EZConsole
Programmer's Reference
Version 0.2
13 March 2004

Procedure Specification Summary

Procedure names are provided in two forms; a long name suitable for use with Fortran 90 and later compilers and a short 6-character name suitable for calling from FORTRAN 77 compilers. Most FORTRAN 77 compilers are able to make use of the long procedure names. The use of long names in FORTRAN 77 code is however non-standard. Argument passing conventions are identical for both procedure names.

Procedure Name	Description
EZConsOpen / EZOPEN	Open the console window and initialize EZConsole
EZConsClose / EZCLSE	Close the console window
EZConsSetTitle / EXSTTL	Set console window title
EZConsGetTitle / EZGTTL	Get console window title
EZConsGetStdHandle / EZGSH	Get console window standard in, out, and error handle ids (only needed if desired to interface with Win32 API)
EZConsWriteText / EZWTXT	Write text at current position
EZConsReadText / EZRTXT	Read text at current position
ezConsSetPrompt / EZSP	Set prompt string
ezConsSetMode / EZSM	Set console mode
ezConsGetMode / EZGM	Get Console mode
ezConsSetCurState / EZSCS	Set cursor size and visibility state
ezConsGetCurState / EZGCS	Get cursor size and visibility state
EZConsSetCurPos / EZSCP	Set cursor/current position
EZConsSetTextAtt / EZSTA	Set text attribute at current cursor position
EZConsCtrlEvent / EZEVNT	Generate control-c/control-break event

Return Code Definitions

Return Code	Description
000	No error
001	Undefined error
002	Application not initialized, ezConsOpen (EZOPEN) is required
003	Internal application error
004	Operating system API error
005	Invalid handle type
006	Trace file not assigned, must be assigned externally
007	Procedure identifier value out of range
008	Error code identifier value out of range
009	Trace type argument out of range

010	Trace sequential file not pre-assigned
011	Standard handle query type out of range
012	Message box type parameter out of range
013	Unable to assign standard input handle
014	Unable to assign standard output handle
015	Unable to assign standard error handle
016	Unable to retrieve console title
017	Unable to retrieve console window handle
018	Unable to retrieve module instance handle
019	Unable to register EZConsole window class
020	Unable to initiate thread
021	Unable to initiate trace window message loop thread
022	Thread priority index out of range
023	Unable to set thread priority
024	Unable to retrieve thread priority
025	Unable to suspend thread
026	Unable to resume thread
027	Console input mode option out of range
028	Console output mode option out of range
029	Unable to set console input mode
030	Unable to set console output mode
031	Unable to retrieve console input mode state
032	Unable to retrieve console output mode state
033	Unable to retrieve console cursor state
034	Cursor size value out of range
035	Cursor visibility value out of range
036	Unable to set console cursor state
037	Text foreground attribute out of range
038	Text background attribute out of range
039	Control event type out of range

Configuration Identification

The EZConsole library was built using Compaq Visual Fortran version 6.6B using default DLL project settings. It is supplied as a Windows DLL and includes the associated import library file and this interface definition document in PDF format. An associated source include file of common constants will be included in a future release.

- EZCONSOLETxx.DLL – The dynamic link library containing the EZCONSOLE procedures
  - EZCONSOLET01 – CVF 6.6C

  - EZCONSOLET02 – Fortran Powerstation 4.0
- EZCONSOLETxx.LIB – The associated import library for the LINK editor
- EZCONSOLE.PDF – This programmer’s reference document

Cautionary Statement

EZConsole is in an early stage of development. The possibility of redesign exists for any of the documented APIs, including specific procedure names, option values, default settings, and return codes. Use of this API to develop applications is discouraged. Testing of the interfaces themselves and recommendations for correction or improvement are welcome. Design will be frozen at the 1.0 version when released. This does not preclude further design changes when new full revisions are made. Minor corrections may be issued to the existing design and will be documented by changing the minor version number component (e.g. 1.1, 1.2, 1.3). New versions (e.g. Version 2.0) will likely contain the same pre-existing procedures plus new functionality. This however, cannot be guaranteed if through usage of a prior version (e.g. Version 1.0) a design flaw is discovered that is serious enough to warrant a design change.

Distribution

The EZConsole library is provided as freeware. It may be distributed as a development tool for either personal or commercial use. Copyright statements within the libraries, includes, and documentation must remain and attribution must be provided to Fortran Library, including reference to the web site (**http://www.fortranlib.com**) within header comments of any including, derivative source code and associated documentation.

