The Basis System, part 4

The Basis Development Team

November 13, 2007

Lawrence Livermore National Laboratory
Email: basis-devel@lists.llnl.gov

COPYRIGHT NOTICE

DISCLAIMER

DOE Order 1360.4A Notice

UCRL-MA-118543

CONTENTS

CHAPTER

ONE

The Basis System

1.1 Environment Variables

Before using Basis, you should set some environment variables as follows.

- `BASIS_ROOT` should contain the name of the root of your Basis installation, `/usr/apps/basis` for example.
- `MANPATH` should contain a component `$BASIS_ROOT/man`.
- Your path should contain a component `$BASIS_ROOT/bin`.
- `DISPLAY` should contain the name of your X-Windows display, if you will be doing X-window plotting.
- `NCARG_ROOT` should contain the name of the root directory of your NCAR 4.0.1 or later distribution, if you have it.

Check with your System Manager for the exact specifications on your local systems.

1.2 Basis Is Both a Program and a Development System

Basis is a system for developing interactive computer programs in Fortran, with some support for C and C++ as well. Using Basis you can create a program that has a sophisticated programming language as its user interface so that the user can set, calculate with, and plot, all the major variables in the program. The program author writes only the scientific part of the program; Basis supplies an environment in which to exercise that scientific programming, which includes an interactive language, an interpreter, graphics, terminal logs, error recovery, macros, saving and retrieving variables, formatted I/O, and on-line documentation.

`basis` is the name of the program which results from loading the Basis System with no attached physics. It is a useful program for interactive calculations and graphics. Authors create other programs by specifying one or more packages of variables and modules to be loaded. A package

is specified using a Fortran source and a variable description file in which the user specifies the common blocks to be used in the Fortran source and the functions or subroutines that are to be callable from the interactive language parser.

Basis programs are *steerable applications*, that is, applications whose behavior can be greatly modified by their users. Basis also contains optional facilities to help authors do their jobs more easily. A library of Basis packages is available that can be added to a program in a few seconds. The programmable nature of the application simplifies testing and debugging.

The Basis Language includes variable and function declarations, graphics, several looping and conditional control structures, array syntax, operators for multiplication, dot product, transpose, array or character concatenation, and a stream I/O facility. Data types include real, double, integer, complex, logical, character, chameleon, and structure. There are more than 100 built-in functions, including all the Fortran intrinsics.

Basis' interaction with compiled routines is particularly powerful. When calling a compiled routine from the interactive language, Basis verifies the number of arguments and coerces the types of the actual arguments to match those expected by the function. A compiled function can also call a user-defined function passing arguments through common.

1.3 About This Manual

The Basis manual is presented in several parts:

I. Running a Basis Program, A Tutorial

II. Basis Language Reference

III. EZN User Manual: The Basis Graphics Package

IV. The EZD Interface

V. Writing Basis Programs: A Manual For Program Authors

VI. The Basis Package Library

VII. MPPL Reference Manual

The first three parts form a basic document set for a user of programs written with Basis. The remainder form a document set for an author of such programs.

Basis is available on most Unix and Unix-variant platforms. It is not available for Windows or Macintosh operating systems.

A great many people have helped create Basis and its documentation. The original author was Paul Dubois. Other major contributors, in alphabetical order, have been Robyn Allsman, Kelly Barrett, Cathleen Benedetti, Stewart Brown, Lee Busby, Yu-Hsing Chiu, Jim Crotinger, Barbara Dubois, Fred Fritsch, David Kershaw, Bruce Langdon, Zane Motteler, Jeff Painter, David Sinck,

Allan Springer, Bert Still, Janet Takemoto, Lee Taylor, Susan Taylor, Peter Willmann, and Sharon Wilson. The authors of this manual stand as representative of their efforts and those of a much larger number of additional contributors.

Send any comments about these documents to ”basis-devel@lists.llnl.gov” on the Internet.

CHAPTER

TWO

Introduction to EZD

2.1 Functionalities of EZD

The **EZD** package is a set of Fortran utilities for controlling graphical devices in programs that use the **National Center of Atmospheric Research(NCAR)** Graphics Library. Graphic devices supported by the EZD depend on the underlying **Graphics Kernel System(GKS)**. A computer system with **Advanced Technology Center(ATC) GKS**, the devices supported are **Computer Graphic Metafile(CGM)** files, **PostScript** files, **Xwindows**, and **Tektronix Graphics terminals**. For the computer system that has only the **NCAR GKS**, the **NCAR CGM** file is supported and additional **Xwindows** is provided if NCAR3.2 or later version is used. (For CGM and PS we use "file" or "device" interchangeably.) The EZD package also has subroutines to properly start and to end the plots, to make a frame advance, to do graphics in quadrant mode, to set up color tables, to write the log files, and to record the error log. The EZD package has stub subroutines *ezchook*, *ezcdispl* which can be replaced by a customized function routines to perform actions when a frame is advanced. A set of parameters are used in EZD to perform some specific controls such as the maximum number of frames in a CGM file, Xwindow display location, problem name of a run etc. A user may inquire the current settings of control parameters and set their values. For **UNICOS** users, EZD also provides the setup utilities to specify *Boxid*, *Security Level*, and *Give/Keep*. These setups will be used to generate proper *lpr* commands to handle user's output files.

