chunky_autodocs

Andrew King <oondy@bigfoot.com>

COLLABORATORS			
	TITLE : chunky_autodocs		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Andrew King <oondy@bigfoot.com>	August 7, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

chunky_autodocs

1.1 chunky.library

```
                                  chunky.library functions and commands
          (c) 1999 Rosande Limited, all rights reserved.
    http://www.q-soft.demon.co.uk and http://www.q-group.demon.co.uk

          Author: Andrew "Oondy" King - oondy@bigfoot.com


               LICENSE
               - Please read before using this library!

               Submissions
               - Who has contributed to this project

LIBRARY FUNCTIONS

               CHK_InitChunky()

               CHK_InitColours()

               CHK_FreeChunky()

               CHK_DrawChunkyArea()

               CHK_DrawChunky()

               CHK_InsertChunky()

               CHK_CreateChunkyFromBitMap()

               CHK_CreateChunkyFromRastPort()

               CHK_SetDrMd()

               CHK_SetAPen()

               CHK_SetOPen()

               CHK_SetABOPen()
```

CHK_Move()

CHK_WritePixel()

CHK_ReadPixel()

CHK_Draw()

CHK_DrawLine()

CHK_DrawRect()

CHK_RectFill()

CHK_DrawEllipse()

CHK_SetRast()

CHK_SetSoftStyle()

CHK_SetFont()

CHK_TextLength()

CHK_Text()

CHK_TextCentre()

CHK_ChooseHardwareMode()

CHK_DrawChunkyChunkyArea()

CHK_DrawChunkyChunky()

CHK_DrawTransparentRectangle()

CHK_GetChunkyPort()

CHK_PutChunkyColours()

CHK_DrawChunkyTiled()

CHK_C2POff()
(v3.1+)

CHK_DrawChunkyWindowArea()
(v3.2+)

CHK_DrawChunkyWindow()
(v3.2+)

CHK_QueryUseOS()
(v3.2+)

PUBLIC STRUCTURES

```
                 struct ChunkyPort

                 struct ColoursCP
               And yes - if you were wondering by looking at the examples in this
document - I am a huge Simpsons fan :)
```

1.2 chunky.library Licensing Information

```
UPGRADING FROM 3.0?  Please read the seperate CHANGES_3_0 file.

NOTE WELL: RECOMPILATIONS OF chunky.library ARE NOT ALLOWED TO BE
DISTRIBUTED!  Rosande Limited can only recompile chunky.library.

chunky.library is copyright (c) 1999 Rosande Limited, all rights reserved.

The source code is free, and copyright remains with the authors of any
submissions or changes.  Documentation is to remain unmodified.  Source
code is available by e-mailing me (if you are a non-commerical user).

FREEWARE, SHAREWARE, AND PUBLIC DOMAIN AUTHORS
----------------------------------------------

You are allowed to redistribute the ORIGINAL UNMODIFIED chunky.library with
your software.  You are not under requirement to redistribute the library
or the documentation, or mention that your software uses chunky.library.
You do not need to credit us, or pay any money, or give us anything in
return..  unless you're feeling generous.

COMMERCIAL SOFTWARE AUTHORS
---------------------------

Redistribution of chunky.library, and commercial software which uses parts
of chunky.library itself or its source code which is copyright to Rosande
Limited requires authorisation from Rosande Limited.

Please see ComLicense.txt for more information and the form to complete,
sign and return.
```

1.3 chunky.library Submissions and Contributors

```
The current list of contributors currently looks like this:

- James L Boyd <jamesboyd@all-hail.freeserve.co.uk>
  Blitz Basic 2 support
  Documentation nagging :)

- Stephan Rupprecht <stephan.rupprecht@primus-online.de>
  c2p/p2c code cleanups

- Petri Koistinen <pkoistin@hit.fi>
  Reminding me to release an update :)
```

1.4 chk_initchunky

```
                NAME

  CHK_InitChunky -- Creates a chunky buffer

SYNOPSIS


              struct ChunkyPort *
               CHK_InitChunky(UWORD Width, UWORD Height)
                                           d0            d1

FUNCTION

  Allocates an area of memory (preferably Fast RAM) for a chunky buffer.
  The buffer is cleared to nothing, and parts of the ChunkyPort are set
  up to default values.

INPUTS

  Width -- the pixel width of the buffer.  Must be greater than zero.
  Height -- the pixel height of the buffer.  Must be greater than zero.

RESULT

  Returns a ChunkyPort structure or NULL if the memory couldn't be
  allocated.

EXAMPLE

  ...

              struct ChunkyPort *
              CP = NULL;

  if(CP = CHK_InitChunky(320, 200)) // allocates a 320x200 chunky buffer
  {
      CHK_FreeChunky(CP);
  }
  ...

NOTES

  The default values of the ChunkyPort are set thus:

  cp_Flags = JAM1
  cp_APen = 1

  Other values are set to zero or NULL.

BUGS

SEE ALSO

  CHK_FreeChunky(), CHK_InitColours()
```

1.5 chk_initcolours

```
                NAME

  CHK_InitColours -- allocate a colour table

SYNOPSIS


              struct ColoursCP *
               CHK_InitColours(
              struct ChunkyPort *
              CP)
                                                A0

FUNCTION

  ChunkyPort's can have individual colour palettes.  When they are
  originally allocated, they do not have a palette, but CHK_InitColours
  sets up a colour table and attaches it to a ChunkyPort.

INPUTS

  CP -- pointer to a valid ChunkyPort

RESULT

  Returns a pointer to the allocated colour table if the ChunkyPort did
  not have a colour table, or a pointer to the colour table the
  ChunkyPort already has if there is one.

  NULL is returned if the allocation failed.

EXAMPLE

  ...
  struct ColoursCP *Cols;
  struct ChunkyPort *CP;

  if(CP = CHK_InitChunky(100, 100))
  {
      if(Cols = CHK_InitColours(CP)) // allocate a colour table for CP
      {
          ...
      }
      CHK_FreeChunky(CP);
  }
  ...

NOTES

  Colour tables are only really useful if you are reading/writing
  ChunkyPort buffers to and from disk.  Maybe they could have other uses,
```

```
  but I'm not sure at the moment.

  When a colour table is attached to a ChunkyPort, the colour table is
  freed when the ChunkyPort is freed.  DO NOT free it yourself!

  Allocated colour tables are all set to black and always have 256
  colours.

BUGS

SEE ALSO

  CHK_FreeChunky()
```