Future Development

The library presently contains numerous additional routines for setting security attributes, initiating and controlling threads, and message boxes. A trace facility is also partially implemented. Since the stated goal was to limit to console facilities, it is undermined whether any of these additional features will be made public. However, many of them were necessary in order to develop the internal trace facility.

Detailed Procedure Specifications

Name: EZConsOpen(RetCode)

Alternate Name: EZOpen(RetCode)

Procedure Type: Subroutine

Arguments:

- `INTEGER, INTENT(OUT) :: RETCODE !integer return code`
    - `0, 2, 4, 13, 14, 15, 16, 17 (see return code definition table)`

Description: Initializes internal structures and opens a standard console. The console may be mixed with Fortran READ/WRITE statements, however, a call to EZCONSOPEN must occur prior to the first Fortran READ/WRITE statement. An application abort will occur otherwise.

Example:

```
CALL EZCONSOPEN(RETCODE)      !Open console
CALL EZSETCURPOS(1,1,RETCODE) !Set current position to 1,1
CALL EZCONSWRITETEXT(‘Text to write’,RETCODE) !Write text at 1,1
CALL EZCONSCLOSE(RETCODE)     !Close console
```

Index

Name: EZConsClose(RetCode)

Alternate Name: EZCLSE(RetCode)

Procedure Type: Subroutine

Arguments:

- `INTEGER, INTENT(OUT)     :: RETCODE !integer return code`
  - `0, 2, 4(see return code definition table)`

Description: Closes the console. Subsequent attempts to write to the console will result in an application run time abort.

Example:

```
CALL EZCONSCLOSE(RetCode) !Close console
```

Index

Name: EZConsSetTitle(text,retcode)

Alternate Name: EZSTTL()

Procedure Type: Subroutine

Arguments:

- `CHARACTER(*), INTENT(IN) :: TEXT !character string to print`
- `INTEGER, INTENT(OUT)     :: RETCODE !integer return code`
    - `0, 2, 4 (see return code definition table)`

Description: Sets the console window title.

Example:

```
CALL EZCONSSETTITLE(‘text to write’,RETCODE) !Set console title
```

Index

Name: EZConsGetTitle(text,retcode)

Alternate Name: EZGTTL()

Procedure Type: Subroutine

Arguments:

```
? CHARACTER(*), INTENT(OUT) :: TEXT !character string to print
? INTEGER, INTENT(OUT)      :: RETCODE !integer return code
    o 0, 2, 4 (see return code definition table)
```

Description: Gets the console window title.

Example:

```
CALL EZCONSGETTITLE(TEXT,RETCODE)  !Get console title value
```

Index

Name: EZConsGetStdHandle(Type,Handle,RetCode)

Alternate Name: EZGSH(Type,Handle,RetCode)

Procedure Type: Subroutine

Arguments:

```
? CHARACTER(*), INTENT(IN) :: TYPE !Input, output, or error
    o 1 = standard input handle
    o 2 = standard output handle
    o 3 = standard error handle
? INTEGER, INTENT(OUT)     :: Handle  !integer handle id
? INTEGER, INTENT(OUT)     :: RETCODE !integer return code
    o 0, 2, 4, 11 (see return code definition table)
```

Description: Gets the standard handle value for use with non-EZCONSOLE Win32 API procedure interfacing. The handle value is not normally required by the EZConsole public API.

Example:

```
CALL EZCONSGETSTDHANDLE(1,HANDLE,RETCODE) !Get standard input
CALL EZCONSGETSTDHANDLE(2,HANDLE,RETCODE) !Get standard output
CALL EZCONSGETSTDHANDLE(3,HANDLE,RETCODE) !Get standard error
```

Index

Name: EZConsWriteText(text,retcode)

Alternate Name: EZWTXT()

Procedure Type: Subroutine

Arguments:

- `CHARACTER(*), INTENT(IN) :: TEXT !character string to write`
- `INTEGER, INTENT(OUT)     :: RETCODE !integer return code`
  - 0, 2, 4 `(see return code definition table)`

Description: Writes a text string at the current cursor position as set by the EZConsSetCurPos() procedure.

