NEW CAPABILITIES OF THE EPICS IOC SHELL*

W. Eric Norum#, Argonne National Laboratory, Argonne, IL 60439, USA

Abstract

The Experimental Physics and Industrial Control System (EPICS) is used at research facilities around the world as the basis for controlling equipment such as particle accelerators, beamlines, and telescopes. Using EPICS 3.14, it is now possible to run Input/Output Controllers (IOCs) on a wide range of hardware and operating system platforms. User interaction with these operating-system-independent EPICS applications takes place through the IOC shell. This presentation highlights some of the recent changes made to the IOC shell that provide a more powerful and flexible user interface.

INTRODUCTION

EPICS [1] is a collection of software tools developed collaboratively that can be integrated to provide a comprehensive and capable control system. Most servers are implemented with EPICS iocCore, a set of real-time software tools designed to run on Input/Output Controllers (IOCs). IOCs are networked computers that provide the direct interface to the technical equipment.

Up to and including EPICS release R3.13, the term 'IOC' implied a VME processor running the vxWorks real-time operating system. With the release of EPICS R3.14 these limitations have been removed [2-4]. Now an IOC can also be a Linux PC, a Windows PC, an OS X Macintosh, a Solaris Sun, an HPUX Hewlett-Packard, or a variety of embedded processors using RTEMS [5].

This greatly increases the range of EPICS applications and makes EPICS an attractive solution to the controls needs of even small laboratories and individual beamline developers.

EPICS applications running on vxWorks use the vxWorks shell to read the application startup script (*st.cmd*) and to provide for interactive entry of diagnostic and debugging commands. The IOC shell provides similar functionality on non-vxWorks platforms. It was first released as a simple command interpreter capable only of reading simple *st.cmd* startup scripts and providing limited interactive operation.

COMMAND EDITING/HISTORY

Shortly after the initial release of the IOC shell the need for more capable interactive operation became clear. The ability to edit command lines during entry and to review, modify, and reenter previous commands were the most commonly requested additions. The IOC shell has been modified to optionally use one of two mechanisms to provide these capabilities. The two mechanisms available are the GNU readline library [6] and the tecla command-line library [7].

The GNU readline library provides a set of functions for use by applications that allows users to edit command lines as they are typed. Both Emacs and vi editing modes are available. The readline library includes additional functions to maintain a list of previously-entered command lines, to recall and perhaps reedit those lines, and perform csh-like history expansion on previous commands.

The tecla library provides UNIX and LINUX programs with interactive command line editing facilities similar to those of the UNIX tcsh shell. In addition to simple command-line editing, it supports recall of previously entered command lines.

For various reasons neither of these packages is distributed with EPICS nor enabled by the distributed configuration build rules. The problems associated with the libraries are:

- Readline is distributed under the terms of the GNU Public License.
- Libtecla is not bound by such restrictive licencing terms but has a flaw in that diagnostic output produced by other threads when the IOC shell is waiting for input will appear garbled. In particular, newlines are not converted to carriage-return/newline pairs.
- Each package is quite large. For example, on the Intel x86 architecture EPICS IOC applications built with readline support are over 150 kbytes larger than those built with no command-line editing or history.

However, if these difficulties are not insurmountable for a particular institution or application, either package provides a convenient and powerful mechanism with which to interact with EPICS IOC applications.

The choice of command-line input support is made by adding

```
COMMANDLINE_LIBRARY=READLINE
```

or

```
COMMANDLINE_LIBRARY=LIBTECLA
```

to an EPICS configuration file and then rebuilding the iocCore libraries. This overrides the default value of

```
COMMANDLINE_LIBRARY=EPICS
```

that provides the limited interactive operation of the original IOC shell distribution.

The prompt string for interactive operation is taken from the IOCSH_PS1 environment variable. If a

*The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract No. W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government
#norume@aps.anl.gov

command-line editing library is used, the amount of command history is set by the IOCSH_HISTSIZE environment variable. The default values for these parameters are 'epics> ' and 50 lines, respectively.

The IOC shell distributed with the most recent release of EPICS (R3.14.4) uses the vxWorks ledLib library to provide command-line editing and command history on vxWorks platforms as well. With this addition some form of command-line editing and history is now available for EPICS IOC shell operation on all platforms.