1.6 chk_freechunky

```
                NAME

  CHK_FreeChunky -- frees a chunky buffer

SYNOPSIS

  void CHK_FreeChunky(
              struct ChunkyPort *
              CP)
                               A0

FUNCTION

  Frees up memory allocated by CHK_InitChunky(), thus removing the
  ChunkyPort from memory.  Any additional buffers, such as colour tables,
  are also freed.

  However, if the ChunkyPort was allocated using CHK_GetChunkyPort() and
  the buffer was NOT copied, this routine will NOT free the ChunkyPort
  meaning your application MUST free the memory yourself.  See
  CHK_GetChunkyPort() for more information.

INPUTS

  CP -- pointer to a valid ChunkyPort.  Passing NULL is safe.

BUGS

  If you pass a pointer to an area of memory which isn't a ChunkyPort or
  is a "corrupted" ChunkyPort, this will probably lead to a crash.

SEE ALSO

  CHK_InitChunky(), CHK_GetChunkyPort()
```

1.7 chk_drawchunky

```
                NAME

  CHK_DrawChunky -- draws an entire chunky buffer onto a RastPort

SYNOPSIS

  void CHK_DrawChunky(
             struct ChunkyPort *
             CP, struct RastPort *rp, UWORD x, UWORD y)
                                A0                  A1            D0      D1

FUNCTION

  As CHK_DrawChunkyArea(), except this routine will draw the entire
  ChunkyPort instead of part.

  See CHK_DrawChunkyArea() for more information.

INPUTS

  CP -- pointer to a valid ChunkyPort
  rp -- pointer to a valid RastPort
  x -- the x-coord in the RastPort to start drawing to
  y -- the y-coord in the RastPort to start drawing to

SEE ALSO

  CHK_DrawChunkyArea()
```

1.8 chk_drawchunkyarea

```
                NAME

  CHK_DrawChunkyArea -- draws part of a chunky buffer onto a RastPort

SYNOPSIS

  void CHK_DrawChunkyArea(
             struct ChunkyPort *
             CP, struct RastPort *rp, UWORD x, UWORD y,
                                 A0                  A1            D0  ↩
                                                  D1
                           UWORD Width, UWORD Height)
                                D2          D3

FUNCTION

  Places the contents of the supplied chunky buffer onto the specified
  RastPort.  It will destroy whatever is currently on the RastPort with
  the chunky buffer from x,y to (x+width),(y+height).
```

```
  It draws from x,y on the RastPort (but always draws from 0,0 in the
  chunky buffer), using the specified width and height.

  THIS FUNCTION HAS BEEN FIXED when used under 3.0 and CGFX.  (v3.1+)

INPUTS

  CP -- pointer to a valid ChunkyPort to draw
  rp -- pointer to a RastPort to draw onto
  x -- the x-coord to start drawing to on the RastPort
  y -- the y-coord to start drawing to on the RastPort
  Width -- the width of the ChunkyPort to draw (must be <= the width of
           the chunky buffer)
  Height -- the height of the ChunkyPort to draw (must be <= the height
            of the buffer)

EXAMPLE

  ...
  struct ChunkyPort *CP;
  struct Window *Window;

  if(CP = CHK_InitChunky(100, 100))
  {
      if(Window = OpenWindowTagList(NULL, NULL))
      {
          // Draw a rectangle on the chunky buffer in colour 3
          CHK_SetAPen(CP, 3);
          CHK_DrawRect(CP, 10, 10, 30, 30);
          // Draw half of the ChunkyPort onto the window at 20,20
          CHK_DrawChunkyArea(CP, Window->RPort, 20, 20, 50, 50);
          ...
          CloseWindow(Window);
      }
      CHK_FreeChunky(CP);
  }
  ...

NOTES

  Colour 0 is always drawn with this function.  Use CHK_InsertChunky() to
  not draw colour 0.

  When used with AGA, several things are not checked to speed up the
  process.  As a bit of a hack, the c2p code draws the chunky information
  direct onto the RastPort's BitMap, thus completely ignoring any
  clipping and such.  If you draw the chunky buffer outside of the
  BitMap, you will corrupt memory - there is NO clipping, and there will
  never be (it's too slow otherwise).

  Also, on AGA, if you are drawing on a window which is, for example,
  overlapping another window, the c2p does not respect overlapping and
  therefore will draw part of the chunky buffer on the window with is on
  top.

  None of these AGA problems occur when used with Cybergraphics because
  the routines obey these restrictions.
```

```
  Drawing to an x,y with is below zero is not supported.  Do not pass a
  width or height of zero.

  Drawing onto an "invisible" RastPort (e.g.  a RastPort that consists of
  a BitMap and is never displayed) is not advised.

BUGS

SEE ALSO

  CHK_DrawChunky(), CHK_InsertChunky()
```

1.9 chk_insertchunky

```
               NAME

  CHK_InsertChunky -- draws a chunky buffer into a RastPort preserving
                      the background

SYNOPSIS

  void CHK_InsertChunky(
              struct ChunkyPort *
              CP, struct RastPort *rp, UWORD x, UWORD y)
                                  A0                   A1             D0          ↩
                                     D1

FUNCTION

  This routine is much like CHK_DrawChunky(), except that any chunky
  pixels in the ChunkyPort that use colour zero will NOT get drawn on the
  RastPort.

  The advantage of this is that most of the time, colour zero is used as
  a "background" colour and is usually not required to be transferred to
  the display.  Therefore this routine ignores all chunky pixels that are
  colour zero, preserving the background of the RastPort, whilst drawing
  on to the "foreground".

INPUTS

  CP -- pointer to a valid ChunkyPort
  rp -- pointer to a valid RastPort
  x -- the x-coord to start drawing to in the RastPort
  y -- the y-coord to start drawing to in the RastPort

NOTES

  This routine requires to make a temporary buffer of the same dimensions
  as the ChunkyPort you are drawing, therefore have memory free to allow
  this routine to run.

  It is slightly slower than CHK_DrawChunky(), but not significantly.
```

```
  Just don't use it for huge drawings.

  Similar restrictions for AGA use apply with CHK_InsertChunky() as
  CHK_DrawChunkyArea() - because this routine calls CHK_DrawChunkyArea().

BUGS

  CHK_InsertChunkyArea() is missing :)

SEE ALSO

  CHK_DrawChunkyArea()
```