2.2 Incorporating EZD in your program

The EZD package was designed that a client program does not need to have Basis to be loaded to generate the executables. This reduces the size of the program and broadens its usability. But because the lack of Basis support, it is necessary for the client program to call some initialization programs when EZD is invoked. The client program also needs to call functional routines to properly close the files when the client program ready to stop the execution. It needs to provide some additional functions to rendering the graphics and other related tasks. The client programs of the EZD should load the library *libezd.a* during the linking process. It is located in `/usr/local/basis/bin/lib`. You will also need to load with the appropriate NCAR and ATC libraries for your site. Refer to Section 1.1 "Environment Variables" in Chapter 1 for the list

of environment variables needed.

Furthermore, additional libraries and include files need to be specified on the command line for proper compilation of code calling EZD routines.

In the following descriptions, `[flags]` should be replaced with flags for the compiler that the user desires, `<atc device libraries>` with the desired ATC device libraries, and `<file>` with the file or files of appropriate type.

On a SUN4, this requires the following format during partial compilation *with* ATC GKS:

```
f77 -c [flags] -Bdynamic -I/usr/local/gks4101 <file.f>
```

and during linking:

```
f77 [flags] -Bdynamic -L/usr/local/gks4101            \
<files.o> /usr/local/basis/bin/lib/libezd.a          \
-lncarg -lgksflb -lgkswiss -lgksgksm                 \
<atc device libraries\> -lgksmsc -lncarv -lncarg_loc  \
-lX11
```

On a Sun4, this requires the following format during partial compilation *without* ATC GKS:

```
f77 -c [flags] -Bdynamic <file.f>
```

and during linking:

```
f77 [flags] -Bdynamic <files.o>               \
/usr/local/basis/bin/lib/libezd.a -lncarg     \
-lgksflb -lgkswiss -lgksgksm -lgksmsc -lncarv \
-lncarg_loc -lX11
```

On an HP700, this requires the following format during partial compilation *with* ATC GKS:

```
f77 -c  [flags] -I/usr/local/gks4101 <file.f>
```

and during linking:

```
f77 [flags] -Wl,-L/usr/local/gks4101 <files.o>       \
/usr/local/basis/bin/libezd.a                       \
-lncarg -lgksflb -lgkswiss -lgksgksm                \
<atc device libraries\> -lgksmsc -lncarv -lncarg_loc \
-lX11 -lm -lBSD
```

On an HP700, this requires the following format during partial compilation *without* ATC GKS:

```
f77 -c  [flags] <file.f>
```

and during linking:

```
f77 [flags] <files.o> /usr/local/basis/bin/libezd.a  \
-lncarg -lgksflb -lgkswiss -lgksgksm -lgksmsc        \
-lncarv -lncarg_loc -lm -lBSD
```

The UNICOS versions of these commands are similar. Instead of `/usr/local/gks330` we use `/usr/local/lib/ATC_GRAFPAK-GKS/gks330`.

2.3 Initialize EZD

Before invoking the functions of EZD, the client program should call `ezdinit` to initialize the EZD. It sets the parameters to its proper values. The functions of EZD depend on these values to behave accordingly.

2.4 Setting Devices

In a computer system with only **NCAR GKS** installed, a user can open CGM files to store graphic output. Addtional device support for the Xwindows when NCAR3.2 or later version is used. The CGM files created by the NCAR GKS are not standard CGM files. The NCAR CGM files can be used as input to the NCAR graphic utilities such as *ncgm2cgm*, *idt*, *ctrans* etc. For example, the utility *ncgm2cgm* translates a NCAR CGM file into a standard CGM file. *idt* lets you view the NCAR CGM file interactively. Please refer to the NCAR manuals for details about its graphic utilities. If **ATC GKS** is installed in the computer system, a user can open multiple devices and direct the graphics output to different devices. One application of these capabilities is, for example, a user can open several Xwindows at the same or different workstations, and display frames in different windows for comparison.

The graphic devices have several states: *opened*, *closed*, *active* and *inactive*. Before a device can be used, it has to be opened, then activated. Only the active devices will receive graphic outputs. Before closing a device, the device needs to be deactivated.

The subroutine *ezcdodev* is the top layer of user interface to control the devices. The calling sequence is:

```
call ezcdodev("device-arg","action-arg","modifier_arg")
```

The argument *device-arg* specifies the device that client program intends to control. The possible values are `cgm`, `ps`, `win`, `tv`, or `tek`. Here `win` and `tv` are synonymous. The argument *action-arg* indicates the actions that the user wants to perform on the device specified. The actions can be `on`, `off`, `close`,`send` and `colormap`. The argument *modifier-arg* is used to specify additional properties of the device. The values are `color`, `mono`, *window name, and colormap name*.

