the editable field, the value of the variable is automatically updated. If the input value does not belong to the defined range, the colour of the numeric information and the editable border area changes, indicating an incorrect input. This incorrect value will not be updated in the application, the last valid value being kept.

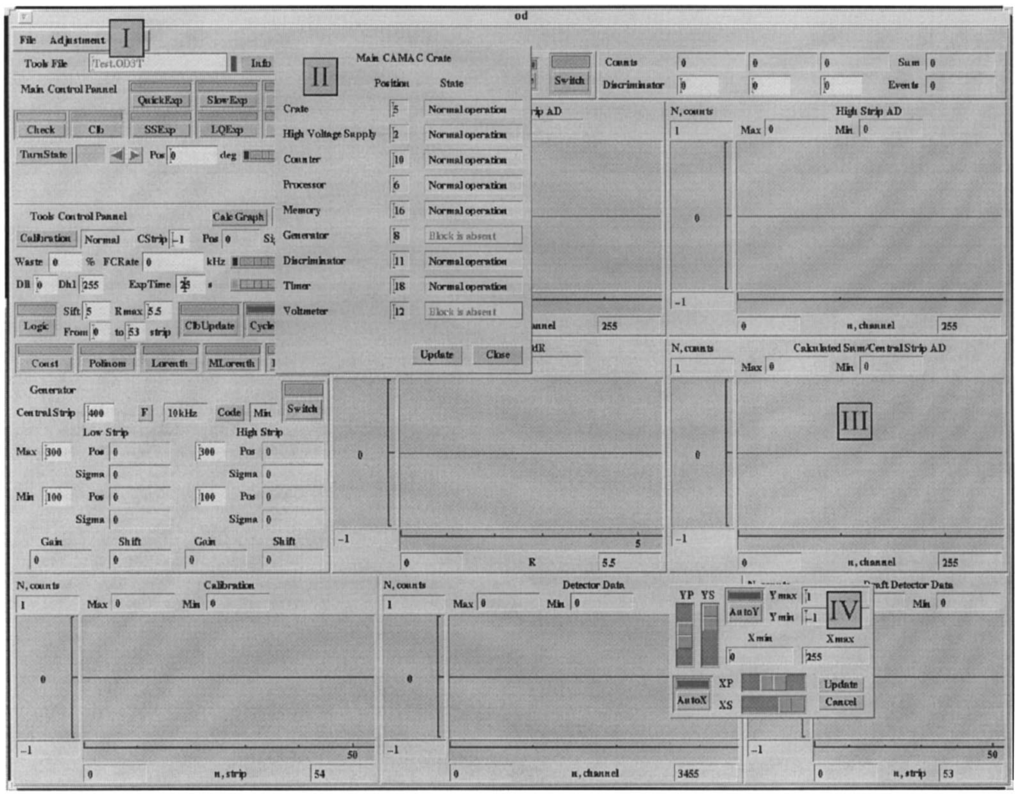
Groups of operational classes are created with the window objects, as with the different kinds of language variables. This group of classes is TIOLInt, . . . , TIOLString. Creation of objects of these classes is similar to creation of those of the language general types. The only difference is that it is additionally necessary to initialize the created object. During initialization one can define the limits of the object changes, *i.e.* the completion function. The completion function will be called after every successful input to the editable field of the object. The objects of these classes may be used in arithmetical equations (Fig. 2).

In the library the specialized class TIOLFile for file operations is created. The file in the Interface Object Library consists of a heading block and several user blocks. The heading block consists of the file identifier, creation date, a comment string and space reserved for future use. The identifier is used by a checking procedure. The user block consists of the number of records, the format of the record, a comment string for the data block and the column of data. All user blocks are defined after initialization by calling the method function of the object AddBlock with the description of the structure block and the pointer to the data that will be used during the Read/Write operations. For reading or writing the file block one has only to point on the necessary block by using the method function Seek(nBlock) and call the Read/Write method function.

