A Reference Guide for the IRAF Client Display Library (CDL)

Michael Fitzpatrick

NOAO/IRAF Group

February 1997

Revised March 1998

ABSTRACT

The Client Display Library (CDL) is a host interface for C, Fortran or SPP programs allowing them to display images or overlay graphics to display servers such as *XImtool* or *SAOimage / SAOtng*. High-level procedures allow IRAF or FITS images to be displayed simply, other routines permit access to all other server functions (e.g. cursor and image readback, frame selection, etc). The library also features a number of functions for doing image overlay graphics; supported graphics primitives include numerous point shapes, lines, circles, ellipses, polygons, annular shapes, and text.

March 4, 1998

Contents

A Reference Guide for the IRAF Client Display Library (CDL)

Michael Fitzpatrick

NOAO/IRAF Group

February 1997

Revised March 1998

1. Introduction

For more than a decade IRAF has used a *display server* as the primary means for image display. IRAF client tasks connect to the server and send or read data using a modification of the IIS Model 70 protocol, originally through named fifo pipes but more recently using unix domain or inet sockets. The advantage to this approach was that IRAF client tasks could make use of the image display functionality without duplicating the code needed for actually displaying the image. The longtime disadvantage was that the IIS protocol used was arcane and undocumented and therefore largely unavailable to applications outside of the IRAF project. The Client Display Library (CDL) provides a public C and Fortran interface for displaying images and overlay graphics that is independent of the underlying protocol used.

Unlike the interface used by IRAF applications, the CDL is meant to provide an easy-to-use, fully featured interface for applications that can be easily evolved for future display servers, communications schemes, or display functionality. Indeed, the CDL is independent of IRAF itself (as are the display servers) so display tasks can be written for any discipline or application.

While this guide assumes programs are written in C, Fortran programmers should find the translation straightforward by referring to the Fortran interface summary. The package source files include example tasks as does this guide; users with problems, questions, or bug reports are encouraged to contact *iraf@noao.edu*. A small code sample demonstrating the problem would be very helpful in finding a solution to any reported problems.

2. Getting Started

All C programs must include the header file **"cdl.h"** in order to get package definitions for constants such as colors and structure definitions used. The Fortran interface does not *require* anything similar, however for fortran compilers which support an `include` directive a **cdlftn.inc** file may be used to define symbolic constants passed to procedures, this file must be included by each procedure using the CDL. Fortran programs not using this file must pass in the constants explicitly, needed values are found throughout this manual. C procedures which return an integer value will return a positive number to indicate an error has occurred and print an error message, otherwise zero is returned.

The **cdl_open()** procedure is used to establish a connection to the server and initialize the package, it returns a CDL structure pointer that is passed to other CDL procedures. For C programs this means a separate pointer may be maintained for each server connection, the Fortran interface is limited to only one server connection per process since the pointer is maintained internally. The connection is terminated using the **cdl_close()** procedure. Between these two calls may be any combination of CDL procedure calls for doing image display or overlay graphics.

For example, the simplest possible program for displaying an IRAF image would look something like:

```
#include "cdl.h"

main (int argc, char *argv[])
{
    CDLPtr cdl = cdl_open ((char *)0);
    cdl_displayIRAF (cdl, argv[1], 1, 1, 1, 1);
    cdl_close (cdl);
}
```

This program displays band one of an image named on the command line to the server in frame one using the default 512x512 frame buffer, zscaling the pixels to 8-bit values automatically. No error checking is performed to verify that a connection was established or that the argument is a valid IRAF image. Most programs will be more complex than this but it should be clear that image display from client applications is a now trivial operation.

Synopsis

```
#include "cdl.h"

CDLPtr cdl_open (char *imtdev)
void cdl_close (CDLPtr cdl)
```

3. Server Connections

The **cdl_open()** procedure takes a single argument specifying the type of connection to make to the server, this routine also initializes the CDL package. If this is a NULL pointer the CDL will attempt to first connect on a unix domain socket, if that fails the standard IRAF /dev/imt1* fifo pipes are tried. The syntax for the *imtdev* argument is as follows:

```
<domain> : <address>
```

where <domain> is one of "**inet**" (internet tcp/ip socket), "**unix**" (unix domain socket) or "**fifo**" (named pipe). The form of the address depends upon the domain, as illustrated in the examples below. The address field may contain up to two "%d" fields. If present, the user's UID will be substituted (e.g. "unix:/tmp/.IMT%d"). The default connection if no imtdev is specified is "unix:/tmp/.IMT%d", failing that a connection is attempted on the /dev/imt1[io] named fifo pipes.

3.1. Domain Sockets

Domain sockets are sockets created on the local host. The connection is usually faster than an inet socket and comparable to a fifo. If the socket name is specified with a '%d' field the client can be assured of a unique socket name for each user allowing multiple clients to be run on the same host by different users.

Example

```
/* Connection to a local host using socket domain socket. */
if ((cdl = cdl_open ("unix:/tmp/.IMT%d")) == NULL) {
     fprintf (stderr, "cannot open domain socket connection\n");
     exit (1);
}
```

3.2. Named FIFO Pipes

This is the traditional approach, and the only one supported by SAOimage (although recent versions contain support for sockets). Any named fifo pipe may be used, the syntax for the *imtdev* string in this case is

fifo:`<input_fifo>`**:**`<output_fifo>`

<u>Example</u>

```
/* Connection to a local host using named fifo pipes. */
if ((cdl = cdl_open ("fifo:/dev/imt1i:/dev/imt1o")) == NULL) {
     fprintf (stderr, "cannot open fifo pipe connection\n");
     exit (1);
}
```

3.3. Inet Sockets

Inet sockets are connections between hosts via a tcp/ip socket. This permits connecting to the server over a remote network connection anywhere on the Internet.

<u>Example</u>

```
/* Connection to a local host using socket 5137. */
if ((cdl = cdl_open ("inet:5137")) == NULL) {
     fprintf (stderr, "cannot open inet socket connection\n");
     exit (1);
}

/* Connection to a remote internet host using socket 5137. */
if ((cdl = cdl_open ("inet:5137:foo.bar.edu")) == NULL) {
     fprintf (stderr, "cannot open inet socket connection\n");
     exit (1);
}
```

3.4. User-Defined Connections

Since IRAF V2.10.3 client tasks have been able to use an **IMTDEV** unix environment variable to set the connection type, the syntax of this variable is the same as described above. If the *cdl_open()* procedure is called with a NULL pointer the IMTDEV environment variable will automatically be checked. To explicitly use this (or any other) variable in the client task the *cdl_open()* procedure may be called as e.g.

```
if ((cdl = cdl_open (getenv("IMTDEV"))) == NULL) {
     fprintf (stderr, "cannot open server connection\n");
     exit (1);
}
```

4. Image Display

4.1. Overview of the Display Process

Basic image display is done most easily using the high-level **cdl_displayIRAF()**, **cdl_displayFITS()** and **cdl_displayPix()** procedures. These routines automatically define an image WCS, clear the frame, set the frame buffer and center the image in the display. For most applications these are all that will be needed, but the **cdl_writeSubRaster()** procedure can also be used to display an image. For example, to display one image in a mosaic or other cases where the task needs low-level access to position the image or write raw pixel values.

In these cases it is the responsibility of the client program to prepare the server for display. The basic steps involved in displaying an image include

Operation	CDL Procedure
Selecting the frame	*cdl_setFrame()*
Clear the frame	*cdl_clearFrame()*
Select the frame buffer configuration	*cdl_selectFB()*
Set the frame buffer configuration	*cdl_setFBConfig()*
Scale the image pixels to 201 display values	*cdl_zscaleImage()*
Define the image WCS	
Set the image WCS	*cdl_setWCS()*
Compute the raster placement in the frame buffer	
Write the pixels to the display	*cdl_writeSubRaster()*

In cases like a mosaic display obviously clearing the frame will only need to be done once and a single WCS for the mosaic should be defined. For simple display the high-level routines handle all of these steps for you, they are included here as checklist of what must be considered when using the CDL for low-level display.

4.2. Displaying IRAF Images

The **cdl_displayIRAF()** procedure can be used to display an IRAF OIF format image (i.e. images with a *.imh* extension) by simply passing in the image name. Pixel files for the image must be accessible from the local machine but can be in any directory, the HDR$ syntax for the imdir is also recognized. Images may be three dimensional, the *band* argument is used to select the image band to be displayed. The *frame* and *fbconfig* arguments select the frame and frame buffer size respectively, the special symbolic value **FB_AUTO** may be used for the *fbconfig* argument to have the procedure automatically select the frame buffer most appropriate for the image size. If the *zscale* flag is greater than zero the image will automatically be converted to 8-bit values using the zscale mapping algorithm. The function returns a positive value if the image cannot be accessed or displayed for any reason, an error message will be printed.

The *cdl_isIRAF()* procedure returns a positive value if the filename argument is recognized as an IRAF image, it does not check whether the pixel file can be successfully accessed. For simply reading the pixels from an IRAF image the **cdl_readIRAF()** procedure may be used. The function returns a zero value and sets the output pixel array, image dimensions and pixel size if successful, otherwise the function returns a positive value. Note that the output pixel values may need to be scaled before they can be displayed.

Synopsis
```
    int cdl_displayIRAF (CDLPtr cdl, char *fname, int band,
        int frame, int fbconfig, int zscale)
    int cdl_isIRAF (char *fname)
    int cdl_readIRAF (char *fname, int band, uchar **pix,
        int *nx, int *ny, int *bitpix, char *title)
```

4.3. Displaying FITS Images

The **cdl_displayFITS()** procedure can be used to display a *simple* FITS image by name. A "simple" FITS file is assumed to be one containing a single image and having no extensions. Other types of FITS files may of course be displayed but the client will have to use other means to import the pixels. FITS image extensions may be supported in a future release of the CDL. The *frame* and *fbconfig* arguments select the frame and frame buffer size respectively, the special symbolic value **FB_AUTO** may be used for the *fbconfig* argument to have the procedure automatically select the frame buffer most appropriate for the image size. If the *zscale* flag is greater than zero the image will automatically be converted to 8-bit values using the zscale mapping algorithm. The function returns a positive value if the image cannot be accessed or displayed for any reason, an error message will be printed.

The *cdl_isFITS()* procedure returns a positive value if the filename argument is recognized as a simple FITS image. For simply reading the image pixels the **cdl_readFITS()** procedure may be used. The output pixel array, image dimensions and pixel size are returned if successful otherwise the function returns a positive value. Note that the returned pixel values may need to be scaled before they can be displayed.

