EDM

Extensible Display Manager

John Sinclair

EDM: Extensible Display Manager
by John Sinclair

Table of Contents

List of Figures

Chapter 1. Overview

Introduction

With the introduction of tools like EPICS and VSystem, the task of building complex distributed control systems has been greatly simplified. Furthermore, extending both the content and functionality of these systems is a trivial task. This is generally not the case for applications that have been built on top of these core components and in most cases this is acceptable. There is one class of applications, namely interactive graphical display tools, where the lack of user extensibility is a severe liability. This deficiency arrests the growth of the suite of common display elements and prevents the development and customization of site specific elements for special requirements. Just as EPICS record and device support is easily extended without in-depth knowledge of the core software, so also should the elements of a display tool be extensible, again without in-depth knowledge of either the core behavior of the display tool or the more esoteric details of window-based graphical user interface systems.

Edm, one example of an EPICS display manager, is an attempt to fulfill this objective. A display manager is a tool that manages a collection of active displays. It provides the ability to create and edit display content (like graphics, text, meters, sliders, buttons, plots, etc.) and uses some facility (.e.g EPICS channel access) to execute the same content resulting in the dynamic presentation of live data.

Other EPICS Display Managers

Additional examples of display managers for EPICS include the following:

- medm
- dm2k
- edd/dm

Display Widgets

The following is a list of some of the more common edm widgets.

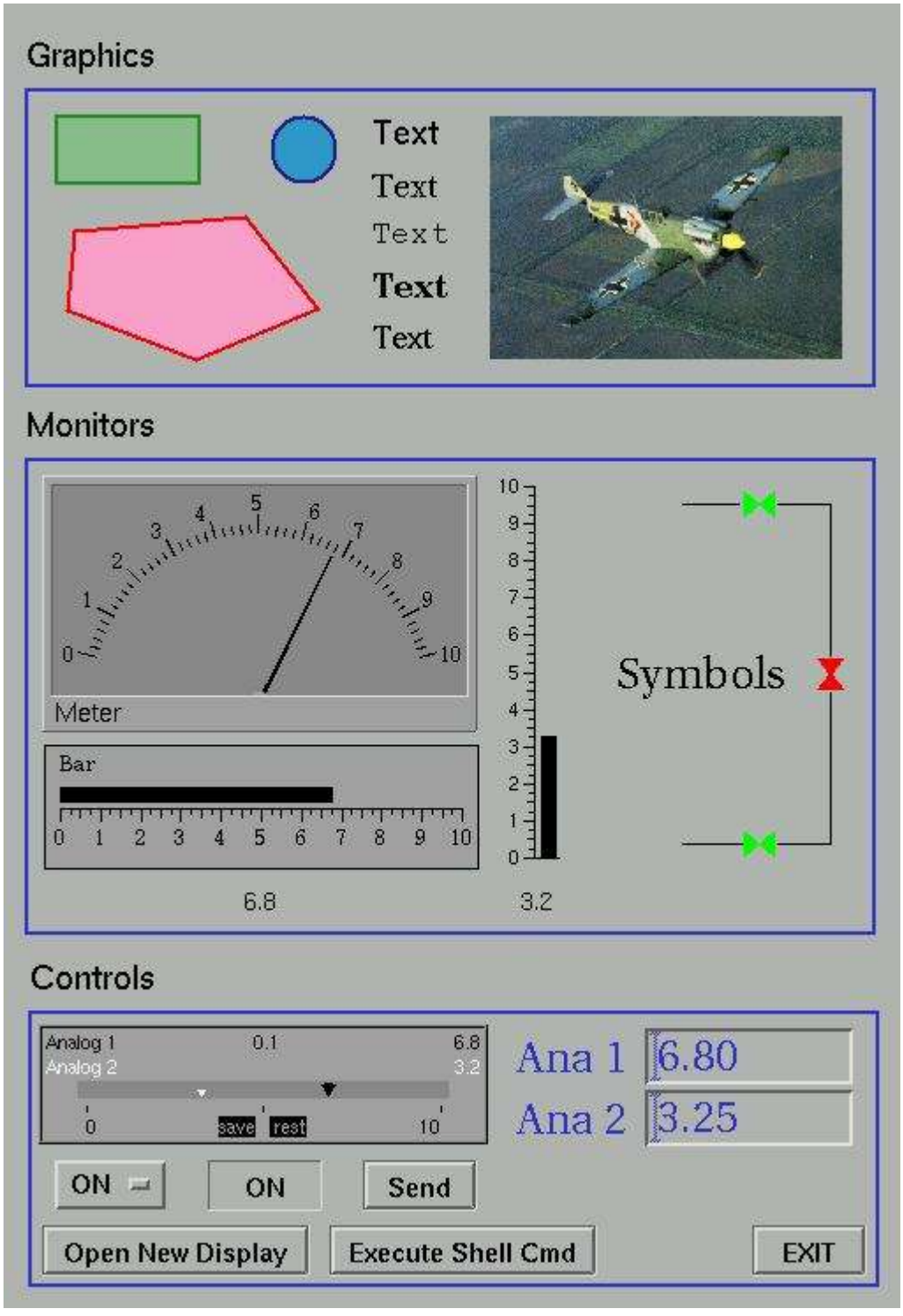
See edm online help topic *Objects* for more information.

- Graphics
  - Lines
  - Rectangle
  - Circle
  - Arc
  - Static Text
  - GIF Image
  - PNG Image
  - HTML

  - Dynamic Symbols (built from graphic elements)
- Monitors
  - Meter
  - Bar
  - Message Box
  - Text
  - Byte
  - X-Y Graph
  - Symbols (built from graphic elements)
- Controls
  - Text
  - Slider
  - Motif Slider (medm-like)
  - Button
  - Menu Button
  - Choice Button
  - Radio Box
  - Message Button
  - Up/Down Button
  - Exit Button
  - Related Display
  - Shell Command
  - Menu Mux (multiplexor)
  - Shell Command
  - Multiplexor

Figure 1-1 is a screenshot showing several display elements provided by the base widject library.

Figure 1-1. Example edm Widgets



Editing Features

Editing functions include the following:

- Display grid/Snap to grid
- Orthogonal line draw/Orthogonal move
- Copy/Cut/Paste (across different displays)
- Raise/Lower
- Group/Ungroup
- Rotate 90/Flip
- Select group PV name edit
- Select group display info edit (colors/fonts)
- Align select group
- Center select group
- Uniformly size select group
- Distribute select group

Chapter 2. Software Installation

System Requirements

Motif is required to build edm. OpenMotif has been used on Linux platforms successfully. LessTif is not currently recommended.

Image widgets require the following external libraries:

- GIF Support: libgif
- PNG Support: libpng

Both libraries are publically available and can be readily found from most web search engines sites.

Installation Procedure

The edm distribution is packaged in a tar file with a name like edm-1-9-1f.tgz. In this case the name corresponds to major version 1, minor version 9, and release 1f.

Place the tar file in the src directory under the EPICS extensions directory. Use gunzip to uncompress the file and then use tar to unpack the archive. Edm should then build like any other EPICS extension.

Unlike most other EPICS extensions, edm requires additional configuration after installation. See chapter 3 before attempting to execute edm.

Example:

1. gunzip edm-1-9-1f.tgz
2. tar -xvf edm-1-9-1f.tar
3. cd edm
4. make

Chapter 3. System Configuration

The following items need to be configured after installation.

- Contents and location of component files
- Content and location of color file
- Content and location of font file
- Contents of print definition file

Location and contents of component files

edmObjects

Edm uses a registry that maps widgets to a class name and shareable library path. The registry is contained in a file named edmObjects which resides in /etc/edm or the directory pointed to by the environment variable EDMOBJECTS.

When edm is installed, an example edmObjects file is included as part of the installation but this file should not be used in the production edm environment. Instead, a new file should be created by edm itself. This is accomplished as follows:

- Choose a location for the edmObjects file, make EDMOBJECTS point to this location, and make sure you have write access.
- Locate each edm component library. After the build, these reside in .../extensions/lib/(ARCH DEPENDENT)/ and, except for libEdmBase.so, they all have uuid-like names (however, not all files so named are component files).

  Example names:

  libEdmBase.so

  lib7e1b4e6f-239d-4650-95e6-a040d41ba633.so

  You may verify that a library is an edm component file as the following example illustrates:

  edm -show /usr/local/src/epics/supTop/R3.13.6/extensions/lib/Linux/libEdmBase.so
- For each component library file, execute edm with the -add option as follows:

  edm -add (full absolute path to library)/(library name)

  e.g. edm -add /usr/local/src/epics/supTop/R3.13.6/extensions/lib/Linux/libEdmBase.so

After populating edmObjects in this manner, you may wish to manually manage it with a text editor. If so, the number of objects at the top must match the number of registry entries. The directory location references may be replaced with the environment variable EDMLIBS. The following shows two equivalent entries (assuming, of course, that EDMLIBS has been set to /epics/extensions/lib/linux-x86).

- activeLineClass /epics/extensions/lib/linux-x86/libEdmBase.so Graphics Lines
- activeLineClass $(EDMLIBS)/libEdmBase.so Graphics Lines

The use of the EDMLIBS variable allows different versions of edm to reference the same edmObjects

file.

edmPvObjects

A similar exercise needs to be performed for a PV component registry file. In this case the environment variable is EDMPVOBJECTS, the file is edmPvObjects, and the only library is libEpics.so.

Do the following:

- Choose a location for the edmPvObjects file, make EDMPVOBJECTS point to this location, and make sure you have write access.
- Locate the edm pv component library. After the build, this file reside in .../extensions/lib/(ARCH DEPENDENT)/ and is named libEpics.so.