The underlying subroutines for the device controls are the subroutines *ezccgm*, *ezcps*, *ezcwin*, and *ezctek* control the **CGM**, **PostScript** files, the **Xwindow** and the **Tektronix Graphic Terminal** respectively. The action `on` *opens* a device if the device has not been opened and then *activates* the device. If the device already opened, the command `on` *activates* it. It has no effect on the device if it is currently active. The action `off` *deactivates* an open device (but the linkage to the device for controlling still exists). The action `close` *deactivates* and then *closes* the device. Issuing the commands `close` or `off` to a non-existing device causes an error. The action `send` *sends the current frame* to the specified device. If the target device is a CGM or a PS file, the send action turns `on` the device (even the device has not been opened before), `send` s a frame, and then turns the device `off`. If the target device is a Xwindow or a Tektronix graphic terminal, then the current frame is *resent* to the device provided the target device is *active*. The action `colormap` sets the named colormap to the device. If the device does not exist at the time when colormap action was invoked, the EZD creates the device and then set its colormap.

Due to the constraints of Xwindows driver, the colormap setting for Xwindows works differently than other devices. If ATC GKS is the underlying gks package, only the first window can be set to the desired colormap and then all sebsequent windows will inherit the colormap set by it. if NCAR GKS is used, no colormap setting is implemented because the possible program crash induced by setting the Xwindow colormap.

The *modifier-arg* is used to specify additional properties of the device. A user can use the modifier to override the default setting for the device. The command modifier `color`, `mono` overrides the default setting for CGM or PS files. CGM files default to “color” and PS files are default to “mono”. One caution with the use of color PostScript file: if it is printed at a black and white printer, the color attribute of the graphics will be plotted in different line styles; as dotted or dashed lines which may change the looks of the graphics without user’s intention. The *modifier-arg* associated with `colormap` action is used to specify the name of the colormap. There are 18 colormaps to choose from. The first 16 colormaps are named as ”`idl1`”, ”`idl2`”,...,”`idl16`”. Those colormaps defintions are borrowed from **IDL** with its **RGB**(Red,Green,Blue) settings. The 17th colormap is named as ”`mycolormap`” for user defined colormap. A user specifies `ezcred`, `ezcgreen`, and `ezcblue` arrays of RGB values to be used in the colormap. The default colormap which varies the color spectrum from blue to green to red. Any colormap name which does not match above mentioned seventeen colormap names will result to use the default colormap. An example to setup a colormap for the CGM file, call `ezcdodev("cgm","colomap","idl1")`. Another usage of the command modifier is to name the Xwindow when it is opened. The name of a window is used to identify it in future actions. As an example, `call ezcdodev("win","on","FirstWindow")` opens an Xwindow in your default workstation and names the window `FirstWindow`. If multiple windows were used, *ezcwin* activates only *one* window. The latest window with action `on` is the *active* window. The

activated window receives graphic output. (If multiple *displays* were used, you may have one active window in each *display*.) Because of this arrangement, a user can direct the graphic output to different windows in order to view and to compare the graphic results interactively.

When a CGM or a PS device is opened, the file name will be determined by the file root name "`fnroot`" parameter with extension `.cgm` or `.ps` correspondingly. The default file root name is `"problem"`. The subroutine *ezccgm* and *ezcps* check (in the current directory) the existence of specified files (either by the user or by the program defaults). If the file already exists, the subroutine tries to append a sequential three digit number to the root name to make a new file name. This avoids clobbering the existing files e.g. problem.001.cgm, problem.002.cgm etc. EZD will create a CGM log file or a PS log file for each CGM/PS file created to record the frame count and all graphic commands in each frame. The naming scheme for the log files is the same as for CGM or PS files but with extension `.cgmlog` or `.pslog`.

2.5 Starting and Ending the plots

The subroutines *pltstart* and *pltend* are provided to properly open and close graphic device(s) for receiving plot commands. They are argumentless routines.

```
call pltstart
call pltend
```

The *pltstart* checks the existence of active devices. It does nothing if an active device already exists. It opens a CGM file automatically if no active devices were found. The *pltend* closes all devices and log files. *pltstart* should be called before any plotting commands and the client program should call *pltend* to terminate graphics before program ends. Explicitly close device by calling *ezcdodev* or implicitly by calling *pltend* is important to leave the device in *proper state*. For example, if CGM file was not properly closed, it will cause command `lpr` to fail on the UNICOS systems.

2.6 Quadrant mode

EZD has utilities to control the plots in *quadrant mode*. Most quadrant routines have companion routines for inquiring the current settings.

The subroutine `ezcdquad(xmin,xmax,ymin,ymax)` defines the quadrant reference box. *xmin*, *xmax*, *ymin*, *ymax* are reals in [0.,1.] as the bounding values of a rectangular region in the frame which has the values (0.,1.,0.,1.). The default reference box of quadrant is the **whole** frame.

The routine `ezcidquad(v1,v2,v3,v4)` returns the current boundary of the quadrant reference box.

The quadrants are defined in a customary way by **bisecting** both the length and the width of the reference box. The quadrant 1 is the upper-left quarter. The quadrant 2 is the upper-right quarter. The quadrant 3 is the lower-left quarter and the quadrant 4 is the lower-right quarter.

