# Lawrence Berkeley National Laboratory

## Lawrence Berkeley National Laboratory

### Title

Experience with ActiveX control for simple channel access

### Permalink

https://escholarship.org/uc/item/8j69g8gh

### Authors

Timossi, C.
Nishimura, H.
McDonald, J.

### Publication Date

2003-05-15

eScholarship.org Powered by the California Digital Library
University of California

LBNL-51591
Submitted to PCaPAC 2002

# EXPERIENCE WITH ACTIVEX CONTROL FOR SIMPLE CHANNEL ACCESS

C. Timossi, H. Nishimura, J. McDonald, Lawrence Berkeley National Laboratory, USA

*Abstract*[*]

Accelerator control system applications at Berkeley Lab's Advanced Light Source (ALS) are typically deployed on operator consoles running Microsoft Windows 2000 and utilizing EPICS[1] channel access for data access. In an effort to accommodate the wide variety of Windows based development tools and developers with little experience in the nuances of EPICS client programming, ActiveX controls have been deployed on the operator stations to both aid the development effort and to standardize the programming interface to the control system. Use of ActiveX controls in the accelerator control environment has been presented in a previous report[2]. Here we report on some of our experiences with the use and development of these controls.

## 1 CONTROL SYSTEM ENVIRONMENT

The ALS control room layout includes twelve MS Windows based operator stations running dozens of applications developed over the ten-year life of the accelerator. These applications are in process of being updated to use channel access, the network based protocol used by EPICS. The developers range widely in their experience with programming and programming tools. The developers range from operators to physicists to control system personnel. Currently, National Instrument's Labview, Microsoft Visual C and Visual Basic, and Borland's Delphi and C++ Builder are the most popular development platforms but there is no clear mandate.

Our previous mechanism for supporting data access from these different tools was to build a C language based Dynamic Link Library (DLL) that is also a standard method of access for the Windows OS. This method, however, required us to understand each development tool's requirements for accessing C language function calls and maintaining this extra layer of code.

Several ActiveX controls were developed. One of these controls is used to implement a programming interface (API) that is a subset of the EPICS client API called Simple Channel Access[†]. Other controls are higher level and present a simplified interface for direct control of device hiding the EPICS model completely.

## 2 SIMPLE CHANNEL ACCESS (SCA)

The EPICS channel access API was designed to implement a high performance network protocol including such features as data and connection call-backs, event notifications and smart aggregation of data requests. For the more casual programmer, accustomed to simple synchronous subroutine calls to get data, the CA API can be difficult to fathom. We've attempted to provide a scaled-down interface for the more casual programmer by providing a subset of features in a library called Simple Channel Access. This interface attempts to preserve the 'grouping' of data calls when appropriate while trying to present the programmer with a simple set of synchronous function calls.

Experience at the ALS shows that applications tend to fall into two categories. First there are applications (typically GUIs) that attempt to display large amounts (hundreds) of data items on a screen as status information. For these applications there is little concern for update rates of more than a few times per second. The order of arrival of the values is not important. For these applications it's most efficient to group as many requests for data as possible into one network request and then to poll every so often for new data. The second type of application performs active control at, perhaps, a much higher rate (10's of milliseconds). An example might be an application that pulses a corrector magnet and observes the response of a thermocouple. For this type of application it's important that the new value for the magnet is sent before the thermocouple value is read; grouping the two data requests into one network request is not appropriate. Unfortunately, we found it difficult to handle these types of applications transparently so it's left to the programmer to decide whether or not to group requests.

## 3 HIGHER LEVEL OBJECTS

While simple, the SCA interface is very low level. Access to data depends on knowing the unique *process variable name* for the item of interest. For a complex accelerator device at the ALS, such as an insertion device (either a wiggler or undulator), control may involve looking up dozens of process variable names and understanding how each relates to the control of the device. For such devices it sometimes makes sense to hide channel access completely and to implement an ActiveX control that exposes methods for the most common operations of the device. This encapsulation of the device's behaviour makes it much easier for the application developer, who may be expert in the operation of the device but cares little for the communication details, to concentrate on the design of an operator friendly control application. More importantly, this encapsulation means that subsequent application developers don't have to re-learn the relationship between

[*] This work was supported by the Director, Office of Science, Office of Basic Energy Sciences, U.S. Department of Energy under Contract No. DE-AC03-76SF00098

[†] Thanks to Loren Shalz, LBNL who developed SCA.

a bewildering number of process variable names and the behaviour of the device.

Developing these higher-level controls is time consuming. There is some extra programming required but the main effort involved is researching the device in question so as to be able to design a sensible interface. This extra effort is not always warranted; particularly in cases where the control requirements are not clearly understood requiring many iterations of the interface.

The insertion devices at the ALS are examples of devices where encapsulation seemed to make sense. These devices have many control variables and several modes of operation. As new insertion devices came on line, extending the existing control application was a constant annoyance; every addition required re-learning the logic of a monolithic application. An ActiveX control was developed and tested in concert with the new application development. This approach allowed for a division of labour between the GUI application programming and the control logic and also made the control of the device more accessible to future applications.

## 4 ACTIVEX RATIONAL

For Microsoft Windows based systems, the ActiveX control has a number of advantages. There are well-developed tools to build them; at the ALS we use MS Visual C++. They are *registered* with the OS. This registration tells a client application, in a standardized manner, where they are and what features they have and which makes *late binding* possible. A good example of late binding is seen in the Labview development environment. Once an ActiveX object is selected from a list of registered controls, the developer has only to right-click on a method to see a list of all the supported methods. Once a method is chosen, the types of all the arguments are also displayed. We are also able to use the MS Visual Installer to create installer programs that take care of the details of both registration and copying the files into the appropriate place in the file system. As important is the ability to uninstall a control using the standard Windows facility for adding and removing programs.

There can be performance issues using ActiveX controls. The ActiveX model was designed to make calls across process and/or machine boundaries. Therefore calling a control's method, which may involve argument marshalling, will take more time than the equivalent library function call. We haven't attempted to measure this delay. As a reference point, the fastest access we've attempted has been for a calibration application deployed as a Labview application running on a PC that sets a DAC and then reads back an ADC from an EPICS-based server (IOC) deployed on a embedded processor. In this case the access rate was purposely limited to 20 milliseconds per request to ensure enough time for the server hardware and software to respond, but this example gives a rough idea of what's possible.

Of course, an ActiveX control is not installable on other operating systems. However, the C++ code used in the implementation of the control is quite portable with the man problem being the need to re-define some of the manifest constants defined for ActiveX error codes.

ActiveX has now been subsumed by Microsoft .NET which defines a new object model. This new model makes ActiveX somewhat obsolete. However, an ActiveX control can still be used directly in the .NET environment (including it's late binding features). We have verified, for instance, that a control can be used in a .NET application developed in the C# language.

## 5 REFERENCES

[1] L.R. Dalesio, et al., "The Experimental Physics and Industrial Control System Architecture", ICALEPCS '93, Berlin, Germany, 1993.

[2] C. Timossi, H. Nishimura, "Accelerator Control Software Construction Based on Software Object Components," PAC'97, Vancouver B.C., Canada, May 1997.

[3] J. McDonald, H. Nishimura, C. Timossi, "Cross Platform Development Using Delphi and Kylix", this conference.