  You may verify that a library is an edm pv component file as the following example illustrates:

  edm -showpv /usr/local/src/epics/supTop/R3.13.6/extensions/lib/Linux/libEpics.so
- Add the pv component library to the registry file as follows:

  edm -addpv (full absolute path to library)/(library name)

  e.g. edm -addpv /usr/local/src/epics/supTop/R3.13.6/extensions/lib/Linux/libEpics.so

Color File Configuration

The edm color file must be manually created using a text editor. The file name is normally *colors.list* and the default location is */etc/edm*. The location may be overridden on a site-by-site or project-by-project basis with the environment variable *EDMFILES*. The name and location may be overridden on a site-by-site or project-by-project basis with the environment variable *EDMCOLORFILE*.

The first line of the color file must contain the color file format version id as three numbers corresponding to major, minor, and release. This should currently be

```
4 0 0
```

The next three items in the color file are described as follows:

- The blink period given in milliseconds, e.g.

  `blinkms=750`
- The number of columns in the color palette, e.g.

  `columns=5`
- The maximum number of RGB component values (256 or 65536), e.g.

  `max=0x10000`

  This is one more than the largest value that may be specified for any RGB component.

So, the first few lines of the colors.list file might look like this

```
4 0 0

blinkms=750
max=0x10000
```

```
columns=5
```

Static Colors

Edm stores and references colors as indices rather than RGB values. Colors can be selected visually or functionally, i.e. from a palette of colors or from a menu of color names. The edm color file establishes this mapping of index to RGB and functional name. In this manner, a list of static colors are specified.

An example static color declaration is shown below:

```
static 25  Controller    { 0 0 65535 }    # blue

# blinking is specified by 2 RGB values
static 26  "blinking red"  { 65535 0 0 41120 0 0 }
```

In this example, note the following:

- The first color's index is 25 and functional name is *Controller*.
- The second color's index is 26 and functional name is *blinking red*.
- The name *Controller* could have been enclosed in quotes but this is not necessary.
- The name *blinking red* must be enclosed in quotes and, in general, this is true for any color name containing white space.
- Comments are introduced by the # character and may be placed anywhere but on the very first line of the color file.

Dyanmic Colors

In addition, another list of dynamic colors may be included where the mapping is index to color rule and name. A color rule is an expression that maps a color name to a set of RGB values depending on the range of an associated color PV. A color rule might look like the following:

```
rule 100 exampleRule {
 =100 || =200 : strange
 >=20         : invisible
 >0 && <10    : red
 >=10 && <20  : "blinking red"
 default      : green
}
```

In this example, note the following:

- The color index is 100 and the functional name is exampleRule.
- *invisible* is a color name with special behavior that causes some widgets to disappear (the widget must support invisibility by color). Even though the color name *invisible* is special, it must nevertheless be defined as a static color. The specified RGB value is utilized by widgets that do not support invisibility by color.
- The names *strange*, *invisible*, *red*, *blinking red*, and *green* must be valid color names defined elsewhere in the color file
- Rule evaluation goes from top to bottom, stopping at the first true condition. The default clause must

therefore come last.

- Symbols
  - `=  equal to`
  - `>  greater than`
  - `>=  greater than or equal to`
  - `<  less than`
  - `<=  less than or equal to`
  - `||  logical or`
  - `&&  logical and`

The Color Palette and Functional Name Menu

The color palette shown in Figure 3-1 and color name menu in Figure 3-2 corresponds to the following color file information:

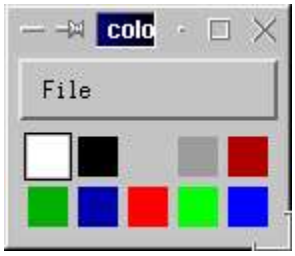
```
4 0 0

blinkms=750
max=0x10000
columns=5

static 6   white        { 0xffff 0xffff 0xffff }
static 7   black        { 0 0 0 }
static 8   grey1        { 50000 50000 50000 }
static 9   grey2        { 40000 40000 40000 }
static 3   "dark red"   { 0xffff 0 0 }
static 4   "dark green" { 0 0xffff 0 }
static 5   "dark blue"  { 0 0 0xffff }
static 0   red          { 0xffff 0 0 }
static 1   green        { 0 0xffff 0 }
static 2   blue         { 0 0 0xffff }

menumap {
red
green
blue
"dark red"
"dark green"
"dark blue"
}
```

Note that there are 5 columns as given by

```
columns=5
```

and that the palette is populated from top-left to bottom-right, row by row, according to the entry order of color indices in the color file. So, *white* is the first specified color in the file and also the first color in the palette. Likewise, *blue* is last in both places.

Figure 3-1. Example edm Color Palette



The listing order in the color name menu is explicitly given in the *menumap* section of the color file. This is clear by comparing the menumap content to the menu shown in Figure 3-2. Note that some colors have been intentionally omitted from the menu.

Figure 3-2. Example edm Color Name Menu

Color Aliases

Some widgets, like the X-Y plot, require many default colors. Instead of referencing color names or indices (which may not exist at a given site), they instead reference color aliases. The X-Y plot uses the following color alias names: *trace0*, *trace1*, ... *trace9*.

These color alias names are included in the color file to establish this mapping for a given configuration. An example is shown below.

```
alias trace0 red
alias trace1 green
alias trace2 blue
alias trace3 "dark red"
alias trace4 "dark green"
alias trace5 "dark blue"
alias trace6 black
alias trace7 white
alias trace8 grey1
alias trace9 grey2
```

Alarm Colors

The *alarm* section of the color file maps color names into EPICS alarm states. An example is given be-

low.

```
alarm {
  disconnected : grey1
  invalid      : grey2
  minor        : blue
  major        : red
  noalarm      : green
}
```

In this case, all alarm sensitive content in the *noalarm* state is shown as green.

The following causes alarm sensitive content in the *noalarm* state to be rendered with user specified colors:

```
alarm {
  disconnected : grey1
  invalid      : grey2
  minor        : blue
  major        : red
  noalarm      : *
}
```

Example Color File

The following is an example of an entire edm color file.

```
4 0 0

blinkms=750
max=0x10000
columns=5

alias trace0 red
alias trace1 green
alias trace2 blue
alias trace3 "dark red"
alias trace4 "dark green"
alias trace5 "dark blue"
alias trace6 black
alias trace7 white
alias trace8 grey1
alias trace9 grey2

static 6   white        { 0xffff 0xffff 0xffff }
static 7   black        { 0 0 0 }
static 8   grey1        { 50000 50000 50000 }
static 9   grey2        { 40000 40000 40000 }
static 3   "dark red"   { 0xafff 0 0 }
static 4   "dark green" { 0 0xafff 0 }
static 5   "dark blue"  { 0 0 0xafff }
static 0   red          { 0xffff 0 0 }
static 1   green        { 0 0xffff 0 }
```

```
static 2   blue         { 0 0 0xffff }
static 10  "blink red"  { 0xffff 0 0 0x3fff 0 0 }
static 11  invisible    { 0 0 0 }

rule 12 rule1 {
 =100 || =200 : grey1
 >=20         : invisible
 >0 && <10    : red
 >=10 && <20  : "blink red"
 default      : green
}

menumap {
red
green
blue
"dark red"
"dark green"
"dark blue"
rule1
}

alarm {
  disconnected : grey1
  invalid      : grey2
  minor        : blue
  major        : red
  noalarm      : *
}
```

Font File Configuration

The edm font file must be manually created using a text editor. The file name is always *fonts.list* and the default location is */etc/edm*. The location may be overridden on a site-by-site or project-by-project basis with the environment variable *EDMFILES*. The name and location may be overridden on a site-by-site or project-by-project basis with the environment variable *EDMFONTFILE*.

All edm fonts must be explicitly specified in the font file.

Version 3 Font File Format

The version 3 font file format accomodates blank lines and comment lines. Comment lines must contain '#' as the first non white-space character.

The first line of the font file must contain the file version id as three numbers corresponding to major, minor, and release. Currently, this should be

```
3 0 0
```

The next two lines in the font file are default font tags that are used to resolve unknown font references. The lines look something like the following:

```
helvetica-medium-r-10.0
helvetica-medium-r-12.0
```

The first tag, in this example helvetica-medium-r-10.0, is used as a substitute for an invalid font referenced in a display file.

The second tag, in this example helvetica-medium-r-18.0, is used as a default font for creating new text when no default has been specified. Specifying a default font is described elsewhere.

The remaining lines specify available fonts. The information is the same as for previous file versions but the form is highly condensed and the font name may now be arbitrarily assigned (previously, the font family mapped directly to the font name).

Note that for all font file versions, blanks in font specification lines are significant and may not be used as separators.

Using the previous file formats, the following lines would specify 8, 10, and 12 point courier fonts. Notice the four groups of three sizes.

```
-*-courier-bold-o-normal--8-80-75-75-m-50-*-*
-*-courier-bold-o-normal--10-100-75-75-m-60-*-*
-*-courier-bold-o-normal--12-120-75-75-m-70-*-*
-*-courier-bold-r-normal--8-80-75-75-m-50-*-*
-*-courier-bold-r-normal--10-100-75-75-m-60-*-*
-*-courier-bold-r-normal--12-120-75-75-m-70-*-*
-*-courier-medium-o-normal--8-80-75-75-m-50-*-*
-*-courier-medium-o-normal--10-100-75-75-m-60-*-*
-*-courier-medium-o-normal--12-120-75-75-m-70-*-*
-*-courier-medium-r-normal--8-80-75-75-m-50-*-*
-*-courier-medium-r-normal--10-100-75-75-m-60-*-*
-*-courier-medium-r-normal--12-120-75-75-m-70-*-*
```

The version 3 file format specifies all this information on one line and the font name is idependent of the font specification. The equivalent version 3 content is shown below.

```
courier=-*-courier-(medium,bold)-(r,o)-normal--*-(80,100,120)-75-75-m-*-*-*
```

Note that the *weight* option must be listed in the order (<non-bold>,<bold>). Likewise the *slant* option must be ordered as (<non-italicized>,<italicized>).