Synopsis
```
    int cdl_displayFITS (CDLPtr cdl, char *fname, int frame,
        int fbconfig, int zscale)
    int cdl_isFITS (char *fname)
    int cdl_readFITS (char *fname, uchar **pix, int *nx, int *ny,
        int *bitpix, char *title)
```

4.4. Displaying Raw Pixels

The **cdl_displayPix()** procedure can be used to display an arbitrary array of pixels of any size. The *nx* and *ny* arguments are the raster dimensions, and *bitpix* is the pixel size and has the same meaning as the FITS BITPIX keyword. The *frame* and *fbconfig* arguments select the frame and frame buffer size respectively, the special symbolic value **FB_AUTO** may be used for the *fbconfig* argument to have the procedure automatically select the frame buffer most appropriate for the image size. If the *zscale* flag is greater than zero the image will automatically be converted to 8-bit values using the zscale mapping algorithm.

Synopsis
```
    int cdl_displayPix (CDLPtr cdl, uchar *pix, int nx, int ny,
        int bitpix, int frame, int fbconfig, int zscale)
```

4.5. Frame Selection

Frame selection is normally done as an argument to one of the display procedures, however frames may be explicitly selected using the **cdl_setFrame()** procedure. This allows client programs to essentially "blink" frames independently, as long as the server supports multiple frames. The **cdl_getFrame()** procedure may be used to get the current frame set in the server.

Synopsis
```
    void cdl_setFrame (CDLPtr cdl, int frame)
    void cdl_getFrame (CDLPtr cdl, int *frame)
```

4.6. Clearing the Display

The current display frame may be explicitly cleared using the **cdl_clearFrame()** procedure. The frame is also cleared prior to displaying new images by the procedures **cdl_displayPix()**, **cdl_displayFITS()**, and **cdl_displayIRAF()**.

Synopsis
```
    int cdl_clearFrame (CDLPtr cdl)
```

4.7. Frame Buffer Selection

The default frame buffer used is 512x512, other sizes may be selected using the **cdl_setFBConfig()** procedure. To set the frame buffer size the client passes the frame buffer number as defined in the frame buffer configuration file (see below) while setting the image WCS. It is important to note that the frame buffer isn’t actually changed in the server until a subsequent **cdl_setWCS()** call, either directly or through some other procedure which sets the WCS (e.g. one of the display procedures).

To get the size of the currently defined frame buffer the user may call the **cdl_getFBConfig()** procedure. This returns not only the current configuration number, but the size as well. To get the size and any arbitrary configuration without actually setting it, the **cdl_lookupFBSize()** procedure may be used. Any configuration not actually defined in the frame buffer configuration file is returned as the default 512x512 size.

Synopsis

```
    void cdl_setFBConfig (CDLPtr cdl, int configno)
    void cdl_getFBConfig (CDLPtr cdl, int *configno, int *width,
        int *height, int *nframes)
    void cdl_lookupFBSize (CDLPtr cdl, int configno, int *width,
        int *height, int *nframes)
```

4.7.1. Automatic Selection

The **cdl_selectFB()** procedure may be used to select the most appropriate frame buffer to use for a given image size. If possible a frame buffer the same size as the image will be used, otherwise one that is larger will be chosen. Rather than simply selecting the first configuration larger than the image, the procedure searches the entire configuration file selecting the one with the least empty space in both dimensions. If the *reset* flag is non-zero this frame is set automatically by the procedure, otherwise the selected dimension is simply returned to the calling program. In either case the new frame buffer will not take effect until a new WCS is defined for the frame.

Synopsis

```
    void cdl_selectFB (CDLPtr cdl, int nx, int ny, int *fb,
        int *w, int *h, int *nf, int reset)
```

4.7.2. The Frame Buffer Configuration File

The size of the frame buffer is not passed directly to the server since this is not part of the communications protocol used. Instead, the frame buffer number is sent as part of the WCS header packet. So that both the server and client can know that a particular frame buffer number corresponds to a specific size, a *frame buffer configuration file* is used which both the client and server read.

The default configuration file is /usr/local/lib/imtoolrc, this can be overridden by defining an **IMTOOLRC** environment variable naming the file to be used, or by creating a .imtoolrc file in your home directory. Since the server must also read the same file, this must be done before starting both the client and server applications.

The format of the frame buffer configuration file is

configno nframes width height [extra fields]

e.g.

```
 1  2  512  512
 2  2  800  800
 3  1 1024 1024          # comment
 :  :   :    :
```

At most 128 frame buffer sizes may be defined, each configuration may define up to 4 frames, configuration numbers need not be sequential but should be in ascending order.

4.8. Image WCS Description

The image WCS is defined using the **cdl_setWCS()** procedure. The WCS defines a mapping between any linear coordinate system and the image pixels, for our purposes we will discuss how the WCS is used to map the frame buffer pixels to image coordinates. It is passed to the server in a string of the form:

Image_Name_String\n a b c d tx ty z1 z2 zt

where:

X’ = a*X + c*Y + tx
Y’ = b*X + d*Y + ty

The terms *a, b, c,* and *d* define a rotation of the WCS wrt the pixel coordinates, the *tx* and *ty* values are translation terms. The remaining three values define the intensity mapping of the display pixels; *z1* is the minimum pixel value used in the transformation, *z2* is the maximum value, and *zt* defines the type of transformation used (0 for none, 1 for linear, 2 for log10).

The WCS may be set explicitly by the calling program or a default appropriate for the image will be set automatically by the high-level display procedures, otherwise a WCS for the frame buffer is defined (i.e. returned coordinates are frame buffer coords). As an example of how the WCS is defined, the default WCS for an image IMX x IMY pixels in a frame buffer FBX x FBY pixels is defined as

```
a  =  1.0;                           /* no rotation */
b  =  0.0;
c  =  0.0;
d  = -1.0;
tx = (IMX / 2) - (FBX / 2) + 1;    /* center in FB */
ty = (FBY / 2) + (IMY / 2);
z1 = z1;                             /* zscale values */
z2 = z2;
zt = 1;
```

Synopsis

```
    int cdl_setWCS (CDLPtr cdl, char *name, char *title,
        float a, float b, float c, float d, float tx, float ty,
        float z1, float z2, int zt)
    int cdl_getWCS (CDLPtr cdl, char *name, char *title,
        float *a, float *b, float *c, float *d, float *tx, float *ty,
        float *z1, float *z2, int *zt)
```

4.9. Image Colormaps

The IIS protocol used does not permit the downloading of user-defined colormaps, all images are loaded as raw grayscale values according to the XImtool colormap model used by currently supported servers. All images containing private colormaps or more than the 201 grayscale values defined by the Imtool colormap model must either convert the image to 8-bit grayscale values by calling the CDL zscale procedures (**cdl_computeZscale()** and **cdl_zscaleImage()**) or scale the images in client code with user LUTs. The CDL zscale procedures scale image to 201 grayscale values so that they are displayed to the full 8-bit range, user LUT transformations or user code for converting to grayscale from a private colormap procedures should do the same.

4.9.1. Imtool Color Model

The IMTOOL color model defines at most 201 grayscale values for use in displaying the image, a set of 16 static colors are also defined for overlay graphics. Pixel values sent to the server should be already scaled to this model, i.e. the image pixels should be scaled to the range 1-200, values above this will either represent the overlay colors or will wrap around to 8-bit values. The CDL zscale procedures will automatically scale arbitrary pixel values to use this color model, the overlay procedures assume color values are defined for the static color range 201-217 but any 8-bit value may be used.

A summary of the color model values is included below:

Color	Description	Color	Description
0	Background	208	Cyan
1 - 200	Image data	209	Magenta
201	Cursor (white)	210	Coral
202	Background (black)	211	Maroon
203	White	212	Orange
204	Red	213	Khaki
205	Green	214	Orchid
206	Blue	215	Turquoise
207	Yellow	216	Violet
217	Wheat	218-255	undefined

4.10. ZScale Intensity Mapping

Since most display servers are only capable of displaying 8-bit pixel values, images with more than 8-bits per pixel must be scaled prior to display. For linear transformations this is typically done using a simple conversion of the image min/max values to the 256 grayscale values, however this doesn't produce very good results when most pixel values are near one of the extremes (usually the image min for astronomical images). To solve this IRAF uses a *zscale* mapping algorithm where a sampling grid is used to approximate the image min/max values rather than computing it directly, a line is then fit to these sample pixels to determine the optimal transformation to the display values. This is not only more efficient but maps the most common pixel values to the display range producing a better image.

The CDL has several routines for doing the same transformation: the *cdl_computeZscale()* procedure is used to compute the optimal *z1* and *z2* values (the min/max used for the zscale transform) for an image of any pixel size. The *bitpix* argument is the number of bits-per-pixel for the input array and has the same meaning as for the FITS *BITPIX* keyword. To then transform the image using these values (or user-defined values) the *cdl_zscaleImage()* procedure is used. The input pixels are modified by this procedure but the array is not reallocated to the smaller size needed by an 8-bit array. The **cdl_setSample()** and **cdl_setSampleLines()** procedures can be used to change the sampling grid and number of sample points (the default is 600 points on 5 lines). The **cdl_setContrast()** procedure can be used to change the default contrast adjustment to the slope used in the transformation (the default is 0.25). If a value of zero is given then the minimum and maximum of the intensity sample is used as the z1/z2 value.

Each of the CDL display procedures has a *zscale* flag to automatically scale the pixels prior to display. Applications wishing to set their own z1/z2 values will need to call the zscale procedures and disable this flag. By default cdl_zscaleImage() will use a linear transform, the **cdl_setZTrans()** procedure may be used to change this. Acceptable values are **CDL_UNITARY** (zero) for a unitary transform, **CDL_LINEAR** (one) for a linear transform, or **CDL_LOG** (two) for a log10 transform.

Synopsis

```
    void cdl_computeZscale (CDLPtr cdl, uchar *pix, int nx,
        int ny, int bitpix, float *z1, float *z2)
    void cdl_zscaleImage (CDLPtr cdl, uchar **pix, int nx,
        int ny, int bitpix, float z1, float z2)

    void cdl_setZTrans (CDLPtr cdl, int ztrans)
    void cdl_getZTrans (CDLPtr cdl, int *ztrans)
    void cdl_setZScale (CDLPtr cdl, float z1, float z2)
    void cdl_getZScale (CDLPtr cdl, float *z1, float *z2)

    void cdl_setSample (CDLPtr cdl, int nsample)
    void cdl_setSampleLines (CDLPtr cdl, int nlines)
    void cdl_setContrast (CDLPtr cdl, float contrast)
    void cdl_getSample (CDLPtr cdl, int *nsample)
    void cdl_getSampleLines (CDLPtr cdl, int *nlines)
    void cdl_getContrast (CDLPtr cdl, float *contrast)
```

4.11. Image Hardcopy

While most servers include some hardcopy capability of their own the CDL provides two procedures for creating hardcopy images from the client (e.g. for a batch processing application). The client will typically read back the entire image, frame buffer, of just a subraster and pass those pixels to the print procedure. Images will be written as Pseudocolor Postscript (to preserve the overlay marker colors) and may be disposed to a file using the **cdl_printPixToFile()** procedure or to any command string accepting input from *stdin* (typically just an 'lpr' command) by using the **cdl_printPix()** procedure.