The `subroutine ezcquad(n)` where n is one of the following integers: 1, 2, 3, 4, 12, 13, 24, 34, and 1234 defines a combination of quadrants represented by each digit. For example, `call ezcquad(12)` defines the region combined with quadrant 1 and 2 to draw the graphic output. `call ezcquad(1234)` is the same as the original whole reference box.

Another example, after called `ezddquad(0.0,0.5,0.5,1.0)`, then the call to `ezcquad(1)` would set the first quadrant of now just defined reference box, i.e. the most upper left one-sixteenth of the original frame for plotting graphics.

`ezcsquad(xmin,xmax,ymin,ymax)` allows a user to set an arbitrary rectangular portion of the **whole** frame bounded by the specified arguments to plot the graphics. The companion `subroutine ezciquad(v1,v2,v3,v4)` can be used to inquire the current quadrant boundary. The call to `subroutine ezcrquad` will restore the reference box defined by *ezcdquad* as the plot region.

2.7 Frame Advance

The frame advance logic is complicated because of the need to lag the actual frame advance for interactive window use and the need to handle quadrant/non- quadrant graphics. The action routine *ezcfradv* does the frame advance if the flag *ezcxn* has been set to 1 (i.e. YES). The routine *ezcnq* displays the current picture by calling *ezcshowf* which calls *ezcdispl* to flush out the graphic contents and calls *ezcfradv* to advance the frame if needed. The *ezcdispl* to flush out the graphic contents is a dummy routine in the EZD and need be supplied by the client program. For each new frame, the client program should call *ezcnf* which sets the flag *ezcxn* to 1 and calls *ezcnq*. After the frame advanced, *ezcnf* resets the flag *ezcxn* to 0 and gets it ready for the next event.

2.8 Error Logging

Error logging facility in the EZD library is performed through the *ezcerror* subroutine. `ezcerror("msg",level)` accepts a quoted string and an integer error severity level as its arguments. The subroutine writes the string to the **standard error file**. **Three levels** of error severity are defined as follows: level one is a commentary, level two is a minor abnormality, the program continues to execute, level three is a fatal error, and it calls the user defined error handler *ezcdie* (the client program may wish to provide this) to manage alternative actions response to the sever errors.

When a user calls this routine to record the error message, he/she also needs to determine the severity of the error and assign the severity level accordingly.

2.9 Color Table

`ezcoltb(indlo,indhi,red,green,blue)` defines color tables

for all **active** devices. *indlo* is the integer for *lower bound* of color indices, and *indhi* is the *upper bound* of color indices, and *red, green, blue* are arrays of fractions of full intensity of red, green, and blue color range in [0., 1.]. With a user defined color table, applications may use special colors to convey some physics quantities such as color cells.

2.10 Set a Predefined Colormap/Color Table

`ezcdodev("device-type","colormap","colormap-name")`

The routine `ezcdodev` can be used to setup a special colormap for a device. If the device does not exist (i.e. has not been `opened`) at the time of the subroutine call, then **EZD** will *open* the device, *activiate* the device and then *setup the requested colormap*. If the device already exits, then colormap is *reset* and then *returned* to its *original state* (e.g. active, inactive) just before the `ezcdodev` was invoked. There are sixteen predefined colormaps named "`idl1`","`idl2`",...,"`idl16`" which are borrowed form the **IDL**'s colormap definitions. The "`idl1`" colormap is the *greyscale* colormap, so the user may use "`greyscale`" as the colormap name. The "`idl2`" has alias "`bluescale`", but some idl colormaps have no proper aliases. The seventeenth colormap named "`mycolormap`", is defined by the user's specifications to the arraysof `ezcred`, `ezcgreen`, and `ezcblue` for its RGB values. Any other name used for colormap will result to use the default colormap, which is a colormap varies the color spectrum from blue to green to red.

Some exceptions when Xwindow and postscript files are involved. The color postscript file and mono postscript file can not switch from one to another. Only color postscript file can change its colormap. The Xwindow device driver from ATC allows the *first* Xwindow to set its colormap first time when it is brought up then the subsequent Xwindows will share the same colormap. For a single Xwindow to change its colormap, the EZD actually closes the window then opens it again with the new colormap. The NCAR Xwindow driver currently will cause the application program to crash if change colormap is attempted. So EZD will not allow any colormap setting for Xwindows in the applications without ATC GKS.

2.11 Box, Security Level, and Give/Keep

For the programs run on the **UNICOS** systems at **Livermore Computer Center**, it is required to specify a *Boxid*, *Security Level*, and *Give/Keep* for the output. The subroutines *ezcdobox*, *ezcdolev*, and *ezcdogk* are provided to set them up. The calling sequence for these routines is

```
call ezcdoxxx("string")  where xxx=box, lev, or gk
```

The argument is a string or a variable with string value. *ezcdobox* accepts a three character string as the argument. The first character is an alphabet among a-z and A-Z, then followed by two alphanumeric characters. *ezcdolev* accepts the argument with the possible values of `uncl`, `UNCL`, `pard`, `PARD`, `crd`, `CRD`, `srd`, `SRD`, or numerical characters `1`, `2`, `4`, and `5` which corresponding to *unclassified*, *pard*, *crd* and *srd*. The argument for *ezcdogk* takes either `give` or `keep`. It defaults to `keep` if not explicitly specified.