For example, the *gothic* font would be specified as follows:

```
gothic=-urw-gothic l-(book,demi bold)-(r,o)-normal--0-(80,100,120)-75-75-p-0-
*-*
```

In this case, the weight modifiers are book (non-bold) and demi bold (bold). The slant modifiers are r (non-italicized) and o (oblique, i.e. italicized).

Appending at least one tab and the keyword *preload* to the font spec line instructs edm to preload the font during initialization.

```
courier=-*-courier-(medium,bold)-(r,o)-normal--*-(80,100,120)-75-75-m-*-*-*
 preload
```

Another keyword that may be specified is *exact*. This option requires that all fonts match the font speci-

fication exactly. The default policy is to allow another font to be used if the desired font cannot be resolved. As before, use a tab character as whitespace to separate keywords from one another and from the font spec.

```
courier=-*-courier-(medium,bold)-(r,o)-normal--*-(80,100,120)-75-75-m-*-*-*
 preload    exact
```

The above example, where the font name is the same as the family name, makes the content compatable with previous file versions. The same font could be renamed as given below.

```
edm courier font=-*-courier-(medium,bold)-(r,o)-normal--*-(80,100,120)-75-75-
m-*-*-*
```

Of course, the actual intent is to change the font spec and keep the name unchanged. For example, suppose the free Microsoft *courier new* scalable font is installed. An existing font given as

```
courier=-*-courier-(medium,bold)-(r,o)-normal--*-(80,100,120)-75-75-m-*-*-*
```

could be changed to

```
courier=-microsoft-courier new-(medium,bold)-(r,o)-normal--0-(80,100,120)-75-
75-m-0-*-*
```

All existing edm screens would now use the new font.

Font Groups

Version 3 files support the notion of font groups where each group contains one font name and multiple fonts specs. An example is shown below.

```
helvetica={
-urw-helvetica-(medium,bold)-(r,o)-normal--0-(80,100,120)-75-75-m-0-*-*    pr
eload
-adobe-helvetica-(medium,bold)-(r,o)-normal--*-(80,100,120)-75-75-m-*-*-*
preload
-*-helvetica-(medium,bold)-(r,o)-normal--*-(80,100,120)-75-75-m-*-*-*    prel
oad
}
```

In this example, edm will first try to use the scalable urw font found in most Linux distributions. If an exact match cannot be made, the adobe font will be tried next. Finally any helvetica font will be tried.

If all font resolution attempts fail, and if the last font spec of the group does not include the *exact* option, it will be used to generate a fallback set of fonts, some of which may be members of a different font family. If even this strategy fails, edm will abort.

Version 2 Font File Format

Version 2 is equivalent to version 1 with the following exceptions:

- The first line is 2 0 0.
- One or more tabs and the keyword *preload* may be appended to a font specification. This instructs edm to preload the font during initialization.

Version 1 Font File Format

The version 1 font file format is somewhat inflexible. Neither blank lines nor comments of any kind are allowed. The first line of the font file must contain the file version id as three numbers corresponding to major, minor, and release. Currently, this should be

```
1 1 0
```

The next two lines in the font file are default font tags that are used in case a font referenced in a display file has not been specified in the font file. This might be the case if a file has been obtained from some external site. The lines look something like the following:

```
helvetica-medium-r-10.0
helvetica-medium-r-12.0
```

The first tag, in this example helvetica-medium-r-10.0, is used as a substitute for an invalid font referenced in a display file.

The second tag, in this example helvetica-medium-r-18.0, is used as a default font for creating new text when no default has been specified. Specifying a default font is described elsewhere.

The remaining lines specify fonts, family by family, four groups of font specifications on separate lines, in repeated blocks for each of the equivalent of bold-italic, bold-normal, medium-italic, and medium-normal.

So, for example, the following lines would specify 8, 10, and 12 point courier fonts. Notice the four groups of three sizes.

```
-*-courier-bold-o-normal--8-80-75-75-m-50-*-*
-*-courier-bold-o-normal--10-100-75-75-m-60-*-*
-*-courier-bold-o-normal--12-120-75-75-m-70-*-*
-*-courier-bold-r-normal--8-80-75-75-m-50-*-*
-*-courier-bold-r-normal--10-100-75-75-m-60-*-*
-*-courier-bold-r-normal--12-120-75-75-m-70-*-*
-*-courier-medium-o-normal--8-80-75-75-m-50-*-*
-*-courier-medium-o-normal--10-100-75-75-m-60-*-*
-*-courier-medium-o-normal--12-120-75-75-m-70-*-*
-*-courier-medium-r-normal--8-80-75-75-m-50-*-*
-*-courier-medium-r-normal--10-100-75-75-m-60-*-*
-*-courier-medium-r-normal--12-120-75-75-m-70-*-*
```

If desired, scalable fonts may be specified as follows.

```
-*-courier-bold-r-normal--0-90-75-75-m-0-*-*
```

Scalable Fonts

In general, it is not possible to guarantee that high quality scalable fonts will be used when appropriate. But if the actual family name of the desired scalable font is used in the spec, it obviously will be. Figure 3-3 shows the rendering on a linux system of a 72 point New Century Schoolbook font specified as

```
-*-new century schoolbook-bold-r-normal--0-720-75-75-p-0-*-*
```

Clearly, the font renders, but the quality is poor.

Figure 3-3. Poor Quality Font

In Figure 3-4, when the font is specified as

```
-urw-new century schoolbook-bold-r-normal--0-720-75-75-p-0-*-*
```

The nice scalable font is used and the results are much more pleasing.

Figure 3-4. High Quality Font

Example Font File

The following is an example of an entire edm font file. This would make 8, 10, 12 and 72 point courier and helvetica fonts available.

```
1 1 0
helvetica-medium-r-10.0
helvetica-medium-r-12.0
-*-courier-bold-o-normal--8-80-75-75-m-50-*-*
-*-courier-bold-o-normal--10-100-75-75-m-60-*-*
-*-courier-bold-o-normal--12-120-75-75-m-70-*-*
-*-courier-bold-o-normal--0-720-75-75-m-0-*-*
-*-courier-bold-r-normal--8-80-75-75-m-50-*-*
-*-courier-bold-r-normal--10-100-75-75-m-60-*-*
-*-courier-bold-r-normal--12-120-75-75-m-70-*-*
-*-courier-bold-r-normal--0-720-75-75-m-0-*-*
-*-courier-medium-o-normal--8-80-75-75-m-50-*-*
-*-courier-medium-o-normal--10-100-75-75-m-60-*-*
-*-courier-medium-o-normal--12-120-75-75-m-70-*-*
-*-courier-medium-o-normal--0-720-75-75-m-0-*-*
-*-courier-medium-r-normal--8-80-75-75-m-50-*-*
-*-courier-medium-r-normal--10-100-75-75-m-60-*-*
-*-courier-medium-r-normal--12-120-75-75-m-70-*-*
-*-courier-medium-r-normal--0-720-75-75-m-0-*-*
-*-helvetica-bold-o-normal--8-80-75-75-p-50-*-*
-*-helvetica-bold-o-normal--10-100-75-75-p-60-*-*
-*-helvetica-bold-o-normal--12-120-75-75-p-69-*-*
-*-helvetica-bold-o-normal--0-720-75-75-p-0-*-*
-*-helvetica-bold-r-normal--10-100-75-75-p-60-*-*
-*-helvetica-bold-r-normal--12-120-75-75-p-70-*-*
-*-helvetica-bold-r-normal--0-720-75-75-p-0-*-*
```

```
-*-helvetica-bold-r-normal--8-80-75-75-p-50-*-*
-*-helvetica-medium-o-normal--10-100-75-75-p-57-*-*
-*-helvetica-medium-o-normal--12-120-75-75-p-67-*-*
-*-helvetica-medium-o-normal--0-720-75-75-p-0-*-*
-*-helvetica-medium-o-normal--8-80-75-75-p-47-*-*
-*-helvetica-medium-r-normal--10-100-75-75-p-56-*-*
-*-helvetica-medium-r-normal--12-120-75-75-p-67-*-*
-*-helvetica-medium-r-normal--0-720-75-75-p-0-*-*
-*-helvetica-medium-r-normal--8-80-75-75-p-46-*-*
```

Note that the fonts do not have to specified in size order (see ...helvetica-bold-r-normal--8... above) but all variants must be included for each size (bold-italic, bold-normal, medium-italic, medium-normal).

Printing

Printing in edm depends on external utilities like those found in the netpbm distribution (formerly libgr-progs). A print definition file is used to specify template print commands and to relate print options to print dialog box fields.

The following command might be used from a Linux shell to print a window:

- xwd | xwdtopnm | pnmdepth 255 | pnmtops -center | lpr -P lp

The above command could be used in the edm print definition file verbatim. The corresponding directive follows.

- printCommand = "xwd | xwdtopnm | pnmdepth 255 | pnmtops -center | lpr -P lp"

The problem with the above example is the lack of options like page size, portrait/landscape, a list of print queues, and so on. Furthermore, the xwd command shown above requires user interaction to select the window to be dumped. The edm print definition file allows options to be specified and related to fields in a print dialog box.