Synopsis
```
    int cdl_printPix (CDLPtr cdl, char *cmd, uchar *pix, int nx,
        int ny, int annotate)
    int cdl_printPixToFile (CDLPtr cdl, char *fname, uchar *pix,
        int nx, int ny, int annotate)
```

4.12. Image Cursor

The image cursor is read using the **cdl_readCursor()** procedure. The returned value is the cursor *(x,y)* position as floating point value in terms of the currently define image WCS. Note that this position must be converted to integer if it is to be used in one of the marker procedures.

Synopsis
```
    int cdl_readCursor (CDLPtr cdl, int sample, float *x,
        float *y, char *key)
```

4.12.1. Cursor Sampling

If the cdl_readCursor() *sample* flag is non-zero the *logical image cursor* position is returned immediately, otherwise the display server will wait for a keystroke before returning the cursor position. The logical image cursor is the last value set by a *cdl_setCursor()* call or the last value returned by a *cdl_readCursor()* call. When sampling the cursor position the keystroke value is undefined.

4.13. Image Readout

The CDL maintains an internal knowledge of where an image has been positioned if it was displayed using one of the *cdl_display** procedures. The **cdl_readImage()** procedure may be used to read back the entire image pixels from the server ignoring the region of the frame buffer outside of the image, the **cdl_readFrameBuffer()** procedure will read back the contents of the entire frame buffer. The dimensions of the array are returned in the *nx* and *ny* arguments.

Synopsis
```
    int cdl_readImage (CDLPtr cdl, uchar **pix, int *nx,
        int *ny)R
    int cdl_readFrameBuffer (CDLPtr cdl, uchar **pix,
        int *nx, int *ny)R
```

4.14. Subraster I/O

The **cdl_writeSubRaster()** procedure is used to write an arbitrary raster to any location in the display. Similarly the **cdl_readSubRaster()** procedure is used to read back an arbitrary raster. When an image has previously been displayed the subraster position is given in image coordinates (e.g. when writing a subregion of edited pixels), otherwise the position is in frame buffer coordinates (e.g. to display multiple images per frame you should use the cdl_writeSubRaster() call). See the section on *Marker Coordinates* for further explanation of the coordinate systems used.

Synopsis
```
    int cdl_writeSubRaster (CDLPtr cdl, int lx, int ly, int nx,
        int ny, uchar *pix)
    int cdl_readSubRaster (CDLPtr cdl, int lx, int ly, int nx,
        int ny, uchar **pix)
```

5. Graphics Overlay

5.1. Marker Coordinates

All marker positions are assumed to be image pixel coordinates, although there is no requirement that the position be on the image itself. When an image WCS is defined (using the CDL display procedures or explicitly) the origin of the coordinates used shifts from the frame buffer lower-left to the lower-left of the image as displayed in the frame. Negative positions are allowed and will either refer to empty pixels if the frame buffer is larger than the image, or pixels outside the frame buffer boundaries. Raster I/O requests will be clipped to the frame buffer endpoints, a request completely outside the frame buffer is an error.

5.2. Mapping a Previously Displayed Image

Ideally any application wishing to draw markers on an image will have also displayed that image, however the **cdl_mapFrame()** procedure may be used to map the requested frame for marker overlay. It does this by reading the WCS defined for that frame and assumes an image has been displayed and centered in the frame buffer, then resets the internal CDL image position. If no image has been displayed the frame buffer is mapped directly. This can be used for example to map an empty frame for displaying just the markers without an image, or for mapping another frame's WCS for use on the current display. The frame is not changed by the procedure call however the current WCS *is* changed.

Synopsis
```
    int cdl_mapFrame (CDLPtr cdl, int frame)
```

5.3. Marking a Coordinate File

Since a common function for programs will be to mark a list of coordinates, the high-level **cdl_markCoordsFile()** procedure is provided to make this easier. The input parameters include a filename expected to contain a set of (x,y) points (real or integer), and arguments specifying the point type, size and color to draw. If the *label* argument is positive each marker point will be labeled with it's relative number in the file. The size, type and color arguments all have the same meaning as for the **cdl_markPoint()** procedure described below.

Synopsis
```
    int cdl_markCoordsFile (CDLPtr cdl, char *fname, int type,
        int size, int color, int label)
```

5.4. Marker Colors

Markers may be drawn using any 8-bit value, in order to use the static overlay colors the color must be in the range 201-217 (see above for notes on the XImtool color model). The "*cdl.h*" include file for C programs, the "*cdlftn.inc*" include for fortran programs, or the "*cdlspp.h*" include for SPP programs, defines the following symbolic constants for each of the static overlay colors:

C_BLACK	202	**C_CORAL**	210
C_WHITE	203	**C_MAROON**	211
C_RED	204	**C_ORANGE**	212
C_GREEN	205	**C_KHAKI**	213
C_BLUE	206	**C_ORCHID**	214
C_YELLOW	207	**C_TURQUOISE**	215
C_CYAN	208	**C_VIOLET**	216
C_MAGENTA	209	**C_WHEAT**	217

5.5. Marker Types

Currently supported marker types include:

Point	*Line*	*Box*	*Polyline*	*Polygon*
Circle	*Circular Annuli*	*Ellipse*	*Elliptical Annuli*	*Text*

The "*cdl.h*" include file for C programs, the "*cdlftn.inc*" include for fortran programs, or the "*cdlspp.h*" include file SPP programs, defines the following symbolic constants for each of the defined *Point* marker types:

M_FILL	1	**M_CIRCLE**	64
M_POINT	2	**M_STAR**	128
M_BOX	4	**M_HLINE**	256
M_PLUS	8	**M_VLINE**	512
M_CROSS	16	**M_HBLINE**	1024
M_DIAMOND	32	**M_VBLINE**	2048

Point markers are drawn using the **cdl_markPoint()** procedure, point types may be logically *OR*'d to create composite markers, closed shapes such as a circles, diamonds, or squares may be *OR*'d with the M_FILL flag to flood-fill the point with the current overlay color.

5.5.1. Point

The **cdl_markPoint()** procedure is used to mark a specific point on the image using one of the marker types listed above. The marker is centered at the coordinates specified by the *x* and *y* arguments, *type* is an integer flag indicating what kind of marker to draw and may be a composite type by logically ORing two or more marker types. *Size* is the width and height of the marker measured in pixel unxits, and *color* is the color used to draw the marker. If the *number* argument is greater than zero that number will be drawn next to the point as a label, creating text labels for point markers can be done using the *cdl_markPointLabel* procedure.

Most marker names are fairly obvious but several are worth special mention: The M_DIAMOND, M_CIRCLE and M_BOX marker types may be logically *OR*ed with the M_FILL flag to produce a filled marker type. Unless *OR*d with the M_POINT flag all point markers will leave the center pixel unchanged. The M_HLINE and M_VLINE markers are most useful in astronomical applications to mark an individual star, they are horizontal and vertical lines respectively with a gap in the middle third of the marker (the M_HBLINE and M_VBLINE are identical but with a width of 3 pixels).

Synopsis

```
    int cdl_markPoint (CDLPtr cdl, int x, int y, int number,
        int size, int type, int color)
    int cdl_markPoint (CDLPtr cdl, int x, int y, char *label
        int size, int type, int color)
```

5.5.2. Line

The **cdl_markLine()** procedure is used to draw a line of the specified color between points (*xs*,*ys*) and (*xe*,*ye*).

Synopsis
```
    int cdl_markLine (CDLPtr cdl, int xs, int ys, int xe, int ye,
        int color)
```

5.5.3. Box

The **cdl_markBox()** procedure is used to draw a box of the specified color with endpoints specified by (*lx*,*ly*) and (*ux*,*uy*). If the *fill* flag is set the box will be filled with the marker color.

Synopsis
```
    int cdl_markBox (CDLPtr cdl, int lx, int ly, int ux, int uy,
        int fill, int color)
```

5.5.4. Circle

The **cdl_markCircle()** procedure is used to draw a circle of the specified color with a center at (*x*,*y*) and radius *radius*. If the *fill* flag is set the circle will be filled with the marker color.

Synopsis
```
    int cdl_markCircle (CDLPtr cdl, int x, int y, int radius,
        int fill, int color)
```

5.5.5. Polyline

The **cdl_markPolyline()** procedure is used to draw a line connecting the *npts* points specified by the *xpts* and *ypts* array in the desired *color*.

Synopsis
```
    int cdl_markPolyline (CDLPtr cdl, int *xpts, int *ypts,
        int npts, int color)
```

5.5.6. Polygon

The **cdl_markPolygon()** procedure is used to draw a closed polygon consisting of *npts* vertices specified by the *xpts* and *ypts* array in the desired *color*. The last point in the array will automatically be connected to the first point by the procedure. If the *fill* flag is set the circle will be filled with the marker color.

Synopsis
```
    int cdl_markPolygon (CDLPtr cdl, int *xpts, int *ypts,
        int npts, int fill, int color)
```

5.5.7. Ellipse

The **cdl_markEllipse()** procedure is used to draw an ellipse of the specified color with a center at (*x*,*y*) and semimajor-axis *xrad* and semiminor-axis *yrad* pixels long. A rotation angle for the ellipse may be specified by passing a non-zero *angle* argument, the angle is measured in degrees from the positive x-axis. If the *fill* flag is set the circle will be filled with the marker color.

Synopsis
```
    int cdl_markEllipse (CDLPtr cdl, int x, int y, int xrad,
        int yrad, float ang, int fill, int color)
```

5.5.8. Circular Annuli

The **cdl_markCircAnnuli()** procedure is used to draw *nannuli* circles separated by *sep* pixels each. The circle is centered at (*x,y*) with an initial radius of *radius* pixels.

Synopsis

```
    int cdl_markCircAnnuli (CDLPtr cdl, int x, int y, int radius,
        int nannuli, int sep, int color)
```

5.5.9. Elliptical Annuli

The **cdl_markEllipAnnuli()** procedure is used to draw *nannuli* ellipses separated by *sep* pixels each. The ellipse is centered at (*x,y*) with an initial semimajor and semiminor axis specified by the *xrad* and *yrad* arguments. Each ellipse will be optionally rotate by an *angle* degrees as measured from the positive x-axis.

Synopsis

```
    int cdl_markEllipAnnuli (CDLPtr cdl, x, y, xrad, yrad, ang,
        int nannuli, int sep, int color)
```

5.5.10. Text

The **cdl_markText()** procedure is used to draw a text string specified by *str* argument with an initial position at (*x,y*) and optionally rotated by *angle* degrees as measured from the positive x-axis. The default *size* is 1.0 and is approximately a 6x13 font, the font size may be scaled by any fractional amount.