When the CGM file closes, the above setups will be used to send a proper `lpr` command to the **UNICOS** operation system to produce fiche. For example, say the Boxid=u51, Severity Level=pard and Give/Keep=keep, the EZD will issue the following command string:

```
lpr -P105 -Bu51 -Spard problem005.cgm
```

when the CGM file problem005.cgm is closed. A default jobname same as the CGM file name is also provided to the `lpr` command to print it on the fiche.

2.12 Stub Routine - ezchook

A stub routine named *ezchook* is called by the *ezcfradv* subroutine with syntax

```
call ezchook("string1","string2").
```

The *ezcnf* calls *ezcfradv* to advance a frame. A user can substitute this stub routine *ezchook* with his/her own subroutine to perform customized tasks when each frame advances.

2.13 Access to Parameters - ezcseti, ezcsetr, ezcsetc, ezcgeti, ezcgetr, ezcgetc

A set of parameters in the EZD package provides special controls to the graphic devices such as maximum number of frames in a CGM file, the Xwindow display workstation other than the default environment variable setup, the root name of the problem etc. Six routines *ezcseti*, *ezcsetr*, and *ezcsetc* are used to set integer, real, character parameters correspondingly. The subroutine *ezcgeti*, *ezcgetr* , and *ezcgetc* are used to inquire and to retreive the current value of a parameter. To access an *array parameter*, the user needs to specify the *index* of the array element by calling `call ezcseti("ezcpidx",ivalue)` before the call for “set” or “get” the parameter.

The arguments to these subroutines all have the same format,

```
call ezcsetx("parameter-name",parameter-value), where x=i,r,c
call ezcgetx("parameter-name",parameter-variable), where x=i,r,c
```

the first argument contains the name of the parameter variable enclosed by double quotes, the second argument is the value you want to set for the parameter or the variable to receive the value of the parameter.

If the user gives a non-existing parameter to the subroutine (including misspelled name), the subroutine produces an error message and then exit. The client program needs to define an error handler to respond this situation.

Following is a list of control parameters, their function and default value:

name	**brief Descriptions**	**default value**
ezccgmc	maximum number of frames in a CGM file	242
ezcpsc	maximum number of frames in a PS file	242
ezcdisp	string to specify Xwindow display	yourhost:0.0
ezcwinsz	string to specify size of Xwindow	-dx -dy -u
ezcwinlb	string to name an Xwindow	blank string
numcol	number of color indices in a color table	192
fnroot	root name used for the CGM/PS files and log files	”problem”
debcolr	debugging flag for color problems	0

-

CHAPTER

THREE

List of Subroutines

This chapter contains a list of subroutines and their arguments. The subroutines are sorted by name. A brief description of each routine is also attached.

3.1 ezcapsfx

Calling Sequence

```
subroutine ezcapsfx(namer,ftype,fnsfx,fname,succ)
```

Description

Append suffix to a given file root name. This routine is called by `ezcwin` and `ezcps` to open a file with unique name.

Arguments

`namer` character*(80), the file root name

`ftype` character*(16), the file type, e.g. cgm, cgmlog, ps, pslog

`fnsfx` integer, the file name suffix as integer

`fname` character*(80), returned unique file name

`succ` logical, success flag of the subroutine process

Procedure

Append the fnsfx to namer if fnsfx is less than 999, otherwise change fnsfx to 1 and extend the file root name with ending "." then append the new fnsfx. If the extension of the root name failed, the return flag succ is set to false.

3.2 ezccgm

Calling Sequence

```
subroutine ezccgm(iflag,istring)
```

Description

Control the CGM devices. This is an underlying subroutine called by `ezcdodev`

Arguments

`iflag` character*(*),action-command-string, the possible values are `on`, `off`, `send` or `close`.

`istring` character*(*), command-modifier-string, the possible values are `color`, `mono`.

Procedure

For the action command “on”:

Open and activate a CGM device if no existing CGM device. Assign a proper file name for the CGM file, and open a CGM log file if it does not exist. Activate a CGM device if it has been deactivated. No action if an active CGM file exists.

For the action command “off”:

Deactivate the current active CGM device. No action if no active CGM device.

For the action command “send”:

Turn the CGM device “on”, “send” a frame, then turn the CGM device “off” (“send” implies “on” so it may open (i.e. create) a CGM file)

For the action command “close”:

Deactivate and then close the CGM file. If the client program runs on UNICOS at LC, proper `lpr` command will be send to the operating system to generate fiche from the close CGM file.

The command-modifier “color”, “mono” specifies the CGM file color.

3.3 ezccidx

Calling Sequence

```
subroutine ezccidx(iws,iwstype)
```

Description

Initialize the color indices table, define foreground and background colors, define a set of named colors with special indices.