The Print Definition File

The following rules are used to locate the edm print defintion file:

- If the environment variable EDMPRINTDEF is defined, its value is taken as the full name, including path, of the print defintion file.
- If EDMPRINTDEF is unset and EDMFILES is defined, then the file edmPrintDef in the directory given by EDMFILES is used.
- If EDMPRINTDEF and EDMFILES are both unset, the file /etc/edm/edmPrintDef is used.

If the print defintion file does not exist or is invalid, printing is disabled and the print menu item is inactive.

Directives

The following directives may appear in any order. Blank lines and comment lines starting with # are ignored.

- printDialog (optional) - gives the maximum width and height of the dialog box

- printCommand (required) - gives command used to print to a printer
- printToFileCommand (required) - gives command used to print to a file
- optionDefaults (optional) - specifies print command option defaults
- option <number> (optional) - specifies a print dialog field and command option modifier

printDialog

The printDialog directive has a simple form as shown below:

- printDialog { w = 400 h = 500 }

The line "w = 400" sets the maximum dialog box width to 400 pixels and "h = 500" sets the maximum height to 500 pixels.

printCommand

The printCommand directive includes legal shell commands and parameters. It is used when *Print To File* is not selected. An example follows.

- printCommand = "xwd -display <DSPNAME> -id <WINID> | xwdtopnm | pnmdepth 255 | pnmtops <opt1> | lpr <opt2>"

  <WINID> translates to the XWindow ID of the window from which the print menu item was selected.

  <DSPNAME> translates to the current X Window display name

  <optN> translates to the value of the Nth option where N ranges from 1 to 10.

printToFileCommand

The printToFileCommand directive includes legal shell commands and parameters. It is used when *Print To File* is selected. An example follows.

- printToFileCommand = "xwd -display <DSPNAME> -id <WINID> | xwdtopnm | pnmdepth 255 | pnmtops <opt1> > <file>"

  <WINID> translates to the XWindow ID of the window from which the print menu item was selected.

  <DSPNAME> translates to the current X Window display name

  <optN> translates to the value of the Nth option where N ranges from 1 to 10.

  <file> translates to the filename specified in the print dialog box by the user.

optionDefaults

Options are stored as strings. The directive optionDefaults gives the initial value of the option strings.

- optionDefaults { opt1 = "" opt2 = "" }

option

The option directive has the following general form:

```
option N = <DIALOG FIELD TYPE> {
  label = "<LABEL STRING>"

  default = <DEFAULT DIALOG FIELD VALUE>

  <OPTION STRING MODIFIERS>

}
```

The option attributs (label, default, modifiers) must be given in the order shown in the above example.

If <*DIALOG FIELD TYPE*> is menu, a string of menu items follow as shown below:

```
option N = menu "item1|item2|item3..." {
  .
  .
  .
```

Dialog Field Type = menu

For the *menu* type, default is given as a number from 0 to n-1, where n is the number of items in the menu string.

Option string modifiers have the following form:

```
  N option [ = | += ] "<OPTION STRING VALUE>"

  e.g. to replace current option with "-noturn"
  0 option = "-noturn"

  to append "-noturn" to the current option
  0 option += "-noturn"
```

A complete example follows.

```
option 2 orientation = menu "Default|Portrait|Landscape" {

  label = "Orientation"

  default = 0

  0 option += ""

  1 option += "-noturn"

  2 option += "-turn"

}
```

Dialog Field Type = toggle

The *toggle* type is much like the *menu* type. As before, default is given as a number, but for the *toggle* type, the number must be 0 or 1.

Option string modifiers have the following form:

```
  [ 0 | 1 ] option [ = | += ] "<OPTION STRING VALUE>"

  e.g. to replace current option with "-center"
  0 option = "-center"

  to append "-center" to the current option
  0 option += "-center"
```

A complete example follows.

```
option 2 center = toggle {

  label = "Center"

  default = 1

  0 option += "-nocenter"

  1 option += "-center"

}
```

Dialog Field Type = text

For the *text* type, default is given as a string.

In this case the option is modified by the *option string modifier* concatenated with the text entry value specified by the user. If a space should separated the two, it must be included at the end of the *option string modifier*.

The single option string modifier has the following form:

```
  option [ = | += ] "<OPTION STRING VALUE>"

  e.g. to replace current option with "-width " (and the user input)
  option = "-width "

  to append "-width " (and the user input) to the current option
  option += "-width "
```

A complete example follows.

```
option 2 pagewidth = text {

  label = "Page Width"

  default = "8.5"

  option += "-width "

}
```

Example Print Definition File

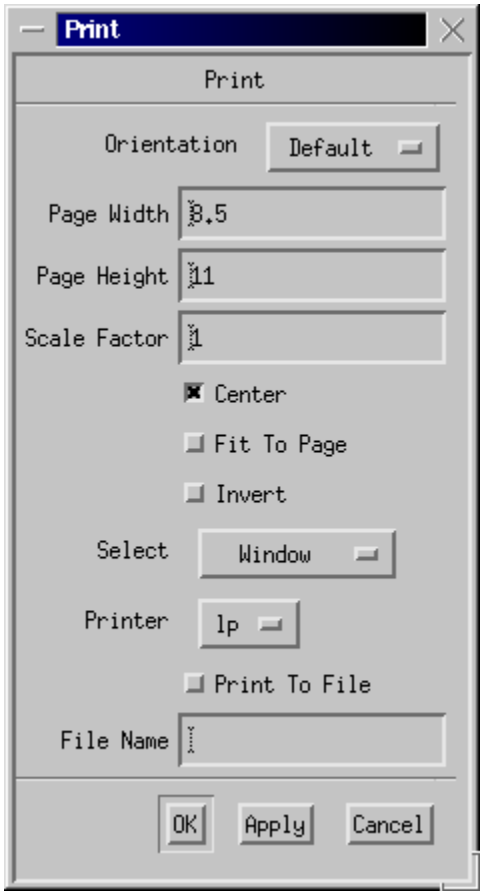
```
Note:

  If the environment variable EDMDEBUGMODE is set to 1,
  the print command will be displayed when a print
  operation is performed.
```

A complete example edm print definition file is shown below. Figure Figure 3-5 shows the corresponding print dialog box.

```
printDialog {
  w = 400
  h = 600
}

printCommand = "xwd <opt1> | xwdtopnm | pnmdepth 255 <opt4> | pnmtops <opt2>
<opt3> | lpr <opt5>"

printToFileCommand = "xwd <opt1> | xwdtopnm | pnmdepth 255 <opt4> | pnmtops
<opt2> <opt3> > <file>"

optionDefaults {
  opt1 = "-display <DSPNAME> "
  opt2 = ""
  opt3 = ""
  opt4 = ""
  opt5 = ""
}

option 2 orientation = menu "Default|Portrait|Landscape" {
  label = "Orientation"
  default = 0
  0 option += ""
  1 option += "-noturn"
  2 option += "-turn"
}

option 2 pagewidth = text {
  label = "Page Width"
  default = "8.5"
  option += "-width "
}

option 2 pageheight = text {
  label = "Page Height"
  default = "11"
  option += "-height "
}

option 3 scale = text {
  label = "Scale Factor"
```

```
  default = "1"
  option += "-scale "
}

option 3 center = toggle {
  label = "Center"
  default = 1
  0 option += "-nocenter"
  1 option += "-center"
}

option 3 fit = toggle {
  label = "Fit To Page"
  default = 0
  0 option += ""
  1 option = "-center -scale 1000"
}

option 4 invert = toggle {
  label = "Invert"
  default = 0
  0 option += ""
  1 option += "| pnminvert"
}

option 1 select = menu "Window|Desktop|Interactive" {
  label = "Select"
  default = 0
  0 option += "-silent -id <WINID>"
  1 option += "-silent -root"
  2 option += "-frame"
}

option 5 printer = menu "lp|lp0" {
  label = "Printer"
  default = 0
  0 option = "-P lp"
  1 option = "-P lp0"
}
```

Figure 3-5. Example Print Dialog Box



Chapter 4. Program Execution, Command Line Options, and Environment

Execution

```
edm [-x] [-noedit] [-ro] [-noscrl] [-m <"m1=string1,m2=string2,...">]
    [-display <display name>] [-ctl <control PV name>]
    [-color rgb or index] [-cmap] [-restart] [-server] [-one] [-open]
    [-port <port #>] [-local] [-eolc] [-(v|V|version)] [-(h|help)]
    [<display-file1 display-file2 ...>]

    -x       Open all displays in execute rather than edit mode

    -noedit  Remove capability to put display in edit mode, used
             with -x to produce execute only operation

    -ro      Read-only mode (when the -server option is in use,
             this option affects the primary server only;
             clients inherit this attribute from the server)

    -noscrl  Disable scroll bars

    -m       Introduces list of macros/expansions

               e.g. -m "facility=beamline-5,section=z"

    -display Takes name of X Display

    -ctl     Takes name of string process variable, writing
             a display file name to this string causes edm
             to open the display in execute mode

             Writing "* SHUTDOWN *" to the pv invokes a remote
             shutdown; edm writes its pid back into the same pv

    -color   Set Colormode - index (default) or rgb

    -cmap    Use private colormap if necessary

    -restart Takes pid number, restart from last shutdown

    -server  Communicate with or become a
             display file server which can
             manage multiple displays

    -one     Like -server but allows only one instance of edm
             per display; all instances must be launched with
             this option and all other options except -display
```

```
                should be identical

    -open    Request server to open specified files; macros may
             be set; a server must exist and must already be
             managing the corresponding display otherwise the
             request will be ignored

    -port #  Use specified TCP/IP port number (default=19000);
             may be used with -server and -one options

    -local   Do not communicate with the display
             file server (default)

    -eolc    Exit on last close; edm exits when last screen is closed

    -version Displays edm version information and exits
    -v
    -V

    -help    Displays this text and exits
    -h
    -?

COMPONENT MANAGEMENT

edm (-add|-remove|-show) <component library file pathname>

    -add     Add all library components to edmObjects

    -remove  Remove all library components from edmObjects

    -show    Show all components in specified library

               e.g. edm -add /usr/lib/someEdmComponents.so

PV COMPONENT MANAGEMENT

edm (-addpv|-removepv|-showpv) <PV component library file pathname>

    -addpv     Add all library components to edmPvObjects

    -removepv  Remove all library components from edmPvObjects

    -showpv    Show all components in specified library

               e.g. edm -addpv /usr/lib/someEdmPvComponents.so
```