Synopsis

```
    int cdl_markText (CDLPtr cdl, int x, int y, char *str,
        float size, float angle, int color)
```

5.6. Text Fonts

The **cdl_setFont()** procedure is used to choose between one of four available fonts as the text marker default: Roman, Greek, Futura, Bold and Times respectively. By default the Roman font will be used. The width of the lines used to draw the text may also be set.

Synopsis

```
    void cdl_setFont (CDLPtr cdl, int font)
    void cdl_setTextWidth (CDLPtr cdl, int width)
```

A complete listing of the Greek character mappings can be found in the file ’greek.ps’ in the ’doc’ subdirectory of the CDL distribution. The *Roman* font is the font implemented in the original version of the CDL and works well for most applications. Both the *Greek* and *Times* fonts are hi-resolution fonts which work best for larger frame buffers but can produce publication quality text. The *Futura* font is a simpler font which can produce better results than the default on small size frame buffers. A *Bold* font automatically increases the text line width by one pixel over the current setting and may be used with any font.

5.6.1. In-Line Font Changes

Text markers are drawn using the font selected with the *cdl_setFont()* routine, however fonts may be change within a string itself (e.g. to set a Greek character) using a \f escape sequence. The escape is followed by the character ’R’ to set a Roman font, ’G’ for Greek, ’F’ for futura, ’B’ for bold and ’T’ for Times. Any number of escapes are permitted within a string, the font change will remain in effect until it is changed, or the end of string at which point any subsequent strings will again be drawn with the default font. Additionally a ’P’ in the escape sequence will change the font to the one previously used, whatever that may be.

The CDL also supports a sub/superscripting of text which can only be done with the font escapes. In this case the escape character followed by a ’U’ produces a superscript and a ’D’ produces a subscript. The changes may be nested permitting several levels of sub/superscripts, these escapes may also be used in conjunction with a font change to cause the sub/superscript to be drawn with a different font. A superscript escape will remain in effect until the end of the string or a \fD escape is seen. Similarly a subscript remains in effect until the end of the string of a \fU escape. Sub/superscripted text is drawn using a smaller font size, there is presently no way to specify a different size for the sub/superscripted text.

Summary of Font Escapes

\fR	change to Roman font
\fG	change to Greek font
\fF	change to Futura font
\fT	change to Times font
\fB	change to bold font
\fP	change to previous font
\fU	begin relative superscripted text
\fD	begin relative subscripted text

5.7. Line Widths and Styles

The **cdl_setLineWidth()** procedure can be used to set the line width used to draw polygon or polyline markers, point markers will not be affected. The **cdl_setLineStyle()** procedure is used to set a line style other than solid.

Synopsis

```
    void cdl_setLineWidth (CDLPtr cdl, int width)
    void cdl_setLineStyle (CDLPtr cdl, int style)
```

The "*cdl.h*" include file for C programs, the "*cdlftn.inc*" include for fortran programs, or the "*cdlspp.h*" include file SPP programs, defines the following symbolic constants for each of the defined line styles:

L_SOLID	0	**L_DASHED**	1
L_DOTTED	2	**L_DOTDASH**	3
L_HOLLOW	4	**L_SHADOW**	5

The *hollow* line style is drawn with a linewidth of five pixels, two pixels of color, a black line, and two pixels of color. It is best used when the marker will traverse extreme changes in brightness, due to the thickness of the line it may work best with larger frame buffers. The *shadow* linestyle is drawn as two pixels of color and two pixels of black and should be used for similar brightness variations, however it effectively shows up as a line only two pixels wide and may be preferred for medium or smaller frame buffers.

The three dashed linestyles are drawn using "gap" spacings of 5 pixels in between line segments. Whether or not these gaps are resolved depends on the size of the frame buffer being used and the magnification used in the display server. By default they should resolve completely using frame buffers up to 1024x1024 pixels, or magnification factors displaying 1024x1024 pixels. If larger sizes are needed the image should be subsampled prior to display to maintain the marker resolution needed for these linestyles.

5.8. Deleting Markers

When markers are drawn the underlying subraster is first saved to an internal structure, erasure is done by simply redisplaying the saved raster. Problems can arise however when markers overlap; when deleting a marker that is *under* another marker the original pixels can overwrite the pixels of the marker on top. This is an unfortunate side effect of the simple scheme used in this version of the package, users can call the **cdl_redrawOverlay()** procedure to help clean up any artifacts left behind.

5.8.1. Individual Markers

The **cdl_deleteMark()** procedure is used to delete a single marker from the display(). The (*x*,*y*) argument is either the center position of the marker if that is know by the application, more typically it will be an approximate position. In the latter case the marker whose center is closest to this position will be deleted. For markers with no defined center the distance used to decide if the marker should be deleted is the distance from the argument position to the edge of the marker. For example, distance from a box or polygon is measured as the distance from to one of the sides, for text it is the distance to the start of the text string. There is no way to *un*delete a marker other than to redraw it.

Synopsis

```
    int cdl_deleteMark (CDLPtr cdl, int x, int y)
```

5.8.2. The Entire Overlay

To erase all markers currently displayed use the **cdl_clearOverlay()** procedure. Markers are erased in the reverse order they were drawn to help reduce the chance that overlaying markers will leave stray pixels.

Synopsis

```
    int cdl_clearOverlay (CDLPtr cdl)
```

5.9. Redraw

The **cdl_redrawOverlay()** procedure can be used to redraw all markers currently in the display list. This is sometimes needed when subraster I/O procedures are used to redisplay subregions and overwrite existing markers.

Synopsis

```
    int cdl_redrawOverlay (CDLPtr cdl)
```

6. ANSI C Function Prototypes

The current release of CDL provides full ANSI C function prototypes for all public and private procedures. By default these will not be used even on systems with native ANSI compilers, once these have been more thoroughly tested they may be enabled on systems which support ANSI compilers. In the meantime, to make use of the CDL prototypes users will need to define the macro **CDL_ANSIC** either when compiling the program or as a definition in the program source preceding the ’cdl.h’ include directive.

For example,

```
    #define  CDL_ANSIC
    #include "cdl.h"
         :
    main (int argc, char **argv)
         :
```

or when compiling using something like

```
    cc -DCDL_ANSIC client.c libcdl.a -lm
```

Function prototypes haven’t been extensively tested as of this date, please report any problems found.

7. Fortran Language Binding Notes

The Fortran language binding routines are implemented in C but should be accessible from any fortran program as though they were real fortran subroutines. The calling sequences are the same as with the C library routines with the following exceptions:

- The CDL package pointer is maintained internally so no 'cdl' pointer is passed in the fortran interface.
- All routines which are integer procedures in the C interface return an extra 'ier' argument to contain the error flag. All Fortran functions are implemented as subroutines.
- The procedure names are the same except that *cdl_* has been replaced with *cf* in the fortran binding. If your compiler is case-sensitive then use all lower case letters.

The binding has been tested on a number of different platforms without problems. The procedure names haven't been restricted to the traditional 6-character fortran names since most modern compilers can handle longer names, if yours isn't one of them contact *iraf@noao.edu* for help in changing the names.

Since the CDL is implemented as a set of C routines, the one aspect that cannot be overlooked in the fortran binding is the between Fortran and C storage order for arrays. In most cases this will not be a problem since the CDL routines are just passing around pointers even if they live for a short while in a fortran program. The problem comes when using the fortran program to read the arrays, for example in using the array returned by the **cfreadIRAF()** procedure, or when passing in arrays for display that originated in the user's fortran code. In these cases the array **must** be transposed to be interpreted correctly. It was assumed that in most applications arrays returned by CDL procedures would be immediately passed to other CDL procedures so having the binding routines transpose the array to/from Fortran storage order was unnecessarily inefficient. This may be changed in later releases if required.

8. SPP Language Binding Notes

The SPP language binding is experimental and is intended to provide a way to quickly prototype tasks, it should not be used in production code as it may not be as portable as the rest of the task. In essence this binding is a layer on top of the Fortran binding since most IRAF platforms still use Fortran as the intermediate code. The calling sequences are the same as with the Fortran library routines with the following caveats:

- The 'cdlspp.h' SPP include file is required by all files which call CDL routines. The binding names are actually SPP macros to resolve the current 6 character limit on procedure names.
- All character string arguments must be dimensioned to at least SZ_FNAME characters in length.
- The CDL package pointer is maintained internally so no 'cdl' pointer is passed in the fortran interface.
- All routines which are integer procedures in the C interface return an extra 'ier' argument to contain the error flag. All SPP functions are implemented as subroutines.
- On HPUX or IBM RS6000 systems the 'cdlspp.h' file must be edited to remove the trailing underscores from the procedure name macros. This is because on these platforms the fortran compiler will not append an underscore to the SPP symbols as it does on other platforms.

9. IIS Protocol Description

The communications protocol used by the CDL and servers such as *XImtool* and *SAOimage*, is a slightly modified version of that used by the IIS Model 70. All operations are initiated by sending a header packet containing a *thing id* (tid) and *subunit* selecting the function to be performed, optionally followed by data up to 32K bytes long. The IIS header packet used is defined as

```
        struct  iism70 {
            short   tid;
            short   thingct;
            short   subunit;
            short   checksum;
            short   x, y, z;
            short   t;
        };
```

The *thing count* field contains the negative number of bytes of data that will be sent following the header packet. The IIS header checksum is computed as

```
    checksum = 0177777 - (tid + subunit + thingct + x + y + z + t);
```

The four IIS registers are set differently depending on the operation, a summary of the header packets for each operation is summarized below.

IIS Header Packet Summary

	TID	Subunit	Thingct	X	Y	Z	T	Data
Read Data	IIS_READ|PACKED	MEMORY	-nbytes	x	y	frame	-	nbytes
Write Data	IIS_WRITE|PACKED	MEMORY	-nbytes	x	y	frame	-	nbytes
Read Cursor	IIS_READ	IMCURSOR	-	-	-	-	-	-
Write Cursor	IIS_WRITE	IMCURSOR	-	x	y	wcs	-	-
Set Frame	IIS_WRITE	LUT|COMMAND	-1	-	-	-	-	2
Write WCS	IIS_WRITE|PACKED	WCS	-N	-	-	frame	fb	N
Read WCS	IIS_READ	WCS	-	-	-	-	-	320
Erase Frame	IIS_WRITE | fb	FEEDBACK	-	-	-	frame	-	-

Where	nbytes	= number of bytes expected or written
	x	= x position of operation in frame buffer coords
	y	= y position of operation in frame buffer coords
	frame	= frame number (passed as bitflag (i.e. 1, 2 ,4 8, etc)
	fb	= frame buffer config number (zero indexed)
	N	= length of WCS string
	wcs	= WCS number (usually zero)
	Data	= the number of bytes of data to be read or written following the header packet.