MACRO EXPANSION

To assist the development of "shrink-wrap" EPICS applications that can be distributed and used without requiring editing of the distributed startup scripts or recompiling of the application code, the IOC shell has been modified to expand environment variables on input command lines. Once a line of input has been input, character sequences of the form $(*var*) or ${*var*} are replaced with the value of the *var* environment variable. There is no expansion inside single quotes or following a backslash.

The application build process creates an *envPaths* file with declarations formed from the configure/RELEASE parameters. The IOC shell epicsEnvSet command is used to set the environment variable values.

```
epicsEnvSet(IOC,"iocmyExample")
epicsEnvSet(TOP,"/home/enm/myExample")
epicsEnvSet(EPICS_BASE,"/home/enm/EPICS/base")
.
.
```

This file is read by the startup script and thus makes the values of all the release parameters available for use by other startup script commands.

Indirect macro definitions are also allowed. The following example passes 'arg1' to the first dbpr command and 'arg2' to the second even though both are invoked with the same argument:

```
epicsEnvSet var1 '${var2}'
epicsEnvSet var2 arg1
dbpr ${var1}
epicsEnvSet var2 arg2
dbpr ${var1}
```

Note the use of single quotes to inhibit macro expansion in the first environment variable assignment. This defers the expansion until the var1 macro is next processed.

As shown in Table 1, the addition of macro expansion capability to the IOC shell has a very small effect upon the memory footprint of an EPICS application. The very small increase in memory usage is a result of the IOC shell macro expansion performed using the macLib macro expansion routines already present in iocCore.

Environment variable macro expansion is built into the IOC shell regardless of the command-line input mechanism in use.

Table 1: Application Size Increase Resulting from Addition of Macro Expansion

Architecture	**Increase (bytes)**
Linux	690
Solaris	2055
Darwin	1113

DIAGNOSTIC VARABLES

It is no longer necessary to declare and register a full IOC shell command to gain access to diagnostic variables. To make a variable available now requires only the following steps:

- Include the epicsExport header file in the C file where the variable will be used:

  `#include <epicsExport.h>`

- Declare the variable in the C source file where it will be used:

  `static double myParameter;`

- Export the variable from the C source file:

  `epicsExportAddress(double, myParameter);`

- Declare the variable in an application database definition file:

  `variable(myParameter, double)`

Once the above steps have been taken, the variable can be accessed from the command line using the IOC shell

```
var [name [value]]
```

command. If both arguments are present, the specified value is assigned to the named variable. If only the name argument is present, the current value of that variable is printed. If neither argument is present, the names and values of all variables registered with the shell are printed.

Integer variables are also supported. The only change is to replace the 'double' keywords with 'int'. For example, to make an integer debugging level variable available, the C source file would contain:

```
.
.
#include <epicsExport.h>
.
.
static int gpibDebugLevel;
epicsExportAddress(int, gpibDebugLevel);
```

and the application database definition file would contain:

```
variable(gpibDebugLevel, int)
```

which could be shortened to

```
variable(gpibDebugLevel)
```

since a missing type is taken to be ‘int’.

CONCLUSION

The additions described in this paper have greatly increased the utility and power of the IOC shell. Although the IOC shell was conceived to provide command-line input capability to non-vxWorks IOC applications, the power and flexibility now offered by the IOCS shell have led to its use even on vxWorks platforms. The command-line macro expansion capability in particular has proven to be a great asset in simplifying the deployment of EPICS IOC applications on vxWorks and non-vxWorks platforms.

REFERENCES

[1] http://epics.aps.anl.gov/

[2] M. Kraimer, “EPICS: Porting iocCore to Multiple Operating Systems,” ICALEPCS’99, Trieste, October 1999, 33-35, 2000.

[3] M.R. Kraimer, J.B. Anderson, J.O. Hill, W.E. Norum, “EPICS: A Retrospective on Porting iocCore to Multiple Operating Systems,” ICALEPCS’01, San Jose, California, November 2001, 238-240, 2002.

[4] R. Lange, J.B. Anderson, A.N. Johnson M.R. Kraimer, W.E. Norum, L.R. Dalesio, J.O. Hill, “EPICS: Recent Developments and Future Perspectives,” these proceedings.

[5] W.E. Norum, “EPICS on the RTEMS Real-Time Executive,” Proceedings of SRI2001, Madison, Wisconsin, August 2001, American Institute of Physics, Review of Scientific Instrumentation, January 2002.

[6] http://www.gnu.org/directory/readline.html

[7] http://www.gnu.org/directory/libs/libtecla.html