Environment

```
ENVIRONMENT VARIABLES

EDMFILES               Location of fonts.list and colors.list
```

```
                        (if unset then /etc/edm) if EDMCOLORFILE
                        and EDMFONTFILE are unset

EDMCOLORFILE            Name and location of color file

EDMFONTFILE             Name and location of font file

EDMHELPFILES            Location of edm help displays
                        (if unset then /etc/edm)

EDMOBJECTS              Location of edmObjects, the edm component
                        object file (if unset then /etc/edm)

EDMPVOBJECTS            Location of edmPvObjects, the edm PV component
                        object file
                        (if unset then /etc/edm)

EDMDATAFILES            Location of display files given as

                          <path 1>:<path 2>:...:<path n>
                          e.g. /usr/displays:/usr/more:/usr/yetMore

                          If EDMDATAFILES is not set then default
                          path is ./

EDMSERVERS              Comma separated list of server names

                        <server1>[:merit],<server2>[:merit]...
                        e.g. node1:2,node2:1.5,node3
                        (node1 is preffered 2/1.5, 2/1 over nodes 2 & 3)

EDMTMPFILES             Location of autosave files (default: /tmp)

EDMPRINTDEF             Name of print defintion file; if unset the file
                        edmPrintDef in the directory given by EDMFILES
                        is used. If EDMFILES is unset, the file
                        /etc/edm/edmPrintDef is used.

EDMUSERLIB              Location of user shared library file (future)

EDMCOMMENTS             Path to comment file (see chapter on EDM and CVS).


EDMHTTPDOCROOT          (version 1-10-x only) URL list separated by "|".
```

If this environment variable is set, file access is performed using libcurl and edm must be compiled with the -DUSECURL=1 option. This results in a list of document root locations, each of which are searched to resolve a file reference. All directories given in EDMFILES and EDMUSERFILES are relative to the document root locations.

This gives a locally executed instance of edm read-only access to display files on a web server. Color files, font files, and png images are also obtained from the remote server. Currently, gif files may not be remotely accessed (this will change in a future release).

Special Purpose Environment Variables

The following environment variables are not intended for use during normal operation of edm.

```
EDMCOLORMODE            Set to rgb or RGB to enable rgb color mode
```

Help files employ rgb color mode to make display file colors somewhat site independent. When an edl file is written in rbg mode, colors are stored as raw R-G-B values instead of as the usual color indices. When a file written in this mode is opened, colors are chosen from the palette that best match the stored R-G-B values.

```
EDMSUPERVISORMODE       Set to TRUE to enable supervisor mode
```

Several widgets employ an optional password protected confirmation mechanism as a prerequisite to performing some operation. This is not secure and serves primarily to remind the user that some operation is not without significant consequences. These widgets also provide a *LOCK* option that disables all functional widget properties (e.g. what PV will be written, what value will be written) and inhibits further password modification. Once the *LOCK* option is selected and the file saved, the widget remains locked permanently (of course, one could simply edit the ascii edl file). Setting EDMSUPERVISORMODE to TRUE unconditionally unlocks all such widgets.

```
EDMDEBUGMODE            Set to any integer value to enable debug mode
```

For developer use only.

```
EDMXSYNC                (version 1-10-x only) Puts X in sync mode
```

This is intended for use when testing or debugging edm. X I/O throughput is significantly degraded in this mode.

```
EDMGENDOC               (version 1-10-x only) Generates widget file format
                        information on save
```

Setting this environment variable makes edm generate widget file format information when a display file is saved. The output is sent to stdout. This gives up-to-date information to those writing display generating scripts and such.

Example output for rectangle widget:

```
# (Rectangle)
object activeRectangleClass
beginObjectProperties
major <int>
minor <int>
release <int>
x <int>
y <int>
w <int>
h <int>
lineColor (index <int> | rgb <int> <int> <int>)
[lineAlarm [(0|1)]]   /* present with no value = 1, absent = 0 */
[fill [(0|1)]]   /* present with no value = 1, absent = 0 */
```

```
fillColor (index <int> | rgb <int> <int> <int>)
[fillAlarm [(0|1)]]   /* present with no value = 1, absent = 0 */
[lineWidth <int>]   /* default = 1 */
[lineStyle ("solid"|"dash")]   /* default = "solid" */
[invisible [(0|1)]]   /* present with no value = 1, absent = 0 */
[alarmPv <expandable string>] /* default = "" */
[visPv <expandable string>] /* default = "" */
[visInvert [(0|1)]]   /* present with no value = 1, absent = 0 */
[visMin <string>] /* default = "" */
[visMax <string>] /* default = "" */
endObjectProperties
```

Conventions:

```
[a] means a is optional

(a | b) means either "a" or "b"

Arrays are specified as follows:

  a {
    0 1
    1 100
  }

In this case, a is an array of 2 elements, element 0 = 1,
element 1 = 100.

Primitive types are enclosed by "<" and ">". The following means
a takes an integer value.

  a <int>

Primitive types include

  <int>
  <real>
  <string>
  <expandable string> (i.e. macros are allowed)
```

Chapter 5. The Project Environment

The environment for a given project is determined by a group of files and environment variables. The following considerations are addressed:

- Definition of the project colors, fonts, calculations, and display schemes
- Location of edm component and PV component files
- Component management
- Location of edm help files
- Display file search path
- Location of edm tmp files (.e.g for autosave)

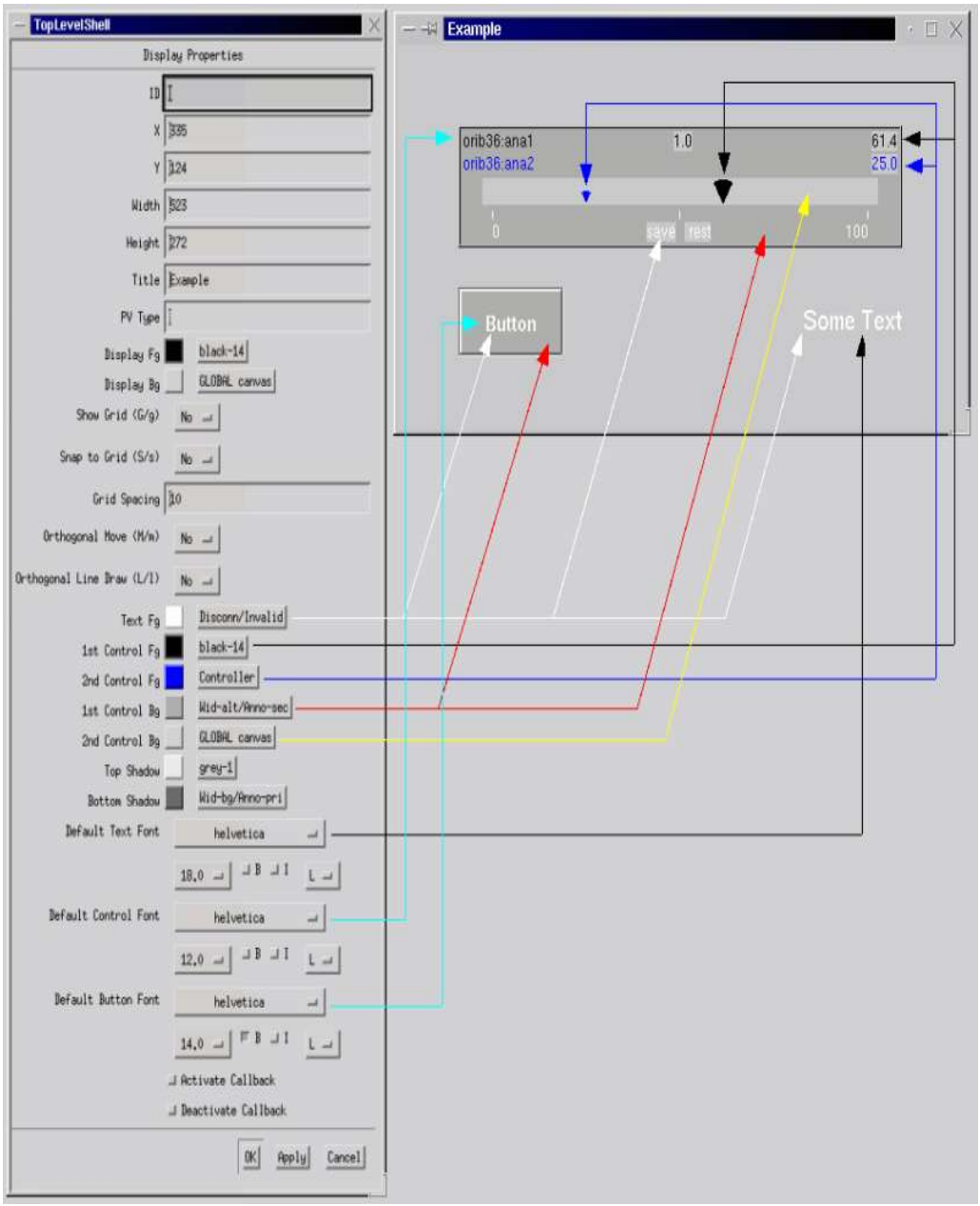
Colors, Fonts, and Display Schemes

The environment variable EDMFILES determines the location of all related files. If EDMFILES is not set, /etc/edm is used. Color and font configuration and content specifics have been given elsewhere in this document so this section will only discuss display schemes in detail.