Example:

```
CALL EZCONSWRITETEXT('Text to write',RETCODE) !Text to write
```

Index

Name: EZConsReadText(text,retcode)

Alternate Name: EZRTXT()

Procedure Type: Subroutine

Arguments:

```
?  CHARACTER(*), INTENT(IN) :: TEXT !character string to write
?  INTEGER, INTENT(OUT)     :: RETCODE !integer return code
      o  0, 2, 4 (see return code definition table)
```

Description: Reads a text string from the current cursor position as set by the EZConsSetCurPos() procedure.

Example:

```
CALL EZCONSREADTEXT(TEXT,RETCODE) !Read text from the console
```

Index

Name: EZConsSetPrompt(text,retcode)

Alternate Name: EZSP(text,retcode)

Procedure Type: Subroutine

Arguments:

- `CHARACTER(*), INTENT(IN) :: TEXT`
- `INTEGER, INTENT(OUT) :: RETCODE`
  - `0, 2 (see return code definition table)`

Description: Sets the prompt string value. This procedure does not write the prompt string to the console. Subsequent reads from the console will echo the prompt string set using this procedure. The prompt string is truncated internally to a maximum length of 80 characters. The prompt defaults to a blank string and is interpreted as null. Set the prompt string to blanks to disable the prompt.

Example:

```
CALL EZCONSSETPROMPT(“c:\”,RETCODE) !Set the prompt string
```

Index

Name: EZConsSetMode(ModeType,Mode,State,RetCode)

Alternate Name: EZSM(ModeType,Mode,State,RetCode)

Procedure Type: Subroutine

Arguments:

- `INTEGER, intent(in)  :: Modetype`
    - `0 = Input`
    - `1 = Output`
- `INTEGER, intent(in)  :: Mode`
    - `ModeType = 0`
        - `0 = Default (Line+Echo+Processed+Mouse)`
        - `1 = Line mode input`
        - `2 = Echo input`
        - `3 = Processed input`
        - `4 = Window input`
        - `5 = Mouse input`
    - `ModeType = 1`
        - `0 = Default (Processed+Wrap)`
        - `1 = Processed output`
        - `2 = Wrap at EOL output (Default`
- `INTEGER, intent(in)  :: State`
    - `0 = Disable`
    - `1 = Enable`
- `INTEGER, intent(out) :: RetCode`
    - `0, 2, 27, 28, 29, 30, 31, 32 (see return code definition table)`

Description: Sets the console mode. This procedure is used to enable or disable various mode state flags. Echo input implies Line input and so EZConsole will force Line mode if Echo mode is requested. Other states are independent. To enable single character read/return, disable line mode. To disable control character processing, disable Processed input.

Example:

```
CALL EZCONSSETMODE(0,0,1,RETCODE) !Set input mode to defaults
```

Index

Name: EZConsGetMode(ModeType,Mode,State,RetCode)

Alternate Name: EZSM(ModeType,Mode,State,RetCode)

Procedure Type: Subroutine

Arguments:

- INTEGER, intent(in) :: Modetype
  - 0 = Input
  - 1 = Output
- INTEGER, intent(in) :: Mode
  - ModeType = 0
    - 0 = Default (Line+Echo+Processed+Mouse)
    - 1 = Line mode input
    - 2 = Echo input
    - 3 = Processed input
    - 4 = Window input
    - 5 = Mouse input
  - ModeType = 1
    - 0 = Default (Processed+Wrap)
    - 1 = Processed output
    - 2 = Wrap at EOL output (Default
- INTEGER, intent(out) :: State
  - 0 = Disabled
  - 1 = Enabled
- INTEGER, intent(out) :: RetCode
  - 0, 2, 27, 33 (see return code definition table)

Description: Returns the console mode. This procedure is used to query the enable or disable state of various console mode flags. Query for default state implies multiple bit states as indicated (the normal initial state of a newly opened console). A logic one in STATE will therefore indicate that all of the default mode states are enabled.

Example:

```
CALL EZCONSGETMODE(0,0,STATE,RETCODE) !Query default settings
CALL EZCONSGETMODE(1,2,STATE,RETCODE) !Query output Wrap state
```

Index

Name: EZConsSetCurState(SizePercent,Visibility,RetCode)

Alternate Name: EZSCS(SizePercent,Visibility,RetCode)

Procedure Type: Subroutine

Arguments:

- `INTEGER, INTENT(IN)      :: SizePercent`
  - `Range:  0 to 100 percent from bottom of cell`
    - `Default:  25 percent`
- `INTEGER, INTENT(IN)      :: Visibility`
  - `0 = Invisible`
  - `1 = Visible (default)`
- `INTEGER, INTENT(OUT)     :: RETCODE !integer return code`
  - `0, 2, 34, 35, 36 (see return code definition table)`

Description: Sets current cursor size as a percentage of a character cell beginning from the bottom of the cell. The horizontal size or position within the cell cannot be changed, only the number of scan lines filled from bottom to top.