Arguments

iws integer, the workstation id associated to this special color table

iwstype integer, the type of this workstation

Procedure

Define the color index 0 and color index 1 with RGB values. The color index 0 is the background color and the color index 1 is the foreground color. The subroutine sets “black” as the background and “white” as the foreground if the variable `bakcol` has value 0 and reverse the setting if `bakcol` has value 1. Other color indices and associated RGB values are defined in the process. This setting is closely coupled to `ezcctoi` subroutine call. The color index returned by `ezcctoi` by the giving color name has the RGB value defined in this subroutine.

3.4 ezcclear

Calling Sequence

```
subroutine ezcclear
```

Description

A dummy routine called by ezcnf. The original usage is to clear the attribute settings. Since this routine is called by every frame advance, it is user replaceable to do some customized tasks.

Arguments

none

Procedure

none

3.5 ezccoltb

Calling Sequence

```
subroutine ezccoltb(indlo,indhi,red,green,blue)
```

Description

Set a set of color indices with the given RGB values.

Arguments

indlo integer, lower bound of the color indices

indhi integer, higher bound of the color indices

red, green, blue – real(indhi-indlo+1), arrays of reals in [0., 1.] of fractions of full intensity of the red, green, blue colors.

Procedure

Set the color table with indices vary from indlo to indhi. Each index associates a color defined by the corresponding indexed array element of red, green, and blue.

3.6 ezcctoi

Calling Sequence

```
subroutine ezcctoi(colorname)
```

Description

Based on the given colorname returns the corresponding color index in the color table.

Arguments

`colorname` character*(32), colorname string

Procedure

Search the colorname array for the given colorname. If a name matched, returns the index in the colorname array. If no name is matched, returns the index 1.

3.7 ezcdodev

Calling Sequence

```
subroutine ezcdodev(devtype,arg1,arg2)
```

Description

A top layer user interface routine to control the graphics devices. This subroutine redirects device control commands to the specific device control routine such as `ezcwin`, `ezcps`, `ezccgm` etc.

Arguments

`devtype` character*(*), the device type that will receive the control commands, the possible values are `cgm`, `ps`, `win`, `tv`, and `tek`.

arg1 character*(*), the action-command-string, the valid commands are `on`, `off`, `send` or `close`

arg2 character*(*), the command-modifier-string, the possible values are `mono`, `color` or a *window name*. When the device is a *cgm* or a *ps*, user can specify color options for the device. The *cgm* is default to “color” and the *ps* is default to “mono”. When the device is a *win*, this command-modifier-string can be used to set the window name.

Procedure

Based on the given device type, this subroutine redirects the command and its modifier to call the subroutine that handles this special device. For example,

```
call ezcdodev("cgm","on","color")
```

will call the underlying subroutine `ezccgm("on","color")` to carry out its command.

3.8 ezcsquad

Calling Sequence

```
subroutine ezcsquad(v1,v2,v3,v4)
```

Description

Set a rectangular portion of the frame to plot the graphics, frame size will not be changed. (vs. *ezcframe* which resets the frame boundary)

Arguments

v1, v2, v3, v4 real(Size4), the xmin, xmax, ymin, ymax values in [0., 1.] of the rectangular region of the frame

Procedure

This routine calls `ezcnq` to handle frame advance and flush out graphic contents if necessary. After the call to `ezcnq`, it sets the portion of frame as defined by `v1`, `v2`, `v3`, `v4` to output the graphics.

3.9 ezciquad

Calling Sequence

```
subroutine ezciquad(v1,v2,v3,v4)
```

Description

Inquire the current quadrant boundaries.

Arguments

v1, v2, v3, v4 real(Size4), the xmin, xmax, ymin, ymax values in [0., 1.] of the rectangular region of the frame set by the last call to ezcsquad.

Procedure

Retrieve the boundary values from the common block.

3.10 ezcquad

Calling Sequence

```
subroutine ezcquad(iquad)
```

Description

Based on the reference box for quadrants, ezcquad(iquad) will set plotting quadrant to the combinations of customary quadrants 1, 2, 3, 4 start from the upper-left corner, upper-right corner, lower-left corner and lower-right corner.

Arguments

iquad integer, one of the following values 1, 2, 3, 4, 12, 13, 24, 34, and 1234

Procedure

This is a short cut to define plotting quadrant to one of the customary quadrant or a combination of customary quadrants by calling `ezcsquad` with proper xmin, xmax, ymin and ymax.

3.11 ezcdquad

Calling Sequence

```
entry ezcdquad(v1,v2,v3,v4)
```

Description

Change the default reference box for quadrants to (v1, v2, v3, v4)

Arguments

v1, v2, v3, v4 real(Size4), xmin, xmax, ymin, ymax values in [0., 1.] with respect to the full frame

Procedure

Set boundary limits to v1, v2, v3, v4 and call NCAR “set routine”

3.12 ezcidquad

Calling Sequence

```
entry ezcidquad(v1,v2,v3,v4)
```

Description

Inquire the default reference box for quadrants.

Arguments

v1, v2, v3, v4– real(Size4), xmin, xmax, ymin, ymax values in [0., 1. respect to the full frame]

Procedure

Retrieve the values stored in the common block for reference box for quadrants.

