

# EzcaFort User's Guide

Xingyue Li  
December 7, 1995

## 1. Introduction

EzcaFort is an Interface between Fortran and the EZCA library which provides Fortran calls to EPICS Channel Access.

Although there is nearly a one-for-one match between the routines in ezcaFort and those in the EZCA library, the syntax of the ezcaFort routines is usually not the same as that of the EZCA routines. This is because the EZCA is for C language instead of Fortran. Of course, if necessary one can consult EZCA Primer

(<http://www.aps.anl.gov/asd/controls/epics/manuals/EzCaPrimer/EzcaPrimer.html>, or <http://www.aps.anl.gov/asd/controls/epics/manuals/EzCaPrimer/EzcaPrimer.ps>).

This version of ezcaFort is just for Sun4 workstation using UNIX operating system.

## 2. Getting Started

### 2.1 Data Types

EzcaFort allows the user to read/write information in 6 different data types. Below are the allowable ezcaFort data types.

<b>Data Type</b>	<b>Corresponding Fortran Format</b>
BYTE	CHARACTER*1 (the data should be between -128 and 127)
STRING	CHARACTER*L (L should be long enough)
SHORT	INTEGER*2
LONG	INTEGER*4
FLOAT	REAL
DOUBL	DOUBLE PRECISION or REAL*8

For example, if one wants to put 1, 2, 3, ..., 20 into a waveform named 'XLI:waveform' he can use the following sentences.

```
CHARACTER BT(20)
INTEGER RET_CODE
DO 10 I = 1, 20
10  BT(I) = CHAR(I)
CALL P_VALUE('XLI:waveform', 'BYTE', BT, 20, RET_CODE)
IF(RET_CODE.NE.0)STOP ' PUTTING UNSUCCESSFULLY !!!'
END
```

Here 'XLI:waveform' is the process variable which should be connected. 'BYTE' tells ezcaFort that BYTE or CHARACTER\*1 data type is being used. Since BT is of type character, the intrinsic function CHAR( ) should be used.

## 2.2 Makefile

Below is the Makefile used to make MYTASK.f.

```
EPICS = .
T_A=sun4
include $(EPICS)/config/CONFIG_BASE

BASE = /usr/local/epics/R3.12.1.4/base
EXTENSIONS = /usr/local/epics/extensions
USR_CFLAGS = -I$(EXTENSIONS)/include -g
USR_LDFLAGS = -L$(EXTENSIONS)/lib/sun4 -L$(BASE)/lib/sun4
USR_LDLIBS = -lezca -lca -lDb -lCom

OBJS = MYTASK.o ezcaFort.o

usr1: $(OBJS)
    f77 $(OBJS) $(USR_LDFLAGS) $(USR_LDLIBS)

include $(EPICS)/config/RULES.Unix
```

## 2.3 Arguments of EzcaFort

Arguments	Meaning
S_NAME	Process variable name, sometimes can be field name such as 'XLI:WAVEFORM1.SCAN'.
N_ELEM	Number of elements in DATA_BUFF (If DATA_BUFF is not an array it should be 1).
DATA_BUFF	Data buffer to store N_ELEM data of a given data type, can be an array, a string or a single variable.
S_TYPE	Type of DATA_BUFF, can be 'string', 'byte', 'short', 'long', 'float' and 'double'.
I_STATU	Return code. 0 represents successful call. Anything else indicates a problem.
D_LOW	Low limit value.
D_HIGH	High limit value.
I2_PRECISION	Precision of the value of the process variable.
I_NANOSEC	Nanoseconds within a second.
N_SECPAST	Seconds since 0000 Jan 1, 1990.
I2_STATE	Status code of G_STATUS( ).
I2_SEVERITY	Severity code of G_STATUS( ).
S_UNITS	Unit of the process variable. Its length should be at least 8 bytes.
S_PREFIX	A user-supplied character string(possibly zero length, i.e. "") that EZCA will use as prefix when displaying error message.
S_BUFF	A user-supplied character string in which error message will be stored if an error occurs.
I_RETCODE	An array in which return codes of group elements will be stored.
I_SIZE	Number of I_RETCODE. I_SIZE should be at least equal to the number of the group elements which the user is interested in.
F_SECONDS	Time in seconds.
I_RETRYCOUNT	Retrycount, one of the two tunable parameters.