1.10 chk_createchunkyfrombitmap

```
                NAME

  CHK_CreateChunkyFromBitMap -- make a ChunkyPort using a BitMap

SYNOPSIS


              struct ChunkyPort *
               CHK_CreateChunkyFromBitMap(struct BitMap *bm, UWORD x, UWORD y,
                                                      A0            D0        ↩
                                                         D1
                                                 UWORD width, UWORD height)
                                                       D2          D3

FUNCTION

  Using the supplied BitMap, this routine creates a ChunkyPort out of a
  BitMap.  As far as the image is concerned it is an exact copy, except
  that the image is chunky.  You can then use all of the chunky.library
  routines with the new ChunkyPort.

INPUTS

  bm -- pointer to a valid BitMap
  x -- x-coord to grab from on the BitMap
  y -- y-coord to grab from on the BitMap
  width -- width of the rectangle to grab from the BitMap
  height -- height of the rectangle to grab from the BitMap

RESULT

  Returns a ChunkyPort with its buffer consisting of a copy of the
  BitMap's pixels or NULL if the BitMap couldn't be converted for some
  reason.

EXAMPLE

  ...
  struct ChunkyPort *cp;
```

```
  struct BitMap *bm;

  if(bm = AllocBitMap(320, 200, 8, BMF_CLEAR, NULL))
  {
      ...
      // Convert the bitmap into chunky
      if(cp = CHK_CreateChunkyFromBitMap(bm, 0, 0, 320, 200))
      {
          ...
          CHK_FreeChunky(cp);
      }
      FreeBitMap(bm);
  }
  ...
```

NOTES

Once the BitMap has been converted into a ChunkyPort, you no longer need to keep the BitMap if you don't need it.

Passing a width or height of zero is daft.

BUGS

SEE ALSO

CHK_CreateChunkyFromRastPort(), CHK_InitChunky()

1.11 chk_createchunkyfromrastport

NAME

CHK_CreateChunkyFromRastPort -- make a ChunkyPort from a RastPort

SYNOPSIS

```
struct ChunkyPort *
 CHK_CreateChunkyFromRastPort(struct RastPort *rp, UWORD x, UWORD  ↩
    y,
                                                      A0                D0  ↩
                                                                D1
                                                UWORD width, UWORD height)
                                                       D2          D3
```

FUNCTION

Exactly the same as CHK_CreateChunkyFromBitMap() except this routine uses a RastPort instead of a BitMap.

See CHK_CreateChunkyFromBitMap() for more information.

INPUTS

```
  rp -- pointer to a valid RastPort
  x -- x-coord to grab from on the RastPort
  y -- y-coord to grab from on the RastPort
  width -- width of the rectangle to grab from the RastPort
  height -- height of the rectangle to grab from the RastPort

RESULT

  A copy of the specified RastPort, except as a ChunkyPort - or NULL for
  a failure.

SEE ALSO

  CHK_CreateChunkyFromBitMap()
```

1.12 chk_setdrmd

```
                NAME

  CHK_SetDrMd -- set the drawing mode for a ChunkyPort

SYNOPSIS

  void CHK_SetDrMd(
              struct ChunkyPort *
              cp, UWORD flags)
                              A0                D0

FUNCTION

  Sets the drawing mode that is applied when a fill, line or text is
  drawn on the specified ChunkyPort.

  Flag definitions are the same for graphics.library/SetDrMd() so see
  graphics/rastport.h for the bit definitions.

INPUTS

  cp -- the ChunkyPort to alter
  flags --  an OR'd string of flags

EXAMPLE

  #include <graphics/rastport.h>
  ...
  struct ChunkyPort *cp;
  ...
  if(cp = CHK_InitChunky(100, 100))
  {
      // Use JAM2 drawing instead of JAM1 (the default)
      CHK_SetDrMd(cp, JAM2);
      ...
      CHK_FreeChunky(cp);
  }
```

```
  ...

NOTES

  Not all combinations work.  JAM1 and JAM2 are supported in all
  functions, although INVERSVID is also supported with text functions.
  Other flags are ignored.

BUGS

  Using more than one flag is not currently supported (e.g.
  JAM1 | INVERSVID).

SEE ALSO

  graphics.library/SetDrMd()
```

1.13 chk_setapen

```
                NAME

  CHK_SetAPen -- Set the primary pen for a ChunkyPort.

SYNOPSIS

  void CHK_SetAPen(
              struct ChunkyPort *
              cp, UBYTE pen)
                              A0              D0

FUNCTION

  Sets the foreground pen which is used to draw lines, fills and text.

INPUTS

  cp -- pointer to a valid ChunkyPort
  pen -- a colour index from 0-255

EXAMPLE

  ..
  CHK_SetAPen(cp, 1); // use colour 1
  CHK_DrawRect(cp, 20, 20, 100, 100); // draw a rectangle in colour 1
  CHK_SetAPen(cp, 2); // use colour 2
  CHK_DrawRect(cp, 30, 30, 100, 100); // draw an overlapping rectangle in colour ↩
      2
  ...

SEE ALSO

  graphics.library/SetAPen(), CHK_SetOPen(), CHK_SetDrMd(),
  CHK_SetABOPen()
```