3.13 ezcrquad

Calling Sequence

```
entry ezcrquad
```

Description

Restore quadrant to the default reference box for quadrants. Refer to ezcdquad

Arguments

none

Procedure

Reset the view port to the last defined reference box for quadrants and set up linear ndc transformation into viewport by calling NCAR “set routine”

3.14 ezcdie

Calling Sequence

```
subroutine ezcdie
```

Description

This is a routine called by EZD routines when abnormal conditions are encountered. This routine raises signal SIGUSR1. The client program should provide alternative actions when the signal is received. Hence this is the interface for the client program exception handler.

Arguments

none

Procedure

Raise the signal SIGUSR1 by calling C routine `raise`.

3.15 ezcdispl

Calling Sequence

```
subroutine ezcdispl
```

Description

This is a stub routine that the client program should replace with its real procedure to flush out the graphic contents. The routine is called by `ezcfradv` which in turn is call by `ezcnf`.

Arguments

none

Procedure

A dummy routine as a place holder. It is called for each frame advance. The user may substitute it with special task routine such as display the graphic contents, write a log entry etc.

3.16 ezcdobox

Calling Sequence

```
subroutine ezcdobox(boxid)
```

Description

Defines the Boxid for the LC UNICOS user to receive output.

Arguments

`boxid` character*(3), three character string to identify the boxid, first character is an alphabet in a-z,A-Z, and the last two characters are two alphanumeric characters.

Procedure

The routine checks the legality of the input string and write it to a common block holding this value. ezccgm will grab this string value from the common block to issue proper `lpr` command to the operating system.

3.17 ezcdogk

Calling Sequence

```
subroutine ezcdogk(gkstring)
```

Description

Defines the GIVE/KEEP flag for the LC UNICOS user to allow disposal of the CGM files after fiche output is generated.

Arguments

`gkstring` character*(*), character string to set the GIVE/KEEP flag to “give” “givekeep” or “keep”

Procedure

The routine verifies the input string and write it to a common block holding this value. ezccgm will grab this string value from the common block to issue proper lpr command to the operating system. The default value is “keep”.

3.18 ezcdolev

Calling Sequence

```
subroutine ezcdolev(lvstring)
```

Description

Defines the Security Level for the output for LC UNICOS users.

Arguments

lvstring character*(6), character string to identify the security level, they are “uncl”/ “UNCL”, “pard”/“PARD”, “crd”/“CRD”, “srd”/“SRD” or “1”, “2”, “4”, “5” correspondingly.

Procedure

The routine verifies the input string associates with its boxid (for classified output to a proper box) and write it to a common block holding this value. ezccgm will grab this string value from the common block to issue proper `lpr` command to the operating system.

3.19 ezcerror

Calling Sequence

```
subroutine ezcerror(msg,sevlev)
```

Description

The routine records the error message to STDERR and calls ezcdie to send signal if fatal error occurs.

Arguments

msg character*(120), error message

sevlev integer, error severity level, level 1= comment, level 2= minor abnormality, level 3= fatal error

Procedure

Copy the error message to the STDERR file and call ezcdie if sevlev = 3. You may work to replace ezcdie or supply a signal handler for the signal raised by it. See ezcdie.

3.20 ezcfradv

Calling Sequence

```
subroutine ezcfradv(note)
```

Description

Perform frame advance if the flag `ezcxn`has been set to 1.

Arguments

`note` character*(*), string input to the stub routine `ezchook`.

Procedure

The routine checks the value of `ezcxn` then either do frame advance or just return to the calling program. Before exitting, `ezcfradv` calls `ezchook` with “note” as the second argument. The user can define `ezchook` to perform customized tasks.

3.21 ezcgetcl

Calling Sequence

```
subroutine ezcgetcl(w)
```

Description

This routine returns the index of last non-blank characters in the given string w.

Arguments

`w` character*(*), string needs to determine the index of last non-blank character.

Procedure

Finds the rightmost occurrence of non-blank character.

3.22 ezchook

Calling Sequence

```
subroutine ezchook(msg1,msg2)
```

Description

This is a stub routine to be replaced by a true function routine supplied by the client program when each frame advances.

Arguments

`msg1`, `msg2` character*(*), strings used to pass the arguments to the true function routine.

Procedure

A stub routine.

3.23 ezcnf

Calling Sequence

```
subroutine ezcnf()
```

Description

Makes a frame advance.

Arguments

none

Procedure

The `ezcnf` routine calls `ezcnq` to do frame advance if the external flag `ezcxn` has the value 1. It also clears the flag `ezcxn` and restore the full *reference box* for quadrants as the plotting area.

3.24 ezcnq

Calling Sequence

```
subroutine ezcnq()
```

Description

Does actual call to do frame advance if not in the *quadrant mode*. It also displays the graphic contents by calling `ezcshowf`. (`ezcshowf` in turn calls `ezcdispl` to do the display business. In EZD, the `ezcdispl` routine is a dummy routine as a place holder. The client program of EZD has to provide a true function routine for displaying the graphic contents.)