IIS_WRITE	= 0400000
IIS_READ	= 0100000
COMMAND	= 0100000
PACKED	= 0040000
IMC_SAMPLE	= 0040000

MEMORY	= 001
LUT	= 002
FEEDBACK	= 005
IMCURSOR	= 020
WCS	= 021

TID fields can be logically OR'd with the PACKED flag indicating the number of data bytes is exactly *thingct* bytes long, otherwise *thingct* must be specified as half the number of data bytes. In a cursor read, if the IIS_READ flag is OR'd with IMC_SAMPLE the logical cursor position (i.e. the last value read or set) is returned immediately, otherwise the server will wait for a keystroke to be hit before returning a string containing the (x,y) position, wcs of the read, and the keystroke. When setting the frame you must send a short integer in the data containing the frame selected.

10. VXIMTOOL Proxy/Display Server Usage

VXIMTOOL is a image display server process much like *XIMTOOL,* except that all it normally does is respond to datastream requests to read and write to internal frame buffers maintained as arrays in memory. Multiple frame buffers and frame buffer configurations are supported. It can be used to debug CDL programs by printing out the protocol packets received, or can simply be used as a dummy server in cases where no image display is really needed. By enabling the -*proxy* flag the server can also be used to repeat the datastream requests to a list of other servers, effectively splitting the image display to a number of other servers. See the *vximtool* man page for details on other command-line arguments and usage.

The program was originally intended as a debugging tool, either in the development of CDL clients directly or in cases where the display may need to go to separate screens as part of a larger project. For example, engineers may wish to "eavesdrop" on the system by viewing images displayed by CDL clients used as part of a data acquisition system. It can also be used as a memory-only display server for CDL clients which need to be run in the background as part of a pipeline processing system requiring a frame buffer for image marking.

In proxy mode the program acts as a relay for the IIS datastream packets, sending image data, frame requests, etc. to a list of other servers specified on the command line. The effect of this is to allow a client to display to this program which then re-displays to each of the other named servers. Of course CDL clients can also do this internally by opening multiple connections, using *vximtool* in proxy mode adds the functionality to programs which may use this feature only ocasionally. A maximum of 8 servers may be named, they may be either on the local host or a remote machine and connections can be established using either fifos or sockets. See above or the *vximtool* man page for details on how to specify the server connection.

The current implementation has a few restrictions users should keep in mind:

- The time to display an image or perform any output operation scales with the number of connected hosts. Each IIS packet is forwarded to each host in turn before processing the next input packet, and connection over a slow network will delay the entire process.

- Cursor and image readback are done by sending the request *only* to the first server named on the command line. This is done to avoid forcing a cursor mode on all servers which cannot be terminated when a response is received from only one server, and means that the first server named should be the one used to control interactive sessions. The remaining servers however can still respond to cursor requests from other applications connected to that server on another channel.

- All named servers must be running prior to starting the proxy server. The connection to the remote servers is established when this task is first run and if no server is running that connection will be ignored. The task will exit if no remote servers can be found for display.

- Any connected server that shuts down while the proxy server is running is likely to cause the program to crash on the next display.

11. C Interface Summary

```
#include  "cdl.h"

        CDLPtr cdl_open        (imtdev)
       int cdl_displayPix      (cdl, pix, nx, ny, bitpix, frame, fbconfig, zscale)
     char cdl_readCursor       (cdl, sample, x, y, key)
        int cdl_setCursor      (cdl, x, y, wcs)
           int cdl_setWCS      (cdl, name, title, a, b, c, d, tx, ty, z1, z2, zt)
           int cdl_getWCS      (cdl, name, title, a, b, c, d, tx, ty, z1, z2, zt)
       void cdl_setFrame       (cdl, frame)
      int cdl_clearFrame       (cdl)
           void cdl_close      (cdl)

     int cdl_displayIRAF       (cdl, fname, band, frame, fbconfig, zscale)
          int cdl_isIRAF       (fname)
       int cdl_readIRAF        (fname, band, pix, nx, ny, bitpix, title)

     int cdl_displayFITS       (cdl, fname, frame, fbconfig, zscale)
          int cdl_isFITS       (fname)
        int cdl_readFITS       (fname, pix, nx, ny, bitpix, title)

  void cdl_computeZscale       (cdl, pix, nx, ny, bitpix, z1, z2)
    void cdl_zscaleImage       (cdl, pix, nx, ny, bitpix, z1, z2)

        int cdl_printPix       (cdl, cmd, pix, nx, ny, annotate)
  int cdl_printPixToFile       (cdl, fname, pix, nx, ny, annotate)

       int cdl_readImage       (cdl, pix, nx, ny)
 int cdl_readFrameBuffer       (cdl, pix, nx, ny)
   int cdl_readSubRaster       (cdl, lx, ly, nx, ny, pix)
  int cdl_writeSubRaster       (cdl, lx, ly, nx, ny, pix)

       void cdl_selectFB       (cdl, nx, ny, fb, w, h, nf, reset)
    void cdl_setFBConfig       (cdl, configno)
    void cdl_getFBConfig       (cdl, configno, w, h, nf)
   void cdl_lookupFBSize       (cdl, configno, w, h, nf)

      void cdl_setZTrans       (cdl, ztrans)
      void cdl_setZScale       (cdl, z1, z2)
      void cdl_setSample       (cdl, nsample)
 void cdl_setSampleLines       (cdl, nlines)
    void cdl_setContrast       (cdl, contrast)
        void cdl_setName       (cdl, imname)
       void cdl_setTitle       (cdl, imtitle)

       void cdl_getFrame       (cdl, frame)
      void cdl_getZTrans       (cdl, ztrans)
      void cdl_getZScale       (cdl, z1, z2)
      void cdl_getSample       (cdl, nsample)
 void cdl_getSampleLines       (cdl, nlines)
    void cdl_getContrast       (cdl, contrast)
        void cdl_getName       (cdl, imname)
       void cdl_getTitle       (cdl, imtitle)
```

int **cdl_mapFrame**	(cdl, frame)
int **cdl_markCoordsFile**	(cdl, fname, type, size, color, label)
int **cdl_markPoint**	(cdl, x, y, number, size, type, color)
int **cdl_markPointLabel**	(cdl, x, y, label, size, type, color)
int **cdl_markLine**	(cdl, xs, ys, xe, ye, color)
int **cdl_markBox**	(cdl, lx, ly, ux, uy, fill, color)
int **cdl_markPolygon**	(cdl, xarray, yarray, npts, fill, color)
int **cdl_markPolyline**	(cdl, xarray, yarray, npts, color)
int **cdl_markCircle**	(cdl, x, y, radius, fill, color)
int **cdl_markCircAnnuli**	(cdl, x, y, radius, nannuli, sep, color)
int **cdl_markEllipse**	(cdl, x, y, xrad, yrad, rotang, fill, color)
int **cdl_markEllipAnnuli**	(cdl, x, y, xrad, yrad, ang, nannuli, sep, color)
int **cdl_markText**	(cdl, x, y, str, size, angle, color)
int **cdl_setFont**	(cdl, font)
int **cdl_setTextWidth**	(cdl, width)
int **cdl_setLineWidth**	(cdl, width)
int **cdl_setLineStyle**	(cdl, style)
int **cdl_deleteMark**	(cdl, x, y)
int **cdl_clearOverlay**	(cdl)
int **cdl_redrawOverlay**	(cdl)

12. C Example Tasks

The examples shown here are for demonstration purposes only. They are based on working example tasks in the CDL source *examples* subdirectory, see the programs there for the full program listing.