1.14 chk_setopen

```
                NAME

  CHK_SetOPen -- Set the outline pen for a ChunkyPort.

SYNOPSIS

  void CHK_SetOPen(
             struct ChunkyPort *
             cp, UBYTE pen)
                             A0              D0

FUNCTION

  Sets the outline pen which is used to draw outlines and embossments on
  text.

INPUTS

  cp -- pointer to a valid ChunkyPort
  pen -- a colour index from 0-255

NOTES

  The outline pen is only used when you wish to render text using some
  special styles exclusive to chunky.library's text rendering functions.
  See CHK_SetSoftStyle() for more information on these new styles.

SEE ALSO

  CHK_SetAPen(), CHK_SetDrMd(), CHK_SetABOPen()
```

1.15 chk_setabopen

```
                NAME

  CHK_SetABOPen -- Sets all three pens in a ChunkyPort at once.

SYNOPSIS

  void CHK_SetABOPen(
             struct ChunkyPort *
             cp, UBYTE a, UBYTE b, UBYTE o)
                              A0              D0       D1       D2

FUNCTION

  Changes the foreground pen, background pen and outline pen at the same
```

```
  time.

  This routine is really only useful when rendering text, since
  background and outline pens are only used with text.

INPUTS

  cp -- pointer to a valid ChunkyPort
  a -- the new colour index for the foreground pen (0-255)
  b -- new colour for the background pen (0-255)
  o -- new colour for the outline pen (0-255)

SEE ALSO

  CHK_SetAPen(), CHK_SetOPen()
```

1.16 chk_move

```
                NAME

  CHK_Move -- Moves a ChunkyPort's graphics pen position.

SYNOPSIS

  void CHK_Move(
              struct ChunkyPort *
              cp, UWORD x, UWORD y)
                         A0              D0       D1

FUNCTION

  Moves the graphics pen to x,y relative from 0,0 in the ChunkyPort.
  This is the starting point for all operations such as CHK_Draw() and
  CHK_Text().

INPUTS

  cp -- pointer to a valid ChunkyPort
  x,y -- new coords for graphics pen

SEE ALSO

  CHK_Draw(), CHK_Text(), graphics.library/Move()
```

1.17 chk_writepixel

```
                NAME

  CHK_WritePixel -- Sets a single "pixel" in a ChunkyPort to a specific
                    pen
```

```
SYNOPSIS

  void CHK_WritePixel(
              struct ChunkyPort *
              cp, UWORD pen, UWORD x, UWORD y)
                                A0              D0        D1       D2

FUNCTION

  Changes a single point in a ChunkyPort to the specified pen.  Does
  pretty much the same thing as graphics.library/WritePixel().

INPUTS

  cp -- pointer to a valid ChunkyPort (NULL is safe to pass)
  pen -- the colour index to use (0-255)
  x,y -- the point in the ChunkyPort to change

NOTES

  If x,y is outside of the ChunkyPort or cp is NULL, this function does
  nothing.

SEE ALSO

  graphics.library/WritePixel(), CHK_ReadPixel()
```

1.18 chk_readpixel

```
                NAME

  CHK_ReadPixel -- Sees what colour index a "pixel" uses in a ChunkyPort

SYNOPSIS

  BYTE CHK_ReadPixel(
              struct ChunkyPort *
              cp, UWORD x, UWORD y)
                                A0              D0       D1

FUNCTION

  Enquires a position in a ChunkyPort to see what colour it is.

INPUTS

  cp -- pointer to a valid ChunkyPort (NULL is safe)
  x,y -- the point in the ChunkyPort you want to examine

RESULT

  Returns the colour index of the specified point.
  Returns -1 if x,y is outside the buffer or cp is NULL.
```

```
SEE ALSO

  graphics.library/ReadPixel(), CHK_WritePixel()
```

1.19 chk_draw

```
                NAME

  CHK_Draw -- Draws a line in a ChunkyPort

SYNOPSIS

  void CHK_Draw(
              struct ChunkyPort *
              cp, UWORD x2, UWORD y2)
                           A0             D0        D1

FUNCTION

  Draws a line from the position set with CHK_Move(), to x2,y2.

INPUTS

  cp -- pointer to a valid ChunkyPort
  x2,y2 -- the point where the line should be drawn to

RESULT

  A line is drawn from x1,y1,x2,y2.

EXAMPLE

  ...
  CHK_SetAPen(cp, 8);  // draw the line in colour 8
  CHK_Move(cp, 100, 100); // move to 100,100
  CHK_Draw(cp, 200, 200); // draw the line from 100,100 to 200,200
  ...

BUGS

  This routine has not been tested since this library was compiled with
  SAS/C.

  If x2,y2 are outside of the chunky buffer, it WILL overwrite memory
  outside of the buffer.  This will be fixed sometime.

SEE ALSO

  CHK_Move()
```

1.20 chk_drawline

```
                NAME

  CHK_DrawLine -- draws a line in a ChunkyPort

SYNOPSIS

  void CHK_DrawLine(
             struct ChunkyPort *
             cp, UWORD x1, UWORD y1, UWORD x2, UWORD y2)
                              A0            D0        D1        D2        D3

FUNCTION

  Shorthand for CHK_Move(cp, x1, y1); CHK_Draw(cp, x2, y2);.

INPUTS

  cp -- a ChunkyPort
  x1,y1,x2,y2 -- coords to draw from and to

NOTES

  No clipping with this function.

SEE ALSO

  CHK_Move(), CHK_Draw()
```

1.21 chk_drawrect

```
                NAME

  CHK_DrawRect -- draws a rectangle on a ChunkyPort

SYNOPSIS

  void CHK_DrawRect(
             struct ChunkyPort *
             cp, UWORD x1, UWORD y1, UWORD x2, UWORD y2)
                              A0            D0        D1        D2        D3

FUNCTION

  Draws a hollow rectangle onto a ChunkyPort.  Hollow means that the
  rectangle is not filled.

INPUTS

  cp -- a ChunkyPort
  x1,y1,x2,y2 -- coords to draw from and to
```

```
NOTES

  No clipping with this function.

SEE ALSO

  CHK_RectFill()
```