Arguments

none

Procedure

call `ezcshowf` to display the graphic contents. call `ezcclear` to clear the collection. In EZD `ezcclear` is a stub routine that the client program can replace it for real task. It will stay in the *current quadrant* when in *quadrant mode*.

3.25 ezcps

Calling Sequence

```
subroutine ezcps(iflag,istring)
```

Description

Controls the PS devices. It is one of the underlying action routines for `ezcdodev`.

Arguments

`iflag` character*(*), action-command-string, the possible values are `on`, `off`, `send` or `close`.

`istring` character*(*), command-modifier-string, the possible values are `mono`, `color`.

Procedure

For the action command “on”:

Open and activate a PS device if no existing PS device. Assign a proper file name for the PS file, and open a PS log file if it does not exist. Activate a PS device if it has been deactivated. No action if an active PS file exists.

For the action command “off”:

Deactivate the current active PS device. No action if no active PS device.

For the action command “send”:

Turn the PS device “on” send a frame, then turn the PS device “off” (“send” implies “on” so it may open(i.e. create) a PS file)

For the action command “close”:

Deactivate and then close the PS file.

The command-modifier “color”, “mono” specifies the PS file color. The PS file has default “mono” for its color specification.

3.26 ezcsetbb

Calling Sequence

```
subroutine ezcsetbb()
```

Description

This subroutine sets the background color to black(default).

Arguments

none

Procedure

The routine sets the `bakcol` value to 0, then `ezccidx` based on this value to set background color to black. The client program should call this routine just before opening the device which will have the desired background color. (Currently this feature has been disabled due to the color table problems in the Xwindow driver)

3.27 ezcsetbw

Calling Sequence

```
subroutine ezcsetbw()
```

Description

This subroutine sets the background color to white.

Arguments

none

Procedure

The routine sets the `bakcol` value to 1, then `ezccidx` based on this value to set background color to white. The client program should call this routine just before opening the device which will have the desired background color. (Currently this feature has been disabled due to the color table problems in the Xwindow driver)

3.28 ezcshowf

Calling Sequence

```
subroutine ezcshowf
```

Description

Display the current picture and set a new frame if not in quadrant mode.

Arguments

none

Procedure

The routine checks for the plotting mode first. If it is in quadrant mode then no frame advance, i.e. does not clean the frame so the previous picture on the frame remains. If it is not in quadrant mode, then frame is advanced. It is then plot the graphics to the frame by calling `ezcdispl`. In the EZD, this `ezcdispl` is just a stub routine which the client program should replace it by a true action routine.

3.29 ezcshowg

Calling Sequence

```
subroutine ezcshowg
```

Description

Invoking graphic display routine `ezcshowf`. In EZD, it is a dummy routine as a space holder.

Arguments

none

Procedure

Invokes `ezcshowf` to display the graphics contents. In EZD, it is a dummy routine and should be replaced by a true display action routine or invokes `ezcshowf` to indirectly display the graphic contents. The command “send” depends on this routine to flush a frameful graphics to the designated device. (e.g. `call ezcdodev("cgm","send","color")`)

3.30 ezctek

Calling Sequence

```
subroutine ezctek(iflag,istring)
```

Description

Controls the Tektronix graphic terminal devices.

Arguments

`iflag` character*(*), action-command-string, the possible values are `on`, `off`, `send` or `close`.

`istring` character*(*), command-modifier-string, has not been used by this special device. (Keep the argument just for consistency with other devices.)

Procedure

For the action command “on”:

Open and activate a Tektronix graphic terminal device if no existing Tektronix graphic terminal device. Activate a Tektronix graphic terminal device if it has been deactivated. No action if an active Tektronix graphic terminal exists.

For the action command “off”:

Deactivate the current active Tektronix graphic terminal device. No action if no active Tektronix graphic terminal device.

For the action command “send”:

Turn the Tektronix graphic terminal device “on” send a frame, then turn the Tektronix graphic terminal device “off”

For the action command “close”:

Deactivate and then close the Tektronix graphic terminal.

The command-modifier is not used for the Tektronix graphic terminal.

3.31 ezcwin

Calling Sequence

```
subroutine ezcwin(iflag,istring)
```

Description

Controls the Xwindow devices.

Arguments

`iflag` character*(*), action-command-string, the possible values are `on`, `off`, `send`, or `close`.

`istring` character*(*), command-modifier-string, a window name string

Procedure

The command-modifier istring provides the window name as the way to identify the recipient of the action-command.

For the action command “on”:

Open and activate an Xwindow device with the given window name if no existing Xwindow device with the same name. This will deactivate other active windows in the same display.

Activate an Xwindow device if the named window was created before and has been deactivated. No action if the named Xwindow already active.

For the action command “off”:

Deactivate the named Xwindow device. No action if the named Xwindow is not active. It is an error try to “off” no existing Xwindow. (e.g. no window with the given name hence can not be “off”ed.)

For the action command “send”:

Send a frame to the active Xwindow.

For the action command “close”:

Deactivate and then close the named Xwindow. It is an error to try to “close” a non-existing Xwindow.

If there is only one window, by default it receives the action command. -

INDEX

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X