12.1. Display Example

```
#include <stdio.h>
#include <unistd.h>
#include "cdl.h"

/*   DISPLAY -- Example task to display an image as a command-line task.
 *  This task is meant to show three ways the CDL can be used to display
 *  an image, see the code comments for a description of each method.
 *
 *  Examples:
 *      To display a simple IRAF or FITS file:
 *          % ./display -frame 2 image.imh
 *          % ./display image.fits
 *
 *      To display a FITS file as a raw image:
 *          % ./display -nx 512 -ny 512 -depth 16 -hskip 5760 -raw dpix.fits
 *
 *  Usage:
 *      display [-depth N] [-fits] [-frame N] [-fbconfig N] [-hskip N]
 *          [-iraf] [-nozscale] [-nx N] [-ny N] [-raw] [-zscale] file
 */

#define NONE    -1
#define   IRAF  0
#define   FITS  1
#define   RAW   2

main (argc, argv)
int  argc;
char *argv[];
{
     CDLPtr    cdl;
     char *fname, title[128];
     int  i, status = 0, frame = 1, fbconfig = 0, zscale = 1;
     int  format = NONE, nx = 0, ny = 0, depth = 8, hskip = 0;
     float     z1, z2;
     int  fb_w, fb_h, nf;
     unsigned char *pix = NULL;

     /*  Process the command line options.  */
     if (argc > 1) {
         for (i=1; i < argc; i++) {
            if (strcmp (argv[i], "-depth") == 0)         depth = atoi (argv[++i]);
            else if (strcmp (argv[i], "-fits") == 0)     format = FITS;
            else if (strcmp (argv[i], "-frame") == 0)    frame = atoi (argv[++i]);
            else if (strcmp (argv[i], "-fbconfig") == 0) fbconfig = atoi (argv[++i]);
            else if (strcmp (argv[i], "-hskip") == 0)    hskip = atoi (argv[++i]);
            else if (strcmp (argv[i], "-iraf") == 0)     format = IRAF;
            else if (strcmp (argv[i], "-nozscale") == 0) zscale = 0;
            else if (strcmp (argv[i], "-nx") == 0)       nx = atoi (argv[++i]);
            else if (strcmp (argv[i], "-ny") == 0)       ny = atoi (argv[++i]);
            else if (strcmp (argv[i], "-raw") == 0)      format = RAW;
            else if (strcmp (argv[i], "-zscale") == 0)   zscale = 1;
         }
     }
```

```
/*  Open the package and a connection to the server.  */
if (!(cdl = cdl_open ((char *)getenv("IMTDEV"))) )
   exit (-1);

fname = argv[argc-1];

/*  METHOD 1:  Displays the image using the high-level format display
  * call.  Display as an IRAF image if the option was set indicating
  * this is the format, otherwise test the file to see if it is anyway.
  */
if (format == IRAF || (format == NONE && cdl_isIRAF (fname))) {
    status = cdl_displayIRAF (cdl, fname, 1, frame, FB_AUTO, zscale);

/*  METHOD 2:  Uses the CDL procedure for getting image pixels from
  * a known format, minimal work required to display an image.  The
  * point here is that you can use this method to process the image
  * yourself prior to display, e.g. subsample the pixels, apply a user
  * LUT, etc but still use the CDL to get the raw image and do the
  * display.
  */
} else if (format == FITS || (format == NONE && cdl_isFITS (fname))) {

        /*  Get the FITS image pixels, exit w/ an error status if something
     * went wrong, the procedure will print what that was.
         */
        if (cdl_readFITS (fname, &pix, &nx, &ny, &depth, title)) {
            cdl_close (cdl);      /* close the package  */
             exit (1);               /* exit w/ error code */
        }

        /*  Now select a frame buffer large enough for the image. The
    * fbconfig number is passed in the WCS packet, but the display
    * call below will compute the correct WCS for the image and
    * transmit that prior to display, all we're doing here is
    * setting up the FB to be used.
         */
        if (fbconfig == 0)
            cdl_selectFB (cdl, nx, ny, &fbconfig, &fb_w, &fb_h, &nf, 0);

       /*  Lastly, display the pixels to the requested frame, do any
   * zscaling requested using the CDL procedure.
        */
        status = cdl_displayPix (cdl, pix, nx, ny, depth, frame,
      fbconfig, zscale);

/*  METHOD 3:  Displays an image of raw pixels.  The client code is
  * responsible for reading the image and calling all the procedures
  * needed for image display, initialize the frame, zscaling pix, etc.
  * While we assume a simple raster format in this program, the user
  * code can read a compressed image format such as GIF, mosaic multiple
  * images for display as a single image, or just about anything that
  * produces a raster for display. The intent here is to show all the
  * lowest level calls needed for displaying the image.
  */
} else if (format == RAW) {
    FILE  *fd;
    int        lx, ly;

    if (nx == 0 || ny == 0) {
        fprintf (stderr, "No size given for raw data.\n");
        exit (1);
    }

    /*  Open the image file if we can.  */
    if (fd = fopen (fname, "r")) {
            /*  Seek to the offset specified.  */
            lseek (fileno(fd), (off_t) hskip, SEEK_SET);

            /*  Allocate the pixel pointer and read the data.  */
            pix = (unsigned char *) malloc (nx * ny * (depth / 8));
            fread (pix, depth/8, nx * ny, fd);

            /*  If we're zscaling and depth is more than 8-bits, do that.  */
            if (zscale && depth > 8) {
                cdl_computeZscale (cdl, pix, nx, ny, depth, &z1, &z2);
                cdl_zscaleImage (cdl, &pix, nx, ny, depth, z1, z2);
            }

            /*  Now select a frame buffer large enough for the image.  We'll
             * ask that this be reset but the change won't go to the server
             * until we send in a WCS, so compute that as well.  For the
             * WCS we assume a simple linear transform where the image is
             * Y-flipped, the (x,y) translation is computed so it is correct
             * for an frame buffer >= than the image size.
             */
                  cdl_selectFB(cdl, nx, ny, &fbconfig, &fb_w, &fb_h, &nf,1);
                  cdl_setWCS (cdl, fname, NULL, 1., 0., 0., -1.,
                (float) (nx / 2) - (fb_w / 2) + 1,    /*  X trans.  */
                (float) (fb_h / 2) + (ny / 2),    /*  Y trans.  */
                z1, z2, CDL_LINEAR);         /*  Z transform  */

            /*  Select and clear the requested frame prior to display.  */
            cdl_setFrame (cdl, frame);
            cdl_clearFrame (cdl);

            /*  Now display the pixels.  We'll compute the image placement
             * ourselves and write the image as a raw subraster of the frame
             * buffer.  In this case we'll center the image, but the CDL
             * cdl_writeSubRaster() procedure can be used to write arbitrary
             * rasters at any point in the frame buffer.
             */
            lx = (fb_w / 2) - (nx / 2);
            ly = fb_h - ((fb_h / 2) + (ny / 2));
                  status = cdl_writeSubRaster (cdl, lx, ly, nx, ny, pix);
        } else
            status = 1;
    } else {
        if (access (fname, F_OK) == 0)
            fprintf (stderr, "'%s': unknown image format.\n", fname);
        else
            fprintf (stderr, "'%s': image does not exist.\n", fname);
        status = 1;
    }

    /*  Now just free the pixel pointer to clean up.  */
    if (pix)
        free ((unsigned char *) pix);
    cdl_close (cdl);        /*  close the package  */
    exit (status);
}
```

12.2. Interactive Graphics Overlay Example

```
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include "cdl.h"

/*
 * TVMARK -- Example task for displaying an marking images.  This program
 * can be used to either display an image and overlay points defined in
 * a coordinate file, map an existing display frame for marking, or option-
 * ally enter a cursor command loop after either of these providing other
 * marking capability.  All options support minimum match.
 *
 * Examples:
 *      % tvmark dpix.fits
 *      % tvmark -coords coords -color 205 dpix.fits
 *      % tvmark -frame 2
 *      % tvmark -coords coords -interactive dpix.fits
 *
 * Usage:
 *      tvmark [-frame N] [-fbconfig N] [-coords <file>] [-size N] [-color N]
 *          [-nolabel] [-fill] [-interactive] [image]
 */

main (argc, argv)
int  argc;
char *argv[];
{
     CDLPtr    cdl;
     char *fname = NULL, *cfname = NULL;
     int  i, status = 0, fill = 0, frame = 1, fb = FB_AUTO, zscale = 1;
     int  color = 201, label = 1, size = 9, interactive = 0;
     float     z1, z2;
     int  fb_w, fb_h, nf;
     unsigned char *pix = NULL;

     /*  Process the command line options.  */
     if (argc > 1) {
         for (i=1; i < argc; i++) {
            if (strncmp(argv[i], "-color",4) == 0) color = atoi (argv[++i]);
            else if (strncmp(argv[i], "-coords",4) == 0) cfname = argv[++i];
            else if (strncmp(argv[i], "-fbconfig",3) == 0) fb = atoi (argv[++i]);
            else if (strncmp(argv[i], "-fill",4) == 0) fill = 1;
            else if (strncmp(argv[i], "-frame",3) == 0) frame = atoi (argv[++i]);
            else if (strncmp(argv[i], "-interactive",4) == 0) interactive = 1;
            else if (strncmp(argv[i], "-nolabel",4) == 0) label = 0;
            else if (strncmp(argv[i], "-nozscale",4) == 0) zscale = 0;
            else if (strncmp(argv[i], "-size",2) == 0) size = atoi (argv[++i]);
         else
             fname = argv[i];
         }
     }

     /*  Open the package and a connection to the server.  */
     if (!(cdl = cdl_open ((char *)getenv("IMTDEV"))) )
        exit (-1);

     /*  If an image was specified display it first, otherwise assume the
      * image has already been loaded in the frame and mark that.
      */
     if (fname) {
         if (cdl_isIRAF (fname))
             status = cdl_displayIRAF (cdl, fname, 1, frame, fb, zscale);
         else if (cdl_isFITS (fname))
```

```
                status = cdl_displayFITS (cdl, fname, frame, fb, zscale);
            else {
             if (access (cfname, F_OK) == 0)
                    fprintf (stderr, "'%s': unknown image format.\n", fname);
             else
                    fprintf (stderr, "'%s': image doesn't exist.\n", fname);
                status = 1;
            }
            if (status)  goto err_;
        } else {

            /*  If we've requested a special frame buffer, set it now.  */
            if (fb > 0)
               cdl_setFBConfig (cdl, fb);

            /*  Map the current display frame for use as an image.  */
            cdl_mapFrame (cdl, frame);
        }

        /*  If a coordinate file was specified read the file and mark those
         * coords with points.
         */
        if (cfname)
               cdl_markCoordsFile (cdl, cfname, M_STAR, size, color, label);

        /*  Lastly, start up an interactive cursor loop if needed.  */
        if (interactive)
            tvmInteractive (cdl, label, fill, color, size);

        /*  Close the package and clean up.  */
err_:   cdl_close (cdl);
        exit (status);
}

/*    TVMINTERACTIVE -- Process commands interactively.    */

tvmInteractive (cdl, label, fill, color, size)
CDLPtr    cdl;
int  label, fill, color, size;
{
        float   angle = 0.0, rx, ry, txsize = 1.;
        int     nx, ny, i, x, y, x2, y2;
        int     number=1, radius=11, xrad=11, yrad=6, nannuli=3, sep=5;
        char    key, cmd[SZ_NAME], str[SZ_NAME];
        unsigned char *pix;

        /*  Process commands until a 'q' keystroke is hit.  */
        while (cdl_readCursor (cdl, 0, &rx, &ry, &key) != 'q') {
            x = (int) (rx + 0.5); /*  convert to int pixels  */
            y = (int) (ry + 0.5);

            switch (key) {
            case ':':             /*  process a colon command  */
             putchar (':');
             gets (str);
             for (i=0; str[i] != ' ' && str[i]; i++)
                 cmd[i] = str[i];
             cmd[i++] = ' ';

             if (strcmp (cmd, "angle") == 0)        angle = atof (&str[i]);
             else if (strcmp (cmd, "color") == 0)   color = atoi (&str[i]);
             else if (strcmp (cmd, "fill") == 0)    fill = atoi (&str[i]);
             else if (strcmp (cmd, "number") == 0)  number = atoi (&str[i]);
             else if (strcmp (cmd, "nannuli") == 0) nannuli = atoi (&str[i]);
             else if (strcmp (cmd, "label") == 0)   label = atoi (&str[i]);
```

```
 else if (strcmp (cmd, "sep") == 0)     sep = atoi (&str[i]);
 else if (strcmp (cmd, "size") == 0)    size = atoi (&str[i]);
 else if (strcmp (cmd, "txsize") == 0)  txsize = atof (&str[i]);
 else if (strcmp (cmd, "xrad") == 0)    xrad = atoi (&str[i]);
 else if (strcmp (cmd, "yrad") == 0)    yrad = atoi (&str[i]);
 else if (strcmp (cmd, "print") == 0) {
     cdl_readFrameBuffer (cdl, &pix, &nx, &ny);
     cdl_printPix (cdl, NULL, pix, nx, ny, 1);
 } else if (strcmp (cmd, "snap") == 0) {
     cdl_readFrameBuffer (cdl, &pix, &nx, &ny);
     cdl_printPixToFile (cdl, &str[i], pix, nx, ny, 1);
 } else if (strcmp (cmd, "status") == 0) {
     printf ("angle  = %-5.3gcolor  = %d", angle, color);
     printf ("fill   = %-5dnumber   = %d\n", fill, number);
     printf ("nannuli= %-5dsep = %d", nannuli, sep);
     printf ("size   = %-5dtxsize   = %g\n", size, txsize);
     printf ("xrad   = %-5dyrad= %d", xrad, yrad);
     printf ("label  = %-5d\n", label);
 }
 break;

case '?':
 /* ......help procedures  */
 break;

case 'p':                    /*  plus mark */
 cdl_markPoint (cdl, x, y, (label ? number++ : 0), size, M_PLUS, color);
 break;
case 'x':                    /*  cross mark      */
 cdl_markPoint (cdl, x, y, (label ? number++ : 0), size, M_CROSS, color);
 break;
case '.':                    /*  point mark      */
 cdl_markPoint (cdl, x, y, (label ? number++ : 0), size, M_POINT, color);
 break;
case '*':                    /*  star mark */
 cdl_markPoint (cdl, x, y, (label ? number++ : 0), size, M_STAR, color);
 break;
case '_':                    /*  horiz dash mark*/
 cdl_markPoint (cdl, x, y, (label ? number++ : 0), size, M_HBLINE, color);
 break;
case '|':                    /*  vert dash mark   */
 cdl_markPoint (cdl, x, y, (label ? number++ : 0), size, M_VBLINE, color);
 break;
case 'o':                    /*  circle mark      */
 cdl_markPoint (cdl, x, y, (label ? number++ : 0), size, M_CIRCLE|fill, color);
 break;
case 's':                    /*  square mark      */
 cdl_markPoint (cdl, x, y, (label ? number++ : 0), size, M_BOX|fill, color);
 break;
case 'v':                    /*  diamond mark   */
 cdl_markPoint (cdl, x, y, (label ? number++ : 0), size, M_DIAMOND|fill, color);
 break;

case 'b':                    /*  Box             */
 printf ("Hit another key to define the box...\n");
 (void) cdl_readCursor (cdl, 0, &rx, &ry, &key);
 x2 = (int) (rx + 0.5);         y2 = (int) (ry + 0.5);
 cdl_markBox (cdl, x, y, x2, y2, fill, color);
 break;
case 'c':                    /*  Circle          */
 printf ("Hit another key to set radius ...\n");
 (void) cdl_readCursor (cdl, 0, &rx, &ry, &key);
 x2 = (int) (rx + 0.5);         y2 = (int) (ry + 0.5);
 radius = (int) sqrt ((double) ((x2-x)*(x2-x) + (y2-y)*(y2-y)));
 cdl_markCircle (cdl, x, y, radius, fill, color);
```

```
             break;
            case ’d’:                  /*  Delete marker    */
             cdl_deleteMark (cdl, x, y);
             break;
            case ’e’:                  /*  Ellipse    */
             cdl_markEllipse (cdl, x, y, xrad, yrad, angle, fill, color);
             break;
            case ’l’:                  /*  Line             */
             printf ("Hit another key to set line endpoint...\n");
             (void) cdl_readCursor (cdl, 0, &rx, &ry, &key);
             x2 = (int) (rx + 0.5);         y2 = (int) (ry + 0.5);
             cdl_markLine (cdl, x, y, x2, y2, color);
             break;
            case ’t’:                  /*  Text string       */
             printf ("Text string: ");
             gets (str);
             cdl_markText (cdl, x, y, str, txsize, angle, color);
             break;
            case ’C’:                  /*  Circular annuli*/
             cdl_markCircAnnuli (cdl, x, y, radius, nannuli, sep, color);
             break;
            case ’D’:                  /*  Delete all markers*/
             cdl_clearOverlay (cdl);
             break;
            case ’E’:                  /*  Elliptical annuli*/
             cdl_markEllipAnnuli (cdl, x, y, xrad, yrad, angle, nannuli, sep, color);
             break;
            default:
             break;
            }
        }
}
```