1.22 chk_rectfill

```
                NAME

  CHK_RectFill -- draws a filled rectangle on a ChunkyPort

SYNOPSIS

  void CHK_RectFill(
             struct ChunkyPort *
             cp, UWORD x1, UWORD y1, UWORD x2, UWORD y2)
                                A0            D0        D1        D2        D3
FUNCTION

  Draws a filled rectangle on a ChunkyPort.  This function acts exactly
  the same as graphics.library/RectFill().

INPUTS

  cp -- a ChunkyPort
  x1,y1,x2,y2 -- coords to draw from and to

NOTES

  No clipping with this function.

BUGS

  Could be a bit slow, but I hardly think it's noticeable.

SEE ALSO

  CHK_DrawRect(), graphics.library/RectFill()
```

1.23 chk_drawellipse

```
                NAME

  CHK_DrawEllipse -- draws an ellipse on a ChunkyPort

SYNOPSIS
```

```
  void CHK_DrawEllipse(
              struct ChunkyPort *
              cp, UWORD cx, UWORD cy, UWORD rx, UWORD ry)
                                A0              D0        D1        D2       D3

FUNCTION

  Draws an ellipse (i.e.  a squashed circle) on a ChunkyPort.  Pretty
  much the same as graphics.library/DrawEllipse().

INPUTS

  cp -- a ChunkyPort
  cx,cy -- centre positions of the ellipse
  rx,ry -- radii of the ellipse on the x-coord and the y-coord

NOTES

  If rx or ry is NULL this function does nothing (although cp MUST NOT be
  NULL).

  No clipping with this function.

BUGS

  Should work :)

SEE ALSO

  graphics.library/DrawEllipse()
```

1.24 chk_setrast

```
                NAME

  CHK_SetRast -- sets an entire ChunkyPort to a specified colour index.

SYNOPSIS

  void CHK_SetRast(
              struct ChunkyPort *
              cp, UBYTE pen)
                             A0              D0

FUNCTION

  Basically fills in the entire ChunkyPort to a colour.  Anything that is
  in the ChunkyPort will get drawn over.

INPUTS

  cp -- pointer to a ChunkyPort
  pen -- the colour index to fill with (0-255)
```

```
RESULT

  A cleared ChunkyPort.

NOTES

  This function's called SetRast because it does the same as graphics.library/ ↩
     SetRast() -
  obviously :)

SEE ALSO

  graphics.library/SetRast(), CHK_RectFill()
```

1.25 chk_setfont

```
                NAME

  CHK_SetFont -- changes the font used on a ChunkyPort

SYNOPSIS

  void CHK_SetFont(
              struct ChunkyPort *
              cp, struct TextFont *tf)
                            A0                      A1

FUNCTION

  Because anything needs to be in chunky format before it can be applied
  to a chunky buffer, we need to convert a standard Amiga font into
  chunky to be able to draw the characters onto the buffer.  Since
  standard Amiga fonts are actually 2 colour BitMaps, it's just a case of
  converting it into chunky.

  And since every ChunkyPort can have a different font, this
  chunky-converted font is actually a buffer assigned to each ChunkyPort.
  Meaning you cannot CHK_SetFont() the same font to more than one
  ChunkyPort at a time - the font needs to be CHK_SetFont()'d for each
  ChunkyPort.  This might seem like a waste of memory, but no font buffer
  is ever over 8K (small price to pay for flexibility I think you'll
  agree).

  Any call to a text rendering function will fail if this routine has not
  been called previously - it only needs to be called once until the
  ChunkyPort is freed.  Of course, it needs to be called again if you
  wish to use different fonts on the same ChunkyPort.

INPUTS

  cp -- pointer to a valid ChunkyPort
  tf -- pointer to a previously opened/obtained TextFont structure or
        NULL
```

RESULT

A font is set up for the ChunkyPort. CHK_Text, etc. can now be called. If the font creation failed, text functions will do nothing.

EXAMPLE

```
...
struct TextFont *tf;
struct ChunkyPort *cp;
struct TextAttr topaz80 = {"topaz.font", 8, NULL, NULL};
...

// Allocate a chunky buffer
if(cp = CHK_InitChunky(320, 200))
{
    // Open the font using the TextAttr structure (topaz.font is always in ↩
       memory)
    if(tf = OpenFont(&topaz80))  // graphics.library function
    {
        // Attach the font to the ChunkyPort
        CHK_SetFont(cp, tf);
        // Draw some text
        CHK_Move(cp, 40, 40);
        CHK_Text(cp, "Eat my shorts!");
        CHK_Move(cp, 100, 100);
        CHK_Text(cp, "Don't have a cow, man!");
        CHK_SetAPen(cp, 10);
        CHK_TextCentre(cp, "El Barto!", 150);
    }
}
if(tf) CloseFont(tf);
if(cp) CHK_FreeChunky(cp);
...
```

NOTES

If you call CHK_SetFont() with the same TextFont in a previous call, this function does nothing. Also if TextFont is NULL, nothing happens.

You do not need to free the ChunkyPort if you which to change the font. Just CHK_SetFont() the ChunkyPort with the new font.

You do not need the TextFont any more after a call to CHK_SetFont().

You CANNOT use non-proportional (varying width) fonts. Only fixed-width fonts (e.g. topaz.font/8) are supported. Using non-proportional fonts will print garbage - you have been warned.

Colour fonts and non-BitMap Compugraphic fonts are not supported.

BUGS

No non-proportional fonts - this may be fixed in the future (although it's not important).