Note

1. "S\_" means a character string. Both uppercase and lowercase can be used. The length of the string should be long enough.
2. "I\_", "I2\_" represent 4 bytes and 2 bytes integer numbers respectively. "N\_" is the same as "I\_" and means "number of".
3. "D\_" and "F\_" represent real\*8 and real float point numbers respectively.

### 3. Main Subroutines

All routines in ezcaFort library are subroutines, so they should be called on using "call" syntax.

#### Subroutines

G\_VALUE(S\_NAME, S\_TYPE, DATA\_BUFF, N\_ELEM, I\_STATUS)  
 P\_VALUE(S\_NAME, S\_TYPE, DATA\_BUFF, N\_ELEM, I\_STATUS)  
 G\_CTRL\_LIMITS(S\_NAME, D\_LOW, D\_HIGH, I\_STATUS)  
 G\_GRAPH\_LIMITS(S\_NAME, D\_LOW, D\_HIGH, I\_STATUS)  
 G\_N\_ELEM(S\_NAME, N\_ELEM, I\_STATUS)  
 G\_PRECISION(S\_NAME, I2\_PRECISION, I\_STATUS)  
 G\_STATUS(S\_NAME, I\_NANOSEC, N\_SECPAST, I2\_STATE, I2\_SEVERITY, I\_STATUS)  
 G\_UNITS(S\_NAME, S\_UNITS, I\_STATUS)

G\_VALUE( ) gets value of the process variable or the field. The result is written into DATA\_BUFF.  
 P\_VALUE( ) writes the value in DATA\_BUFF into the process variable or the field. Below is an example.

```

character*40 nm
integer*2 s1, s2           ! SHORT is integer*2 in SUN Workstation
nm = 'XLI:testAI.DISV'    ! 'XLI:testAI' should be connected.
call g_value(nm, 'short', s1, 1, ireturn)
if(ireturn.eq.0)then
  write(*,200)s1
  write(*,205)
  read(*,*)s2
  call p_value(nm, 'Short', s2, 1, ireturn)
  call g_value(nm, 'Short', s1, 1, ireturn)
  write(*,210)s1
else
  stop ' Something is wrong with g_value( )!'
endif
nm = 'XLI:testAI'
call g_value(nm, 'FLOAT', f1, 1, ireturn)
write(*,220)f1
write(*,225)
read(*,*)f2
call p_value(nm, 'float', f2, 1, ireturn)
call g_value(nm, 'float', f1, 1, ireturn)
write(*,230)f1
stop ' Okey'
200 format(/10x, 'XLI:testAI.DISV =', i2)
205 format(/10x, 'Input new value for field DISV: ', $)
210 format(/10x, 'New XLI:testAI.DISV =', i2)
220 format(/10x, 'XLI:testAI = ', f7.2)
225 format(/10x, 'Input new value for XLI:testAI: ', $)

```

```
230 format(/10x, 'New XLI:testAI =', f7.2)
    end
```

In this document, G\_ and P\_ always mean "get" and "put", respectively. So G\_CTRL\_LIMITS( ) gets control limits, G\_GRAPH\_LIMITS( ) graphic limits. If someone wants to know how many elements there are in a given process variable, he can use G\_N\_ELEM( ). Any one can guess the purposes of G\_PRECISION( ), G\_STATUS( ) and G\_UNITS( ).

## 4. Error Handling

### Subroutines

```
E_MESSAGE_ON( )
E_MESSAGE_OFF( )
ERR_PREFIX(S_PREFIX)
G_ERR_STRING(S_PREFIX, S_BUFF, I_STATUS)
```

By default, if an error occurs EZCA will display error messages. The user can toggle this automatic error reporting feature with E\_MESSAGE\_ON( ) or E\_MESSAGE\_OFF( ). The default state is ON. When an error occurs, ERR\_PREFIX( ) uses S\_PREFIX as a prefix to the error message. Meanwhile G\_ERR\_STRING( ) puts the optional prefix and the error message into a user-supplied string S\_BUFF. In both ERR\_PREFIX( ) and G\_ERR\_STRING( ), S\_PREFIX can be zero length, i.e. "".