12.3. Image Mosaic Example

```
#include <stdio.h>
#include <unistd.h>
#include "cdl.h"

/* MOSAIC -- Example task to mosaic several images on a display. Demonstrates
 * usage of low-level routines for complex display operations.
 */

main (argc, argv)
int      argc;
char     *argv[];
{
        CDLPtr          cdl;
        char     *fname = NULL, title[128];
        int      i, j, k, status=0, label=0, frame=1, fb=FB_AUTO, zscale=1;
        int      sample=1, pad=0, col=204, imx, imy, bitpix, nimages, nim;
        int      ii, xinit, rowx, rowy, nnx, nny, fb_w, fb_h, nf, mx, my, nx, ny;
        float    z1, z2;
        unsigned char *pix = NULL;

        /* Process the command line options. */
        if (argc > 1) {
          for (i=1; i < argc; i++) {
          if (strncmp (argv[i], "-fbconfig",3) == 0) fb=atoi(argv[++i]);
            else if (strncmp (argv[i],"-frame",3) == 0) frame=atoi(argv[++i]);
            else if (strncmp (argv[i],"-color",3) == 0) col=atoi(argv[++i]);
            else if (strncmp (argv[i],"-label",4) == 0) label=1;
            else if (strncmp (argv[i],"-nozscale",4) == 0) zscale=0;
            else if (strncmp (argv[i],"-nx",3) == 0) nx=atoi(argv[++i]);
            else if (strncmp (argv[i],"-ny",3) == 0) ny=atoi(argv[++i]);
            else if (strncmp (argv[i],"-pad",4) == 0) pad=atoi(argv[++i]);
            else if (strncmp (argv[i],"-sample",4) == 0) sample=atoi(argv[++i]);
            else
                break;
          }
        }
        nimages = argc - i;

        /* Open the package and a connection to the server. */
        if (!(cdl = cdl_open ((char *)getenv("IMTDEV"))) )
           exit (-1);

        /* Clear the frame to begin. */
        (void) cdl_clearFrame (cdl);

        /* Loop over each of the images in the list. */
        nim = rowx = rowy = nnx = nny = 0;
        for (k=0; k < ny && nim < nimages; k++) {
            rowy += nny + pad;
            for (rowx = xinit, j=0; j < nx && nim < nimages; j++) {

                /* Get the image name for display. */
                fname = argv[i++];

                /* Figure out what kind of image it is and get the pixels. */
                if (cdl_isIRAF (fname))
                    status = cdl_readIRAF (fname, 1, &pix, &imx, &imy, &bitpix, title);
                else if (cdl_isFITS (fname))
                    status = cdl_readFITS (fname, &pix, &imx, &imy, &bitpix, title);
                else {
                    fprintf(stderr, "'%s': unknown or nonexistant image.\n", fname);
                    status = 1;
                }
                if (status)  goto err_;
                /* Compute subsampled image size. */
                if (sample > 1)
                    nnx = imx / sample, nny = imy / sample;
                else
                    nnx = imx, nny = imy;

                /* Unless we asked for a specific FB size find one large enough
                 * to handle the mosaic.  We don't check to be sure what's
                 * returned is really large enough.
                 */
                if (nim == 0 && fb == FB_AUTO)
                    cdl_selectFB (cdl, nx*nnx+(pad*(nx-1)), ny*nny+(pad*(ny-1)), &fb, &fb_w, &fb_h, &nf, 1);
                else {
                    cdl_setFBConfig (cdl, fb);
                    cdl_lookupFBSize (cdl, fb, &fb_w, &fb_h, &nf);
                }

                /* Define a WCS for the frame. */
                cdl_setWCS (cdl, "image mosaic", title, 1., 0., 0., -1., 0., (float) ny*imy+(pad*(ny+1)), 1., 255., 1);

                /* The first time through figure out the placement so the
                 * entire mosaic is centered in the frame.
                 */
                if  (nim == 0) {
                    mx = (nx * nnx) + pad * (nx-1);
                    my = (ny * nny) + pad * (ny-1);
                    rowy = (fb_h - my) / 2;
                    xinit = rowx = (fb_w - mx) / 2;
                }

                /* Compute the zscaled imaged pixels. */
                if (zscale) {
                    cdl_computeZscale (cdl, pix, imx ,imy, bitpix, &z1, &z2);
                    cdl_zscaleImage (cdl, &pix, imx ,imy, bitpix, z1, z2);
                }

                /* Subsample the image if requested. */
                if (sample > 1) {
                    int l, m, n=0;
                    for (l=0; l < imy; l+=sample)
                        for (m=0; m < imx; m+=sample)
                            pix[n++] = pix[(l*imx)+m];
                }

                /* Write the image to the frame buffer. */
                if (cdl_writeSubRaster (cdl, rowx, rowy, nnx, nny, pix)) goto err_;

                /* Draw the image name as a label. */
                if (label) cdl_markText (cdl, rowx+10, rowy+10, fname, 1., 0., col);

                nim++;          rowx += nnx + pad;
            }
        }

        /* Close the package and clean up. */
err_:   cdl_close (cdl);
        exit (status);
}
```

13. Fortran Interface Summary

include "**cdlftn.inc**"

cfopen	(imtdev, ier)
cfdisplayPix	(pix, nx, ny, bitpix, frame, fbconfig, zscale, ier)
cfreadCursor	(sample, x, y, key, ier)
cfsetCursor	(x, y, wcs, ier)
cfsetWCS	(name, title, a, b, c, d, tx, ty, z1, z2, zt, ier)
cfgetWCS	(name, title, a, b, c, d, tx, ty, z1, z2, zt, ier)
cfsetFrame	(frame)
cfclearFrame	(ier)
cfclose	()
cfdisplayIRAF	(fname, band, frame, fbconfig, zscale, ier)
cfisIRAF	(fname, isiraf)
cfreadIRAF	(fname, band, pix, nx, ny, bitpix, title, ier)
cfdisplayFITS	(fname, frame, fbconfig, zscale, ier)
cfisFITS	(fname, isfits)
cfreadFITS	(fname, pix, nx, ny, bitpix, title, ier)
cfcomputeZscale	(pix, nx, ny, bitpix, z1, z2)
cfzscaleImage	(pix, nx, ny, bitpix, z1, z2)
cfprintPix	(cmd, pix, nx, ny, annotate, ier)
cfprintPixToFile	(fname, pix, nx, ny, annotate, ier)
cfreadImage	(pix, nx, ny, ier)
cfreadFrameBuffer	(pix, nx, ny, ier)
cfreadSubRaster	(lx, ly, nx, ny, pix, ier)
cfwriteSubRaster	(lx, ly, nx, ny, pix, ier)
cfselectFB	(nx, ny, fb, w, h, nf, reset)
cfsetFBConfig	(configno)
cfgetFBConfig	(configno, w, h, nf)
cflookupFBSize	(configno, w, h, nf)
cfsetZTrans	(ztrans)
cfsetZScale	(z1, z2)
cfsetSample	(nsample)
cfsetSampleLines	(nlines)
cfsetContrast	(contrast)
cfsetName	(imname)
cfsetTitle	(imtitle)
cfgetFrame	(frame)
cfgetZTrans	(ztrans)
cfgetZScale	(z1, z2)
cfgetSample	(nsample)
cfgetSampleLines	(nlines)
cfgetContrast	(contrast)
cfgetName	(imname)
cfgetTitle	(imtitle)

cfmapFrame	(frame, ier)
cfmarkPoint	(x, y, number, size, type, color, ier)
cfmarkcoordsfile	(fname, type, size, color, label, ier)
cfmarkPointLabel	(x, y, label, size, type, color, ier)
cfmarkLine	(xs, ys, xe, ye, color, ier)
cfmarkBox	(lx, ly, ux, uy, fill, color, ier)
cfmarkPolygon	(xarray, yarray, npts, fill, color, ier)
cfmarkPolyline	(xarray, yarray, npts, color, ier)
cfmarkCircle	(x, y, radius, fill, color, ier)
cfmarkCircAnnuli	(x, y, radius, nannuli, sep, color, ier)
cfmarkEllipse	(x, y, xrad, yrad, rotang, fill, color, ier)
cfmarkEllipAnnuli	(x, y, xrad, yrad, ang, nannuli, sep, color, ier)
cfmarkText	(x, y, str, size, angle, color, ier)
cfsetfont	
cfsettextwidth	(width)
cfsetlinewidth	(width)
cfsetlinestyle	(style)
cfdeleteMark	(x, y, ier)
cfclearOverlay	(ier)
cfredrawOverlay	(ier)

14. Fortran Example Tasks

The examples shown here are for demonstration purposes only. They are based on working example tasks in the CDL source *examples* subdirectory, see the programs there for the full program listing.