SEE ALSO

```
  CHK_SetSoftStyle(), graphics.library/OpenFont(), CHK_Text()
```

1.26 chk_setsoftstyle

```
                NAME

  CHK_SetSoftStyle -- changes the style to draw text in

SYNOPSIS

  ULONG CHK_SetSoftStyle(
             struct ChunkyPort *
             cp, ULONG styles)
                                A0               D0

FUNCTION

  Text can be drawn in different styles to show emphasis, etc.  Several
  styles are supported, and can be combined.  These are:

  FS_NORMAL    - no style
  FSF_BOLD     - apply bold to the text
  FSF_3D       - give the text a 3D-effect (try it and see)
  FSF_WIDE3D   - as FSF_3D except nearly twice the width
  FSF_OUTLINE  - draw the text normal but give the text an outline
  FSF_EMBOSSED - emboss the text

  These settings will apply to all text you draw on the ChunkyPort until
  it is changed, or the font is changed.

INPUTS

  cp -- pointer to a valid ChunkyPort
  styles -- flags to apply (listed above)

RESULT

  Returns the styles applied.

EXAMPLE

  ...
  // Use embossed text
  CHK_SetSoftStyle(cp, FSF_EMBOSSED);
  // Use bold and an outline
  CHK_SetSoftStyle(cp, FSF_BOLD | FSF_OUTLINE);
  ...

NOTES

  Some combinations are pointless (e.g.  FSF_3D | FSF_WIDE3D).

  FSF_UNDERLINED and FSF_ITALIC are not supported.
```

```
  Normal styles are defined in graphics/text.h and the new ones are in
  chunky.h.

SEE ALSO

  CHK_SetFont(), CHK_Text(), chunky.h
```

1.27 chk_textlength

```
                NAME

  CHK_TextLength -- figures out the pixel width of some text

SYNOPSIS

  LONG CHK_TextLength(
             struct ChunkyPort *
             cp, STRPTR text, WORD length)
                               A0               A1            D0

FUNCTION

  Using the font which is attached to a ChunkyPort, this routine
  calculates the pixel width of the text specified.

INPUTS

  cp -- pointer to a valid ChunkyPort with a font set with CHK_SetFont()
  text -- the string you want the width of
  length -- the character length of text (strlen(text) for example)

RESULT

  The pixel width of the text or NULL if failed.

EXAMPLE

  ...
  struct ChunkyPort *cp;
  unsigned char *text = "Ay Caramba!";
  long l;
  ...

  l = CHK_TextLength(cp, text, strlen(text));
  printf("The pixel width of the text is %ld.", l);
  ...

NOTES

  If length is zero or the ChunkyPort has no font, this returns NULL.

SEE ALSO
```

```
  graphics.library/TextLength(), CHK_SetFont()
```

1.28 chk_text

```
                NAME

  CHK_Text -- draw text characters in a ChunkyPort

SYNOPSIS

  void CHK_Text(
              struct ChunkyPort *
              cp, STRPTR text)
                           A0                D0

FUNCTION

  Outputs the relevant characters specified in the text string to a
  ChunkyPort using the pre-defined font.

  The graphics pen position in the ChunkyPort is not changed.

INPUTS

  cp -- a pointer to a valid ChunkyPort with font
  text -- the string to draw

NOTES

  If text is NULL, nothing happens.  If no font has been set, nothing
  happens.

BUGS

  The graphics pen position may be moved two or three pixels if any style
  is used (not so for FS_NORMAL draws).

  Text is not clipped if it overruns the buffer - be warned.

SEE ALSO

  CHK_SetFont(), CHK_TextCentre(), graphics.library/Text()
```

1.29 chk_textcentre

```
                NAME

  CHK_TextCentre -- draws some text centred on the x-coord

SYNOPSIS
```

```
  void CHK_TextCentre(
              struct ChunkyPort *
              cp, STRPTR text, UWORD y)
                               A0               A1          D0

FUNCTION

  As CHK_Text(), except the text is automatically centred on the
  x-coordinate.

  The graphics pen position in the ChunkyPort is changed to the top-left
  corner of where the text is to be drawn.

INPUTS

  cp -- pointer to a valid ChunkyPort with font
  text -- the string to print
  y -- the y-coord to draw on

NOTES

  This routine DOES clip the text on the x-coord and width, unlike
  CHK_Text().

BUGS

  Clipping here should be in CHK_Text().

SEE ALSO

  CHK_Text(), CHK_SetFont()
```

1.30 chk_choosehardwaremode

```
NAME

  CHK_ChooseHardwareMode -- allows chunky.library to decide to use CGFX
                            or AGA

SYNOPSIS

  void CHK_ChooseHardwareMode(ULONG ModeID)
                                    D0

FUNCTION

  This important function determines what drawing method should be used
  for blasting chunky data onto the display.

  It looks through the display driver database to see if the specified
  screen mode is native to the Amiga (e.g.  AGA) or on a graphics board
  (e.g.  Cybergraphics).
```

```
  This routine MUST be called as soon as you open a screen or viewport
  before you draw any chunky information onto it (using CHK_DrawChunky(),
  etc.).

INPUTS

  ModeID -- the 32-bit display mode ID number used when you open a
            screen/viewport

EXAMPLE

  ...
  struct Screen *Screen;
  ULONG ModeID = (PAL_MONITOR_ID | HIRES_KEY);  // == 0x29000 (AGA)
  // ULONG ModeID = 0x50011000;                 // == Picasso96: 640x480x256 ( ↩
     GFX card)
  ...

  if(Screen = OpenScreenTags(NULL, SA_LikeWorkbench, TRUE,
                                   SA_DisplayID, ModeID,
                                   TAG_DONE))
  {
      // Determine the best hardware method (this will use AGA)
      CHK_ChooseHardwareMode(ModeID);
      ...
      // Free to use drawing routines now without worries of retargetting
      CHK_DrawChunky(cp, rp, 0, 0);
      ...
  }
  if(Screen) CloseScreen(Screen);
  ...

  // Using either of the two ModeID's shown in this example requires NO change  ↩
     to any
  // code.  That's just how flexible chunky.library is :)  It's easy to just use ↩
      a screen
  // mode requester and not care about what screen mode is used
  // -- RTG or AGA.. who cares?  :)

NOTES

  If a screen mode ID that is passed is not recognised by
  cybergraphics.library/IsCyberModeID() or cybergraphics.library couldn't
  be opened, this routine FORCES to use custom c2p code (which only works
  on AGA - it will crash if used on a graphics board).

  Not calling this routine at all in your program will make
  chunky.library ALWAYS use custom c2p code making your program AGA only
  and not RTG compatible.  This routine MUST be called!

BUGS

  Although chunky.library does not support Picasso96 direct, P96's
  emulation of cybergraphics.library is suitable.

  Maybe this should double-check to see if it really is safe to use
  custom c2p code if Cybergraphics does not recognise the mode.
```

```
SEE ALSO

  A good book on c2p (if there is one :).
```