Example:

```
CALL EZCONSSETCURSTATE(100,1,RETCODE) !Create a block cursor
```

Index

Name: EZConsGetCurState(SizePercent,Visibility,RetCode)

Alternate Name: EZGCS(SizePercent,Visibility,RetCode)

Procedure Type: Subroutine

Arguments:

```
? INTEGER, INTENT(OUT)      :: SizePercent
     o  Range:  0 to 100 percent from bottom of cell
          ✍ Default:  25 percent
? INTEGER, INTENT(OUT)      :: Visibility
     o  0 = Invisible
     o  1 = Visible (default)
? INTEGER, INTENT(OUT)     :: RETCODE !integer return code
     o  0, 2, 33 (see return code definition table)
```

Description: Retrieves the current cursor size as a percentage of a character cell (beginning from the bottom of the cell) and the cursor visibility state.

Example:

```
CALL EZCONSGETCURSTATE(SizePercent,Visibility,RETCODE) !Query
the cursor state
```

Index

Name: EZConsSetCurPos(xpos,ypos,retcode)

Alternate Name: EZSCP(xpos,ypos,retcode)

Procedure Type: Subroutine

Arguments:

- `INTEGER, INTENT(IN) :: XPOS !cursor x cell position`
    - `XPOS = 0 – 79 columns, larger values will wrap to next line`
- `INTEGER, INTENT(IN) :: YPOS !cursor y cell position`
    - `YPOS = 0 – OS dependent maximum buffer size`
- `INTEGER, INTENT(OUT)     :: RETCODE !integer return code`
    - `0, 2, 4 (see return code definition table)`

Description: Sets current cursor position to position XPOS, YPOS. The next EZCONSWRITETEXT call will begin writing at this character cell position. Subsequent writes are appended to the position at conclusion of the write. To overwrite the same position requires a repeated call to EZCONSSETCURPOS to reset the current position.

Example:

```
CALL EZCONSSETCURPOS(1,1,RETCODE) !Set cursor position to 1,1
```

Index

Name: EZConsSetTextAtt(foreatt,backatt)

Alternate Name: EZSTA()

Procedure Type: Subroutine

Arguments:

- INTEGER, INTENT(IN) :: foreatt, backatt !Cell attribute settings
  - 0 = BLACK
  - 1 = BRIGHT WHITE
  - 2 = BRIGHT RED
  - 3 = BRIGHT GREEN
  - 4 = BRIGHT BLUE
  - 5 = BRIGHT YELLOW
  - 6 = BRIGHT PURPLE
  - 7 = BRIGHT TURQUOISE
  - 8 = DIM WHITE (GRAY)
  - 9 = DIM RED
  - 10 = DIM GREEN
  - 11 = DIM BLUE
  - 12 = DIM YELLOW
  - 13 = DIM PURPLE
  - 14 = DIM TURQUOISE
- INTEGER, INTENT(OUT)   :: RETCODE !integer return code
  - 0, 2, 4, 37, 38 (see return code definition table)

Description: Sets text attribute at the current cursor position. This affects all text subsequently written by EZCONSWRITETEXT until EZCONSSETTEXTATT is called again.

Example:

```
CALL EZCONSSETTEXTATT(2,1,RETCODE) !Set color to red foreground
on white background.
```

Index

Name: EZConsCtrlEvent (EventType,RetCode)

Alternate Name: EZEVNT(EventType,RetCode)

Procedure Type: Subroutine

Arguments:

- `INTEGER, INTENT(IN) :: EVENTTYPE !event type`
  - `1 = control-c`
  - `2 = control-break`
- `INTEGER, INTENT(OUT)     :: RETCODE !integer return code`
  - `0, 2, 4, 39 (see return code definition table)`

Description: Generates a control event of the indicated type.

Example:

```
CALL EZCONSCTRLEVENT(1,RETCODE) !Issue a control-c interrupt
```

Index