14.1. Display Example

```
C =========================================================================
C  FDISPLAY -- Example fortran program showing the use of the Client
C  Display Library (CDL) Fortran interface for displaying images.
C =========================================================================

      PROGRAM FDISPLAY
      character*64   imname

C     Initialize the CDL package
      call cfopen (0, ier)
      if (ier .gt. 0) then
          write (*,*) 'open: Error return from CDL'
          goto 999
      endif

      write (*, "('Image Name: ', $)")
      read (5, *) imname
      write (*, "('Frame Number: ', $)")
      read (5, *) iframe
      write (*, "('Frame buffer configuration number: ', $)")
      read (5, *) ifb

C     If we've got a FITS format image, go ahead and display it.
      call cfisFITS (imname, isfits)
      if (isfits .gt. 0) then
          call cfdisplayFITS (imname, iframe, ifb, 1, ier)
      else
C         We've got an IRAF format image, go ahead and display it.
          call cfisIRAF (imname, isiraf)
          if (isiraf .gt. 0) then
              call cfdisplayIRAF (imname, 1, iframe, ifb, 1, ier)
          else
C             Unrecognized image, punt and exit.
              write (*,*) 'Unrecognized image format'
          endif
      endif

C     Clean up and exit.
999   continue
      call cfclose (ier)
      end
```

14.2. Interactive Graphics Overlay Example

```
C ==========================================================================
C  FTVMARK --  Example fortran program showing the use of the Client
C  Display Library (CDL) Fortran interface for doing graphics overlay. No
C  checking of the error flag is done here for space considerations.
C ==========================================================================

      PROGRAM FTVMARK
      include   "cdlftn.inc"
      character*64   imname

C     Initialize the CDL package
      call cfopen (0, ier)

      write (*, "('Image Name: ', $)")
      read (5, *) imname
      write (*, "('Frame Number: ', $)")
      read (5, *) iframe
      write (*, "('Frame buffer configuration number: ', $)")
      read (5, *) ifb

C     If we've got a FITS format image, go ahead and display it.
      call cfisFITS (imname, isfits)
      if (isfits .gt. 0) then
          call cfdisplayFITS (imname, iframe, ifb, 1, ier)
      else
C         We've got an IRAF format image, go ahead and display it.
          call cfisIRAF (imname, isiraf)
          if (isiraf .gt. 0) then
              call cfdisplayIRAF (imname, 1, iframe, ifb, 1, ier)
          else
C             No valid image given, so map the current display for marking.
              call cfmapFrame (iframe)
          endif
      endif

C     Now that we've got an image displayed or mapped, enter a cursor loop to mark the image.
      call markInteractive ()

C     Clean up and exit
999   continue
      call cfclose (ier)
      end

C     MARKINTERACTIVE -- Subroutine for processing the cursor loop.
      subroutine markInteractive ()
      include   "cdlftn.inc"
      real      angle, rx, ry, txsize
      integer        nx, ny, x, y, x2, y2, fill, size, color
      integer        number, radius, xrad, yrad, nannuli, sep
      character key
      character*64   cmd, str

C     Allocate a 1024x1024 array for pixels.
      character pix(1048576)

C     ....Initialize the local parameters to use

C     Read a cursor keystroke telling us what to do.
10    call cfreadCursor (0, rx, ry, key, ier)

C     Round the real cursor position to integer pixel positions.
          x = nint (rx + 0.5)
          y = nint (ry + 0.5)
```

```
C       Check the keystroke and take the appropriate action.
C           Colon Commands
            if (key .eq. ':') then
C             Read a three character command and value field and process the colon command
              read (*,'(A3, i4)') cmd, ival
              if (cmd(1:3) .eq. 'ang') then
                  angle = real (ival)
              else if (cmd(1:3) .eq. 'col') then
                  color = ival
              else if (cmd(1:3) .eq. 'fil') then
                  fill = ival
                   :
                   ....and so on to set local variables
                   :
              else if (cmd(1:3) .eq. 'pri') then
C                 Print contents of the current frame buffer
                  call cfreadFrameBuffer (pix, nx, ny, ier)
                  call cfprintPix ("lpr", pix, nx, ny, 1, ier)
              else if (cmd(1:3) .eq. 'sta') then
                   ....print out the status (value) of variables
              endif

C         Point Markers
          else if (key .eq. 'p') then
              call cfmarkPoint (x, y, 1, size, M_PLUS, color, ier)
          else if (key .eq. 'x') then
              call cfmarkPoint (x, y, 1, size, M_CROSS, color, ier)
          else if (key .eq. '_') then
              call cfmarkPoint (x, y, 1, size, M_HBLINE, color, ier)
          else if (key .eq. 'o') then
C             Example of a filled point marker
              call cfmarkPoint (x, y, 1, size, or(M_CIRCLE,fill), color, ier)
                   :
                   ....and so on to set other types of point markers

C         Other Markers
          else if (key .eq. 'b') then
              print '("Hit another key to define the box ....")'
              call cfreadCursor (0, rx, ry, key, ier)
              x2 = nint (rx + 0.5)
              y2 = nint (ry + 0.5)
              call cfmarkBox (x, y, x2, y2, fill, color, ier)
          else if (key .eq. 'd') then
              call cfdeleteMark (x, y, ier)
          else if (key .eq. 'e') then
              call cfmarkEllipse (x, y, xrad, yrad, angle, fill, color, ier)
         else if (key .eq. 't') then
            print '("Text string:  ", $)'
            read (*,'(A64)') str
            call cfmarkText (x, y, str, txsize, angle, color, ier)
                   :
                   ....and so on to set other types of markers

C         Quit
          else if (key .eq. 'q') then
              goto 998
          endif

C       Loop back until we want to quit
        goto 10
998     continue
        end
```

15. SPP Interface Summary

#include "**cdlspp.h**"

cdl_open	(imtdev, ier)
cdl_displayPix	(pix, nx, ny, bitpix, frame, fbconfig, zscale, ier)
cdl_readCursor	(sample, x, y, key, ier)
cdl_setCursor	(x, y, wcs, ier)
cdl_setWCS	(name, title, a, b, c, d, tx, ty, z1, z2, zt, ier)
cdl_getWCS	(name, title, a, b, c, d, tx, ty, z1, z2, zt, ier)
cdl_setFrame	(frame)
cdl_clearFrame	(ier)
cdl_close	()
cdl_displayIRAF	(fname, band, frame, fbconfig, zscale, ier)
cdl_isIRAF	(fname, isiraf)
cdl_readIRAF	(fname, band, pix, nx, ny, bitpix, title, ier)
cdl_displayFITS	(fname, frame, fbconfig, zscale, ier)
cdl_isFITS	(fname, isfits)
cdl_readFITS	(fname, pix, nx, ny, bitpix, title, ier)
cdl_computeZscale	(pix, nx, ny, bitpix, z1, z2)
cdl_zscaleImage	(pix, nx, ny, bitpix, z1, z2)
cdl_printPix	(cmd, pix, nx, ny, annotate, ier)
cdl_printPixToFile	(fname, pix, nx, ny, annotate, ier)
cdl_readImage	(pix, nx, ny, ier)
cdl_readFrameBuffer	(pix, nx, ny, ier)
cdl_readSubRaster	(lx, ly, nx, ny, pix, ier)
cdl_writeSubRaster	(lx, ly, nx, ny, pix, ier)
cdl_selectFB	(nx, ny, fb, w, h, nf, reset)
cdl_setFBConfig	(configno)
cdl_getFBConfig	(configno, w, h, nf)
cdl_lookupFBSize	(configno, w, h, nf)
cdl_setZTrans	(ztrans)
cdl_setZScale	(z1, z2)
cdl_setSample	(nsample)
cdl_setSampleLines	(nlines)
cdl_setContrast	(contrast)
cdl_setName	(imname)
cdl_setTitle	(imtitle)
cdl_getFrame	(frame)
cdl_getZTrans	(ztrans)
cdl_getZScale	(z1, z2)
cdl_getSample	(nsample)
cdl_getSampleLines	(nlines)
cdl_getContrast	(contrast)
cdl_getName	(imname)
cdl_getTitle	(imtitle)

cdl_mapFrame	(frame, ier)
cdl_markCoordsFile	(fname, type, size, color, label, ier)
cdl_markPoint	(x, y, number, size, type, color, ier)
cdl_markPointLabel	(x, y, label, size, type, color, ier)
cdl_markLine	(xs, ys, xe, ye, color, ier)
cdl_markBox	(lx, ly, ux, uy, fill, color, ier)
cdl_markPolygon	(xarray, yarray, npts, fill, color, ier)
cdl_markPolyline	(xarray, yarray, npts, color, ier)
cdl_markCircle	(x, y, radius, fill, color, ier)
cdl_markCircAnnuli	(x, y, radius, nannuli, sep, color, ier)
cdl_markEllipse	(x, y, xrad, yrad, rotang, fill, color, ier)
cdl_markEllipAnnuli	(x, y, xrad, yrad, ang, nannuli, sep, color, ier)
cdl_markText	(x, y, str, size, angle, color, ier)
cdl_setFont	(font)
cdl_setTextWidth	(width)
cdl_setLineWidth	(width)
cdl_setLineStyle	(style)
cdl_deleteMark	(x, y, ier)
cdl_clearOverlay	(ier)
cdl_redrawOverlay	(ier)
cdl_setDebug	(level)