1.31 chk_drawchunkychunkyarea

```
                NAME

  CHK_DrawChunkyChunkyArea -- copy data from one ChunkyPort to another

SYNOPSIS

  void CHK_DrawChunkyChunkyArea(
              struct ChunkyPort *
              Dest,
              struct ChunkyPort *
              Src,
                                          A0                          A1
             UWORD DestX, UWORD DestY, UWORD SrcX, UWORD SrcY, UWORD Width,  ↩
                UWORD Height)
                  D0           D1           D2          D3          D4  ↩
                               D5

FUNCTION

  This routine is like CHK_DrawChunkyArea(), except it does not draw from
  ChunkyPort to RastPort.  It draws from ChunkyPort to ChunkyPort instead
  - useful for off-screen rendering.

  The theory behind this routine is as follows:

  1.  Allocate two BitMap's with RastPort's for double-buffering.
  2.  Allocate a ChunkyPort to use as a go-between buffer.
  3.  In the main redraw routine, CHK_DrawChunkyChunkyArea all the
      ChunkyPort's you want to display onto the go-between buffer.
  4.  CHK_DrawChunky() the go-between buffer onto one of the RastPort's.
  5.  Swap buffers and loop from step 3.

  That's the main point of this function, although I'm sure there are
  other uses for it.

INPUTS

  Dest -- the destination ChunkyPort to copy to
  Src -- the source ChunkyPort to copy from
  DestX,DestY -- where to copy to in the destination
  SrcX,SrcY,Width,Height -- the area of the rectangle to copy from the
                            source

EXAMPLE

  ...
  struct ChunkyPort *src, *dest;
  ...
```

```
  if((src = CHK_InitChunky(320, 200)) && (dest = CHK_InitChunky(320, 200)))
  {
      // Got two buffers.. draw a box on one...
      CHK_DrawRect(src, 20, 20, 100, 100);
      // And copy that box to the other at a different position
      CHK_DrawChunkyChunkyArea(dest, src, 40, 40, 20, 20, 100, 100);
      ...
  }
  if(dest) CHK_FreeChunky(dest);
  if(src)  CHK_FreeChunky(src);
  ...

NOTES

  If the destination buffer is too small to recieve the copy or the
  coordinates don't agree, this routine does nothing.

  Copying data to the same buffer has not been tested - overlapping may
  have strange effects.

SEE ALSO

  CHK_DrawChunky(), CHK_DrawChunkyChunky()
```

1.32 chk_drawchunkychunky

```
                NAME

  CHK_DrawChunkyChunky -- copies the whole of one ChunkyPort to another

SYNOPSIS

  void CHK_DrawChunkyChunky(
              struct ChunkyPort *
              dest,
              struct ChunkyPort *
              src, UWORD x,
                                      A0                          A1              ↩
                                         D0
                           UWORD y)
                              D1

FUNCTION

  This is just a simple routine which copies an entire ChunkyPort into
  another at a specified coordinate.

  It is equalivant to doing
  CHK_DrawChunkyChunkyArea(dest, src, x, y, 0, 0, w, h).

INPUTS

  dest -- destination ChunkyPort to copy to
```

```
  src -- source ChunkyPort to copy from
  x,y -- coordinate in the destination to start copying to

SEE ALSO

  CHK_DrawChunkyChunkyArea()
```

1.33 chk_drawtransparentrectangle

```
                NAME

  CHK_DrawTransparentRectangle -- draws a filled "holy" rectangle into a
                                  ChunkyPort

SYNOPSIS

  void CHK_DrawTransparentRectangle(
              struct ChunkyPort *
              cp, UWORD x, UWORD y, UWORD w,
                                                 A0              D0       D1        ↩
                                                    D2
                                     UWORD h)
                                       D3

FUNCTION

  Draws a "transparent" rectangle.  It is not true-transparency, but
  gives the impression of such an effect.

  These rectangles are just every-after-one pixel is colour 0
  (transparent) and the others pixels are coloured to whatever the last
  CHK_SetAPen() colour was.

  Effective, but not truely transparent; graphics card owners could
  improve this to be so, or even die-hard AGA fanactics - but changing
  the palette is out of the scope of this library.  Or then again, maybe
  not :)

INPUTS

  cp -- ChunkyPort to render onto
  x,y,w,h -- box coordinates and sizes

NOTES

  w,h are clipped if they are too big for the ChunkyPort.

  The reason why I wrote this routine is that my custom GUI uses this to
  draw some boxes.  It's just here because it was in my original chunky
  incarnations and never bothered to remove it.  Still, some folk may
  find a practical use for it :)

SEE ALSO
```

```
  CHK_DrawRect(), CHK_RectFill(), any good 3D renderer to see a real
    transparent box :)
```

1.34 chk_getchunkyport

```
                NAME

  CHK_GetChunkyPort -- creates from/uses a block of memory as a
                       ChunkyPort

SYNOPSIS


              struct ChunkyPort *
               CHK_GetChunkyPort(APTR MemoryLocation, BOOL CopyBuf)

FUNCTION

  WARNING!!!  This function is very low-level and should only be used by
  people who know what they are doing (which always helps).  You
  shouldn't really know about this :)

  Using the supplied memory location, this routine "converts" or copies
  from this position a ChunkyPort.

  The main use of this function is that if raw ChunkyPort data is read
  from disk (or included as part of the executable), you can use this as
  a ChunkyPort - providing it is actually a ChunkyPort structure.

  It will expect the data to be like this in memory (NO GAPS):

   ..  an entire ChunkyPort structure (which is of sizeof(struct
       ChunkyPort) bytes) ..
   ..  the chunky buffer found in cp->cp_Chunky (which is of
       cp->cp_BufSize bytes) ..
   ..  optionally a ColoursCP structure (which is sizeof(struct
       ColoursCP) bytes) ..

  If the data is NOT organised in this way, you WILL get a crash.

  If CopyBuf is FALSE, you MUST NOT free the memory allocated at
  MemoryLocation UNTIL you have CHK_FreeChunky()'d the resulting
  ChunkyPort.  THEN you MUST free the data - chunky.library will NOT free
  this memory (although it will free any font buffers).

  If CopyBuf is TRUE, you can free the memory at MemoryLocation (if it is
  yours) as the routine will duplicate the memory block and create a
  ChunkyPort as if it were allocated using CHK_InitChunky().  You can
  then CHK_FreeChunky() the resulting ChunkyPort as normal.

  Please don't use this routine unless absolutely necessary or you have
  mastered this library and understand it fully.  Just use normal
  BitMap's and CHK_CreateChunkyFromBitMap() them - it's infinitely more
  safe.
```

```
INPUTS

  MemoryLocation -- pointer to somewhere in memory to make a ChunkyPort
                    from
  CopyBuf -- TRUE to copy the data, FALSE to reuse the same memory
             location (careful)

RESULT

  Returns a pointer to a ChunkyPort if successful, or NULL if the memory
  wasn't a ChunkyPort (as far as we could tell).

EXAMPLE

  Want an example?  Make sure you've understood the whole library first -
  then e-mail oondy@bigfoot.com.  This function is very advanced.

NOTES

  Don't come crying if you can't make this work.  It is very tricky.

BUGS

  None - if you do it right.

SEE ALSO

  The other commands :)
```