Because ezcaFort is just an interface between Fortran and EZCA, when something is wrong the displayed error message is issued by EZCA instead of ezcaFort. In these circumstances, the user should check his/her ezcaFort call. EzcaFort just gives one kind of warning such as "Argument 2 of g\_value( ) is not available!!!" if such an error occurs.

## 5. Groups

### Subroutines

```
GROUP_ON(I_STATUS)
GROUP_OFF(I_STATUS)
GROUP_REPORT(I_RETCODE, I_SIZE, I_STATUS)
```

Sometimes it is more efficient to do a large block of unconditional reads and/or writes in a group. In these circumstances, ezcaFort merely checks the validity of arguments of each ezcaFort main subroutine(described in Section 3) call. The actual work is postponed until the end of the group is encountered. If something in the group is wrong GROUP\_OFF( ) returns the first encountered unsuccessful return code. Since GROUP\_REPORT( ) returns I\_SIZE return codes, users interested in the return status of all the ezcaFort calls should use it.

Not all programs should use ezcaFort groups. The following is an example which is a poor candidate for groups.

```
integer*2 svalue(2)
data svalue/10, -21/
call E_MESSAGE_OFF( )
call GROUP_ON(I_STATUS)
```

```

call G_VALUE('myai', 'float', fvalue, 1, i_status)
if(fvalue.lt.0.0)call p_value('mywaveform', 'short',
# svalue, 2, i_status)
call GROUP_OFF(I_STATUS)
if(i_status.ne.0)call ERR_PREFIX("")
end

```

In this program, CALL P\_VALUE( ) will never be executed since the value in variable FVALUE is garbage.

## 6. Monitors

### Subroutines

```

MONITOR_ON(S_NAME, S_TYPE, I_STATUS)
MONITOR_OFF(S_NAME, S_TYPE, I_STATUS)
MONITOR_CHECK(S_NAME, S_TYPE, I_STATUS)
DELAY(F_SECONDS, I_STATUS)

```

If the user has a process variable whose value will not change very often but will be read frequently, then the user should establish a monitor on that process variable. By using MONITOR\_ON( ) and MONITOR\_OFF, the user can place and remove monitors at any time.

Calling MONITOR\_ON( ) immediately establishes a CA monitor of the specified request type on the named process variable. Any time the value of the process variable changes (presumably infrequently) the new value is cached automatically and silently. All subsequent reads of that process variable under that request type will not generate a CA read, but rather, will simply read the cached value. Obviously, this is more efficient.

When MONITOR\_CHECK( ) is called on, it returns a non-zero value if there is a new (unread) value in the monitor, otherwise it returns 0. This function is particularly useful when the read operation is expensive in time, e.g., reading large arrays.

DELAY( ) should be called whenever monitors are used and there is a substantial amount of time between any two adjacent ezcaFort calls since under these circumstances it is possible to lose changes in variables. Between all such pair of calls, the user should call DELAY( ). Usually, the value of argument F\_SECONDS should be around 0.01 seconds.

## 7. Tunnig EZCA

### Subroutines

```

G_TIMEOUT(F_SECONDS)
G_RETRYCOUNT(I_RETRYCOUNT)
P_TIMEOUT(F_SECONDS)
P_RETRYCOUNT(I_RETRYCOUNT, I_STATUS)

```

In these four subroutines, arguments F\_SECONDS and I\_RETRYCOUNT are used to determine when to stop waiting for connections and confirmations of reads and writes. EZCA uses them by waiting F\_SECONDS seconds and then, if necessary, waiting F\_SECONDS seconds a maximum of I\_RETRYCOUNT more times, resulting in a maximum total timeout time of F\_SECONDS\*(1 +

I\_RETRYCOUNT).

Empirically, under normal circumstances EZCA can reliably process(read or write) 200 process variables per second. The default value for F\_SECONDS and I\_RETRYCOUNT are 0.05 seconds and 599 times, respectively. So the default maximum total timeout is 30 seconds. If necessary, users should adjust F\_SECONDS and I\_RETRYCOUNT accordingly.