Display Schemes

Display schemes determine the default colors and fonts used when creating widgets in a given display. The display scheme consists of the cosmetic properties shown in the display property dialog box which may be obtained by middle-clicking on a edm display (while in edit mode) and selecting *Display Properties* from the menu. Figure 5-1 shows this property box and correlates the various color names with their respective uses in a slider widget.

Figure 5-1. Display Schemes



When a new edm display file is created, the display scheme information is populated from the file named *default.scheme* located in $EDMFILES. One task in configuring the project environment is therefore creating this file. This is done as follows:

- Execute edm and create a new display
- Middle-click on the display background and select *Display Properties* from the menu
- Set the various display scheme properties as desired (Text Fg, 1st Control Fg, 2nd Control Fg, 1st Control Bg, 2nd Control Bg, Top Shadow, Bottom Shadow, Default Text Font, Default Control Font, Default Button Font)
- Press the OK button to close the dialog box
- Middle-click on the display background and select *Save Display Scheme...* from the menu
- Save the file *default.scheme* in the appropriate directory (as given by $EDMFILES)
- Exit edm now, don't save the display; when you next start edm the display scheme will be properly set.

Display Scheme Sets

A single display scheme is sometimes not adequate. It may be useful to have a separate scheme for each widget or multiple schemes, one per group of widgets per sub-system. This is possible but requires manual configuration.

The file *schemes.list* in $EDMFILES associates a scheme set name with each edm widget over some number of scheme sets. For example, suppose you have three sub-systems named *front*, *middle*, and *back*. Furthermore, suppose you only care about rectangles, sliders, and text entry widgets. Configuration consists of two steps

Step 1

Manually create (tediously) one scheme file per widget per sub-system. These could be named front-rec.scheme, middle-rec.scheme, back-rec.scheme, front-sli.scheme, middle-sli.scheme, back-sli.scheme, front-te.scheme, middle-te.scheme, and back-te.scheme.

Step 2

Create the file (with a text editor) *schemes.list* in $EDMFILES with the following contents:

```
front {
  Graphics-activeRectangleClass front-rec
  Controls-activeSliderClass    front-sli
  Controls-activeXTextDspClass  front-te
}
middle {
  Graphics-activeRectangleClass middle-rec
  Controls-activeSliderClass    middle-sli
  Controls-activeXTextDspClass  middle-te
}
back {
  Graphics-activeRectangleClass back-rec
  Controls-activeSliderClass    back-sli
  Controls-activeXTextDspClass  back-te
}
```

The component type-name (e.g. Graphics-activeRectangleClass) may be discovered by examining the

file *edmObjects*, the location of which is given by the environment variable $EDMOBJECTS (or /etc/edm if $EDMOBJECTS is not set).

Once this file is created, a scheme set may be selected by Middle-clicking on the display background and selecting *Select Scheme Set* from the menu and then selecting the scheme set name from the sub-menu.

Location of edm component and PV component files

The edm widget component file, *edmObjects*, resides in the directory given by $EDMOBJECTS or in /etc/edm if the environment variable is not set.

The edm PV component file, *edmPvObjects*, resides in the directory given by $EDMPVOBJECTS or in /etc/edm if the environment variable is not set. This file supports an obsolete class of generic PV objects and will not be further discussed. A future version of edm will use this file to manage the generic PV objects currently in use.

Component Management

Edm widgets are distributed as independent component packages. This serves as a quality control device and prevents a new, but perhaps unstable, component from affecting the stability of a previous configuration. The new component may be installed, tested, and subsequently, if a problem arises, uninstalled without any code modification.

As an example, consider the following html component which happens to be packaged separately. The library file name is libHtml.so.

The package contents may be viewed as follows:

```
edm -show /myPath/libHtml.so

Edm component file is ./edmObjects

Component                                Menu Type    Menu Text

html                                     Graphics     html
```

The package may be installed as follows:

```
edm -add /myPath/libHtml.so


Edm component file is ./edmObjects

Adding component: html, menu type: Graphics, menu text: html
```

```
Moving current edm component file to ./edmObjects~

The new component has been successfully added
```

The package may be uninstalled as follows:

```
edm -remove /myPath/libHtml.so
```

```
Edm component file is ./edmObjects

Removing component: html

Moving current edm component file to ./edmObjects~
Component was successfully removed
```

Notice than in each command, the library file reference includes an absolute path. This is a strict requirement.

Location of edm help files

Edm help files reside in the directory given by $EDMHELPFILES or in /etc/edm if the environment variable is not set.

Display file search path

The environment variable EDMDATAFILES is a colon separated list of directories. This list is searched in the given order whenever an edm display file is referenced in a related display widget and also whenever a symbol or graphical image file is refenced in some associated widget.

If the environment variable is not set, the current directory is used as the search path.

Location of edm tmp files

Edm tmp files reside in the directory given by $EDMTMPFILES or in /tmp if the environment variable is not set.

Chapter 6. PV Types

Introduction

Several different PV types are supported by some edm widgets (in version 1-10-x, all widgets support all PV types). Currently, supported types include EPICS, CALC, and LOC.

EPICS - Prefix = EPICS\ (or NULL)

The default edm PV type is EPICS and, in fact, a PV reference may include the prefix *EPICS*. So, when referring to an EPICS PV, the names *EPICS\somePv* and *somePv* are equivalent.

Calculations - Prefix = CALC\

A CALC PV must include the *CALC* prefix. For example, a legal CALC PV name is CALC\somePv and this is distinctly different from the simple name somePv (an EPICS PV).

CALC PVs may work in conjunction with a global configuration file which must be named calc.list and, for a production system, must reside in the location pointed to by $EDMFILES. If calc.list resides in the current directory it will be utilized in place of the one in $EDMFILES. This behavior facilitates development but obviously introduces a deployment pitfall. Enhancements will be made to future versions to give the same flexibility without the deployment gotcha.

A couple of simple CALC PV references could look like these that follow:

```
CALC\sum(pv1, pv2)

CALC\diff(pv1, 10.5)
```

The above definitions will produce new PVs containing the values pv1+pv2 and pv1-pv2 respectively. For example, if value1 and value2 are the names of two EPICS PVs, and an edm widget references the name CALC\sum(value1,value2), then a new PV will be created containing the value value1+value2.

The file calc.list would look like this:

```
CALC1

# Example definition file for the CALC ProcessVariable
#
# First line "CALC1 ..." is magic and provides version number
# as well as EMACS mode with is not necessary
# but makes for - by definition - cool coloring
# when this file is edited in emacs.

# Empty lines and comments like these,
# beginning with '#', are ignored.
#
# Syntax for the rest of the file:
#
# <name>
```

```
# <implementation>
#
# The name is used to refer to a CALC PV as "CALC\name"
# or "CALC\name(argA, argB, argC, ...).
#
# Implementation is in EPICS CALC record syntax,
# with A, B, C, ... being variable names for the arguments.
#
#
# Note: arguments are not checked!
# If e.g. CALC\sum(a,b) requires two arguments,
# nonone cares if you provide 0, 2 or 5 arguments instead of 2.

# Example: CALC\sum(x, y) adds the arguments
sum
# Implementation:
A+B

# Example: CALC\diff(x, y) subtracts the arguments
diff
# Implementation:
A-B
```

One serious shortcoming of this strategy is the requirement to restart edm whenever the calculation file is modified.

As of version 1-10-1d, calculation PVs may be created on-the-fly, independent of a calc.list file. The syntax extension uses curly braces to enclose the expression.

The above CALC PVs could be equivalently declared as

```
CALC\{A+B}(pv1, pv2)

CALC\{A-B}(pv1, 10.5)
```

In this case, no calculation file is necessary and new calculations may be declared on-the-fly, with no need to restart edm.

As of version 1-10-1zc, the calc.list file may contain an additional line per calculation that specifies how the argument list is to be rewritten. This can simplify usage for certain cases. For example, suppose a calculation always involves the value of a PV and its alarm severity as follows:

```
# CALC\valid(x, y) returns value if valid else -1
valid
# Implementation:
B==0?A:-1
```

The previous method for defining a pv for this calculation requires two arguments as the following example illustrates:

```
CALC\valid(pv1,pv1.SEVR)
```

The new feature permits the calculation to be specified thus:

```
# CALC\valid(x) returns value if valid else -1
valid
# The next line generates two arguments from one
@$(A),$(A).SEVR
# Implementation:
B==0?A:-1
```

The implementaion is the same, but the pv is declared as

```
CALC\valid(pv1)
```

In this case, A is rewritten as A and B as A.SEVR giving CALC\valid(pv1,pv1.SEVR).

This may also be useful when a naming convention is used to describe a group of pvs associated with a device. For example, if two instances of a device are associated with the following groups of PVs

```
dev1:vReading
dev1:iReading
dev1:tempReading

dev2:vReading
dev2:iReading
dev2:tempReading
```

and a color rule is used to indicate the state of the device, then the following old style calc PV definition, given as

```
CALC\deviceState(dev1:vReading,dev1:iReading,dev1:tempReading)

CALC\deviceState(dev2:vReading,dev2:iReading,dev2:tempReading)
```

might be replaceable by

```
CALC\deviceState(dev1)

CALC\deviceState(dev2)
```

Local - Prefix = LOC\

Local PVs are internal to edm and must include the *LOC* prefix. For example, a legal LOC PV name is LOC\somePv=<type>:<value> (e.g. LOC\somePv=d:1.234).