For presentation of one-dimensional arrays the TIOLGraph class was created. To optimize the quantity of information output the graphical controls were hidden in an additional dialogue panel that is called by pushing the middle mouse button under the plot area. Several arrays may then be connected to this object with different colours. To update the information of this object the method function Update is used after any modifications of the arrays (Fig. 2).

```
//-----------------------------------------------------------------//
TIOLInt IntVar;                // Creation of the object
   IntVar.Init(...);           // Initialization
int nVar;
nVar = 10;
IntVar = nVar;                 // Output to the screen
nVar = IntVar;                 // Input from the screen
TIOLGraph MainGraph;           // Creation of the object
float fltData[nGraphSize];
   MainGraph.Init(...);        // Initialization
   ...                         // Modification of the fltData[] array
   MainGraph.Update();         // Output of the modified data from
                               // fltData[] array to the screen

//-----------------------------------------------------------------//
TGCP :: TGCP(...):TWLBoard(...) {
AddText("Generator", ...); AddText("Central Strip", ...);
CentralStripCode.Init(...); Frequency.Init(...); StripCode.Init(...);

AddText("Low Strip", ...); AddText("High Strip", ...);
AddText("Max", ...); AddText("Pos", ...); AddText("Sigma", ...);
LowStripCodeMAX.Init(...); LowStripCodePosMAX.Init(...);
LowStripCodeSigmaMAX.Init(...);

AddText("Pos", ...); AddText("Sigma", ...);
HighStripCodeMAX.Init(...); HighStripCodePosMAX.Init(...);
HighStripCodeSigmaMAX.Init(...);

AddText("Min", ...); AddText("Pos", ...); AddText("Sigma", ...);
LowStripCodeMIN.Init(...); LowStripCodePosMIN.Init(...);
HighStripCodeMIN.Init(...);

AddText("Pos", ...); AddText("Sigma", ...);
HighStripCodePosMIN.Init(...); LowStripCodeSigmaMIN.Init(...);
HighStripCodeSigmaMIN.Init(...);

AddText("Gain", ...); AddText("Shift", ...);
AddText("Gain", ...); AddText("Shift", ...);
LowStripCodeGain.Init(...); LowStripCodeShift.Init(...);
HighStripCodeGain.Init(...); HighStripCodeShift.Init(...);

pFrequencyButton = new TWLButton(...);
pStripCodeButton = new TWLButton(...);
pSwitch = new TWLSwitch(...);
}
```

Figure 2
Examples using the objects of the TIOLInt and TIOLGraph classes. A view of the simplified source code of the constructor of the TGCP class of the application (the Generator panel in Fig. 3).



Main CAMAC Crate

	Position	State
Crate	5	Normal operation
High Voltage Supply	2	Normal operation
Counter	10	Normal operation
Processor	6	Normal operation
Memory	16	Normal operation
Generator	8	Block is absent
Discriminator	11	Normal operation
Timer	18	Normal operation
Voltmeter	12	Block is absent

Figure 3
View of the OD3Tools application.

The main class for CAMAC access is TIOLCLib. The object of the TIOLCLib class uses objects of the TIOLCCrate class previously defined with the objects of the TIOLCBlock class, or derived from it.

For simplifying experiment programming the TIOLProcess class was created. Objects of this class are initialized with the pointer pertinent to some procedure that has to be carried out during the experiment. A data buffer may be defined for data exchange between different objects of the TIOLProcess class. Synchronization during the operation with the data buffer is carried out automatically.

Fig. 3 shows the view of the OD3Tools application for control of experiments using a fast one-coordinate position-sensitive X-ray detector (see Aultchenko *et al.*, 1995). From the menu bar (I) the dialogue panel (II) was created for checking the state of CAMAC modules and for changing their descriptors. Another dialogue panel (IV) was created by pushing the middle mouse button under the plot area of the graphical object (III). These dialogue panels are a feature of the Interface Object Library. The simplified source code of the constructor of the TGCP class, for a Generator panel, is shown in Fig. 2.

4. Conclusions

The base Interface Object Library with graphical user, CAMAC and process interfaces was created in the Solaris 2.4 operating system. There is a smaller IBM PC realization of this library for Windows 95 (written in Borland C++ 4.0). There is the possibility of adding another device interface using a corresponding device access library. In the example of the OD3Tools application for controlling the fast detector, good real-time response of the application during actual experiments was achieved.

The authors would like to thank Professor N. Yagi for the valuable and constructive discussions about the contents of the Interface Object Library. Special thanks are expressed to Dr R. Thissen for very useful remarks about the window interface of the Interface Object Library. This work was made possible with the financial support of the Ministry of Education, Science, Sports and Culture (MONBUSHO) of the Government of Japan.

References

Aultchenko, V. M., Baru, S. E., Dubrovin, M. S., Titov, V. M., Velikzhanin, Ju. S. & Usov, Ju. V. (1995). *Nucl. Instrum. Methods Phys. Res. A*, **367**, 79–82.

Heller, D. (1992). *Motif Programming Manual*. USA Sebastopol, CA: O'Relly and Associates.

Kinder, S. H., McSweeney, S. M. & Duke, E. M. H. (1996). *J. Synchrotron Rad.* **3**, 296–300.

Skinner, J. M., LaBarca, R. S. & Sweet, R. M. (1996). *Nucl. Instrum. Methods Phys. Res. A*, **383**, 627–630.

Stroustrup, B. (1991). *The C++ Programming Language*. Reading, MA: Addison Wesley.