Specifying local PVs

Local PV types are double (d), integer (i), enumerated (e), and string (s). Example declarations are shown below.

- LOC\realPv=d:1.234
- LOC\intPv=i:55
- LOC\stringPv=s:this is a string
- LOC\enumPv=e:0,zero,one,two (initial value is "zero")
- LOC\enumPv=e:1,zero,one,two (initial value is "one")

- LOC\enumPv=e:2,zero,one,two (initial value is "two")

Edm creates a local PV the first time it is encountered. If types or values differ between two declarations, the first one wins. To ensure predictable beavior, the declarations must be identical.

Scope

The default scope of a local PV is global. By default, the PV is visible from all windows on all X displays managed by a given instance of edm. The scope may be limited to a single X display (application context) or to a single window. The following examples show how this may be accomplished.

- LOC\$(!A)realPvDisplayScope=d:1.234
- LOC\$(!W)realPvWindowScope=d:1.234

Special Functions

The following special functions are available.

- RAND() - returns random value between [0.0,1.0) for a double type (e.g. LOC\randPv=d:RAND())

Chapter 7. Creating and Editing Display Windows

File Operations

See edm online help topic *File Operations* for more information.

Setting the Default Working Directory

When edl files are opened or saved, the operation is usually performed in one of the directories listed in the EDMDATAFILES environment variable. Initially, the default file selection dialog directory is set to the first directory given in EDMDATAFILES. This default may be changed by making a selection from the *Path* menu (on the main window). The menu contains each directory in EDMDATAFILES.

Creating a New Display File

Creating a new edl file is simple. In the main window, select *New* from the *File* menu. A new window is created which may be resized as desired.

Opening an Existing Display File

In the main window, select *Open...* from the *File* menu. A file selection dialog pops up from which an edl file may be selected. As mentioned previously, the default file selection directory may be set using the *Path* menu before the open operation is performed. Once selected, the display window is opened and positioned at the previously saved origin.

Using the Mouse

See edm online help topic *Mouse Operations* for more information.

Some Definitions

- Left B means left button, Right B means right button, etc.
- Press means *press and hold*
- Click means *press and release*
- Drag means *press and hold the button while moving the cursor*
- Shift middle B click means *hold down the shift key and click the middle button*

Simple Select Operations

Left B Click

- Select single object (deselect current)

Shft-Left B Click

- Toggle selection (add to current select group)

Ctrl-Left B Click

- Cycle single object selection

  This is especially useful for finding one object among several stacked objects. Note that this only works when one and only one object is selected.

Multiple Select Operations

Middle B drag select box from Top-L to Bot-R

- Select all enclosed

Middle B drag select box from Bot-R to Top-L

- Select by corners

Create/Edit Operations

Left B drag and release

- Create new object

Left Click on selected object(s)

- Edit object(s)

Control Middle B drag over object

- Drag-and-Drop PV name

Move and Resize Operations

Left B drag interior

- Move object

Left B drag control point

- Resize object

Ctrl-Left B drag interior or control point

- Move object (prevent unintentional resize)

  This operation makes it possible to move very small objects.

Execute Mode Operations

Middle B drag over object

- Drag-and-Drop PV name

Control Middle B click over object

- Drag/Display object properties

Display Window Attributes

Various operations may be performed on edm displays and widgets from menus that are called up by middle-clicking on the display background. Three different menus are available depending on whether one, many, or no widgets are currently selected.

To set display attributes, make sure no widgets are selected and then middle-click on an edm display background. Select the *Display Properties* menu item.

Attributes include size and position, window title, the display scheme (colors and fonts), and several edit mode parameters (grip,snap...).

Note that some edit mode parameters are listed with a pair of shortcut key labels in parentheses [e.g. Show Grid (G/g)]. The shortcut keys set (uppercase) and clear (lowercase) the corresponding attribute when the attributes dialog is not open.

The *ID*, *Activate Callback*, and *Deactivate Callback* attributes are not currently used.

Creating Objects (except lines)

See edm online help topic *Creating Objects* for more information.

Objects (rectangles, meters, buttons, etc.) are created by left-dragging a box on an edm display window background. The initial size of the object is determined by the size of the box and when the left button is released, a menu pops up listing available object types (graphics, monitors, controls). Selecting a type calls up a sub-menu enumerating all available objects for the selected type. Selecting an object closes the menu and calls up an object property dialog box. After the object attributes are specified and the OK button pressed, the dialog is closed and the object is created.

Editing Objects (except lines)

See edm online help topic *Editing Objects* for more information.

Left-clicking on a selected object calls up the object property dialog box. New users often associate this operation with a double-click. Avoid this temptation. Although it is true that two successive left clicks will result in an edit operation, the clicks must be separated by some amount of delay. Object properties may be modified as desired. Pressing the Apply button applies all modifications without closing the dialog box. Pressing OK applies modifications and closes the dialog. Pressing Cancel closes the dialog without applying modifications.

The functions performed by the OK, Apply, and Cancel buttons may also be achieved by double clicking the left, center, and right mouse buttons respectively. This may, at times, eliminate excessive mouse pointer movement.

If multiple objects are selected, clicking on one of them opens the first property dialog box. When the OK button is pressed, the property box for the next item is opened, an so on, until each selected object has been processed. The edit operation is terminated when all objects have been visited or the Cancel button is pressed.

Creating Line Objects

See edm online help topic *Line Objects* for more information.

Lines are created by left-dragging a box on an edm display window background and choosing Lines from the Graphics sub-menu. Doing this calls up the line object property dialog box. After the object attributes are specified and the OK button pressed, the dialog is closed and the cursor changes slightly. At this point, the following operations may be performed.

- Left B click appends a new line segment end-point
- Right B click inserts a line segment end-point after a reference end-point (mouse pointer must be on reference end-point)
- Middle B drag moves segment end-points
- Ctrl Right B click deletes a line segment end-point (mouse pointer must be on the end-point to be deleted)
- Ctrl middle B click deletes the last line segment end-point
- Shift Left B click or Left B double-click completes the create operation

Editing Line Objects

See edm online help topic *Line Objects* for more information.

Clicking on a lone selected line object calls up a menu which provides the option to edit line properties or line segments. If the properties option is selected, editing proceeds in the same manner as that for any edm object. If the segments option is selected, the object property dialog is not called up, instead, the line is redrawn with control points around each segment end-point and all line create operations may be performed. The edit operation is terminated in a manner similar to the create operation, i.e. with a Shift Left B click or a Left B double-click. In addition, a Shift Right B click aborts all modifications and terminates the edit.

Aligning Objects

See edm online help topic *Aligning Objects* for more information.

When two or more objects are selected, various object alignment operations may be performed by middle-clicking on the display window background and selecting the desired operation from a menu. The following operations are possible:

Align Left, Right, Top, Bottom

- Aligns the specified edge of all objects with the most extreme instance (leftmost, topmost, etc.)

Center Vertical, Horizontal, Both

- This operation uses the location of the first object selected as a reference location. The operation is then performed in the context of this reference. *Center Vertical* centers all objects on an imaginary vertical line passing through the center of the reference object. *Center Horizontal* centers all objects on an imaginary horizontal line passing through the center of the reference object. *Center Both* centers all objects on an imaginary point at the center of the reference object.

Size Height, Width, Height & Width

- This operation uses the dimensions of the first object selected as reference dimensions. The operation is then performed in the context of this reference. *Size Height* sets the height of all selected objects the same as the reference height. The remaining operations work in a similar manner.

Distribute Vertical, Horizontal, Midpoint Vertical, Midpoint Horizontal, 2D

- This operation uses the location of the first object selected as a reference location. The operation is then performed in the context of this reference. *Distribute Vertical* distributes all selected objects along an imaginary vertical line passing through the center of the reference object with equal space between the edges of successive objects. *Distribute Midpoint Vertical* distributes all selected objects along an imaginary vertical line passing through the center of the reference object with equal space between the midpoints of successive objects. *Distribute Horizontal* and *Distribute Midpoint Horizontal* work in a similar fashion. *Distribute 2D* does not use a reference location. Instead, all selected objects are distributed in a row/column format using the smallest imaginary box that would contain all selected objects. The objects are distributed with equal space between the edges of successive objects.

Miscellaneous Operations

Other operations that may be performed from the display window menus are listed below.

Execute

- Put display in execute mode. This is discussed at greater length in a later section of this document.

Save

- Save current display using existing file name.

Save As...

- Save current display using specified file name (file might not yet exist).

Save To Current Path

- Save current display to last directory specified by *path* menu item. This enables a read-only display to be saved into a directory in which write access is permitted.

Select All

- Select all objects on the display.

Copy

- Copy all selected objects to the clipboard.

Cut

- Cut all selected objects to the clipboard.

Paste

- Paste objects in the clipboard to the current display using the current mouse location. Objects may be copied from one display window and pasted into another that is managed by the same edm process.

Paste in Place

- Paste in the clipboard to the current display using the exact location of the original copied objects.

This is usually used in conjunction with the *Orthogonal Move* mode to create arrays of objects that are being manually aligned.

Close

- Close the current display window.

Open...

- Open a new display file and create a new window.

Load Display Scheme...

- Load a new display scheme file (which changes the display attribute colors and fonts).

Save Display Scheme...

- Save current display colors and fonts as a display scheme file.

Edit Outliers

- Find widgets located outside the dimensions of the curreny display window and edit the first one found.

Find Main Window

- Find, deiconify if necessary, and raise the edm main window. Some window managers iconify the window if it is currently on the desktop so this operation may have to be performed twice.

Refresh

- Redraw entire contents of display window.

Help

- Launch online help.

Raise

- Raise stacking order of object(s) above all other objects.

Lower

- Lower stacking order of object(s) below all other objects.

Group

- Group all selected objects into a single composite object.

Ungroup

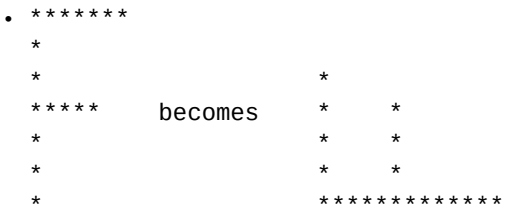
- Ungroup a composite object.

Rotate Clockwise

- 

  Not all objects may be rotated.

Rotate Counterclockwise

- 

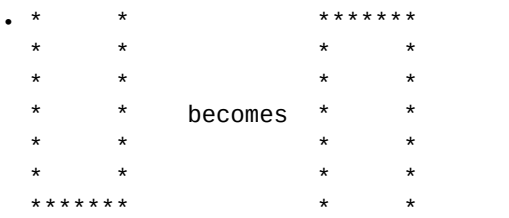
  Not all objects may be rotated.

Flip Horizontally

- 

  Not all objects may be flipped.

Flip Vertically

- 

  Not all objects may be flipped.

Edit Display Info

- Calls up a dialog box in which color and font information may be modified for all selected objects.

Edit PV Names

- Calls up a dialog box in which PV names may be modified for all selected objects.

Deselect

- Deselects all selected objects.

Chapter 8. Display Execution

To execute a newly created edm display window or one that has been opened in edit mode, make sure no widgets are selected and then middle-click on an edm display background. Select the *Execute* menu item.

In execute mode, edm connects to all PVs, creates and renders display widgets, and presents a live update of all text and graphics content, changing displayed values, colors, and images as the values of associated PVs change.

If edm is executed with the *-x* option, all display windows are opened in execute mode by default. Furthermore, if the *-noedit* option is specified, windows may not be deactivated and placed in edit mode and new display windows may not be created.

Chapter 9. Macro Symbols and Values

When edm is executed, a list of *symbol=value* pairs may be entered from the command line as follows:

```
edm -m "sym1=val1,sym2=val2,sym3=val3..."
```

These symbols may be referenced in the specification of certain object properties by embedding a reference like

```
$(symbol)
```

into the property value string. A simple example will illustrate this.

Suppose edm was started as

```
edm -m "1=one"
```

This associates the value *one* with symbol *1*. Now if an object property is specified as

```
$(1)_two_three
```

then, in execute mode, the property value would be expanded to *one_two_three*. Multiplexor objects can also affect the value of symbols in execute mode and related display objects can pass new symbols to child display windows.

Symbols are used to parameterize PV names and widget labels to build a single template display window, multiple instances of which may be used in systems containing arrays of similar subsystems.

Chapter 10. Internal Symbols

A set of internal symbols may be used by the related display widget and the shell command widget. The related display widget may assign internal symbol values to child macro arguments. The shell command widget may likewise include internal symbols references in the shell command contents.

For example, one internal symbol is <DSPNAME> which translates to the X Window display name.

The related display widget might include the following in the contents of the *macros* field:

```
Dsp=<DSPNAME>
```

The shell command widget might include the following in the contents of the *shell command* field:

```
xterm -display <DSPNAME>
```

The list of internal symbols are given below.

<WINID>

The X Window ID - this is something you might use as shown below:

```
xwd -id <WINID>
```

<TITLE>

Title of the edm display

<HELPDIR>

edm help file directory

<DSPNAME>

X Window display name

<DSPID>

X Window display name with each "." translated to a "-" (host:0.1 becomes host:0-1)

<env:*environment-variable-name*>

The contents of the associated environment variable

```
for example, <env:OSTYPE> might translate to Linux
```

Chapter 11. EDM and CVS

EDM will display an embedded CVS revision in the title bar after the file name. The CVS revision is contained in the file comment area and is specified as shown in the following example:

```
# $Revision: 1.1.2 $
```

This may initially be given as

```
# $Revision$
```

CVS will expand this form to the one above it after a commit operation is performed.

An edm comment template file may be utilized to insert comments into an edl file when the file is saved by setting the environment variable EDMCOMMENTS to the path of the template file. This provides a method for automatically inserting the CVS *Revision* symbol into edl files when they are created. So that the template file comments are inserted only once, edm prepends the following line to the comment block:

```
# <<<edm-generated-comments>>>
```

Chapter 12. Remote File Access

Introduction

It is possible to run edm on a local cpu (e.g. from home) and access edl files and some configuration files remotely (e.g. at work) from a web server. If the associated EPICS PVs are accessible through a channel access gateway, edm may be employed to monitor/troubleshoot a live production system from a remote location.

This has been implemented in rather simple-minded manner to accomodate a single user running edm from home in a technical support activity. If more that one user are using the same local cpu to run edm in this manner, each must set the environment variable EDMTMPFILES to a unique value.

You may also control the permissions of temporary files using the environment variable EDMTM-PUMASK.

==========

BY DEFAULT EDM DOES NOT HAVE REMOTE FILE ACCESS CAPABILITY.

==========

To verify the capability of a particular build instance, set the environment variable EDMDEBUGMODE to 1 and run edm. If remote file access is turned on the following will be displayed on stdout

```
Using curl for URL-based access
```

If not, you will see

```
Using local access only
```

To turn on remote file access, you must edit Makefile.Host in $EPICS_EXTENSIONS/src/edm/lib. In addition, curl must be installed on the build host and libcurl.so must be accessible.

```
For a linux build, add "-DUSECURL" to the line starting with

  USR_CXXFLAGS_Linux +=

and add "curl" to the line starting with

  USR_LIBS_Linux =

for example, change

  USR_CXXFLAGS_Linux += -Wall -D__linux__ -D_BSD_SOURCE \
  -DSINGLE_THREADED=1 -D__epics__=1

  USR_LIBS_Linux = Xm Xt Xp Xtst X11 pthread dl

to

  USR_CXXFLAGS_Linux += -Wall -DUSECURL -D__linux__ -D_BSD_SOURCE \
```

```
-DSINGLE_THREADED=1 -D__epics__=1

USR_LIBS_Linux = Xm Xt Xp Xtst X11 pthread dl curl
```

After making this change, cd back to $EPICS_EXTENSIONS/src/edm and do

```
make clean

make
```

Example Configurations

There are two ways to access remote files. The first is the most simple and makes it possible to add a remote location to the normal search paths. The second replaces the entire file access environment and facilitates the case where edm is being run from home, for example, to perform remote diagnostic activities.

Example 1 - Simple Remote Access

One way of accessing edl files on a remote http server is simply by adding a URL to the environment EDMDATAFILES. This works for EDMFILES too, but EDMFILES points to a single location.

For example, setting EDMDATAFILES as show below

export EDMDATAFILES=/top/one:/top/two:http://remoteNode/remoteTop/one

results in the following locations being searched when a file open is attempted:

```
/top/one

/top/two

http://remoteNode/remoteTop/one
```

Example 2 - Remote File Access Only

In this example, file access is exclusively remote and local access is possible only by including the "file://" prefix in some URL.

Given:

```
The files colors.list and fonts.list accessible via web server from
http://www.webserver.com/files/

Several edl files accessible via web server from
http://www.webserver.com/edlFiles/
```

```
Help files accessible via web server from
http://www.webserver.com/helpFiles/

Local edm print definition file is /edm/edmPrintDef

Local collection of .edl files in /edm/localEdlFiles
```

Define the following environment variables as shown below (bash syntax shown).

```
export EDMHTTPDOCROOT="http://www.webserver.com|file:///"

export EDMFILES=/files

export EDMDATAFILES=/edlFiles:/edm/localEdlFiles

export EDMHELPFILES=/helpFiles

export EDMPRINTDEF=/edm/edmPrintDef
```

When edm is executed, locations are searched for .edl files in the following order:

```
1) http://www.webserver.com/edlFiles

2) /edlFiles

3) http://www.webserver.com/edm/localEdlFiles

4) /edm/localEdlFiles
```

If you have trouble, set the environment variable EDMDEBUGMODE to 1 and run edm once again. You may thus observe server, directory, and file names as edm searches for valid data sources.

When edm is run in this manner, performing interactive file open operations will result in some directories being invalid. The file selection box does not support remote access.

Note that local access also requires a URL-based name, hence the second document root specification

```
  file:///
```

Many web servers may be employed in this scheme. The environment variable EDMHTTPDOCROOT is a list of locations separated by the "|" character. The following example is legal:

```
export EDMHTTPDOCROOT="http://web1.com|http://web2.com|http://web3.com"
```

Chapter 13. Input Filters

Introduction

In version 1-10-0p of edm, an input file filter mechanism was added which enables edm to read foreign display files. Configuration of this facility involves the following steps:

- Create and populate the file named edmFilters.
- Define the environment variable EDMFILTERS to point to the directory in which edmFilters resides.
- Create or obtain an executable that performs the conversion from the foreign file to an edm compatible file.

File Format for edmFilters

The edmFilters file may contain multiple entries, one per line for each file extension type. Comment and blank lines are not currently allowed.

One entry in edmFilters contains the foreign file extension and the conversion command with its arguments. The input file name is given as the symbol %f which is replaced by the actual name at run-time.

```
Example:

.adl adl2edl -rgb %f
.xml xml2edl %f
```

In the preceeding, both adl2edl and xml2edl are assumed to reside within the execution path.

Executable Requirements

The executable must accept the input file name on the command line and must send its output to stdout. Error messages should be written to stderr or to a file.