1.35 chk_putchunkycolours

```
                NAME

  CHK_PutChunkyColours -- copies attached colours in a ChunkyPort to a
                          ViewPort

SYNOPSIS

  BOOL CHK_PutChunkyColours(
              struct ChunkyPort *
              cp, struct ViewPort *vp)
                                    A0                    A1

FUNCTION

  Copies information from a ColoursCP structure in a ChunkyPort to a
  ViewPort immediately.

  This function is for advanced use only.

INPUTS

  cp -- pointer to a ChunkyPort with a valid ColoursCP structure
```

```
  vp -- pointer to a valid ViewPort

RESULT

  Returns TRUE/FALSE success.

NOTES

  chunky.library colour tables are rather private at the moment without
  functions to add, remove or change colours in the table.  Please don't
  use the ColoursCP structures until I write routines to support them -
  i.e.  don't roll your own methods (unless you submit them to include in
  a future release).

SEE ALSO

  CHK_InitColours()
```

1.36 chk_drawchunkytiled

```
                NAME

  CHK_DrawChunkyTiled -- tiles a ChunkyPort onto a RastPort

SYNOPSIS

  void CHK_DrawChunkyTiled(
              struct ChunkyPort *
              cp, struct RastPort *rp, UWORD x, UWORD y,
                                  A0                  A1            D0  ↩
                                              D1
                           UWORD w, UWORD h)
                             D2       D3

FUNCTION

  Draws a ChunkyPort over a RastPort using the specified rectangle.  If
  the ChunkyPort is smaller than the specified rectangle the graphic is
  drawn until it fits.  If the ChunkyPort is larger than the rectangle,
  it is clipped.

  Useful for drawing static backgrounds and textures, etc.

INPUTS

  cp -- ChunkyPort to draw
  rp -- RastPort to draw onto
  x,y,w,h -- rectangle dimensions to draw into on the RastPort

NOTES

  Colour zero is not made see-through.

BUGS
```

```
    Does not draw correctly width-ways and may even draw outside the buffer
    :(

    This is really a quick bodge-togethor, and doesn't really work.  Maybe
    fixed some time.

  SEE ALSO

    CHK_DrawChunky(), CHK_InsertChunky()
```

1.37 struct ChunkyPort

```
NOTE:  entries marked with ### are private and should not be touched.  All
entries are READ ONLY.

struct ChunkyPort
{
  unsigned long     cp_Identifier;  // == CHKP                                  ↩
     ###
  unsigned char    *cp_Chunky;      // Memory allocated for the chunky data
  unsigned short    cp_cx,          // Draw pen x position in buffer
                    cp_cy;          // Y pen
  unsigned char     cp_APen;        // Foreground pen
  unsigned char     cp_OPen;        // Outline pen
  unsigned char     cp_BPen;        // Background pen
  unsigned char     cp_IPen;        // Shine/shadow pen
  unsigned short    cp_Flags;       // Flags which are the same as graphics/ ↩
     rastport.h

  struct TextFont  *cp_Font;        // Font to use for text rendering
  unsigned char    *cp_TxtChunky;   // Chunky data buffer used for text rendering  ↩
     ###
  unsigned short    cp_TxHeight;    // Height of the text                          ↩
     ###
  unsigned short    cp_TxBaseline;  // Baseline of the text                        ↩
     ###
  unsigned short    cp_TxStyle;     // Softstyles in graphics/text.h plus these...
  unsigned short    cp_NoFree;      // TRUE if buffer was grabbed from a memory  ↩
     location ###
  unsigned long     cp_BufSize;     // Number of bytes used for cp_Chunky
  unsigned short    cp_Width,       // Pixel width of buffer
                    cp_Height;      // Pixel height of buffer
  struct ColoursCP *cp_Colours;     // Pointer to colour palete data or NULL if  ↩
     none ###
  unsigned long     cp_Reserved[7]; ###
};
```

1.38 struct ColoursCP

```
NOTE: These structures are PRIVATE AND READ ONLY!

struct cp_Colour32
{
  unsigned long R, G, B;
};

struct ColoursCP
{
  unsigned long      cpc_Identifier; // == CPCL
  short              cpc_ColourRecord;
  short              cpc_ColourFirst; //  = 0
  struct cp_Colour32 cpc_ColourTable32[256+1];
};
```

1.39 "

```
  NAME

  SYNOPSIS

  FUNCTION

  INPUTS

  RESULT

  EXAMPLE

  NOTES

  BUGS

  SEE ALSO
```

1.40 chk_c2poff

```
  NAME

    CHK_C2POff -- turns C2P off (v3.1)

  SYNOPSIS

    BOOL CHK_C2POff(void)

  FUNCTION

    Disables usage of the custom AGA C2P routines, forcing the library to
    use OS drawing routines.

    OBSOLETE!  DO NOT USE THIS FUNCTION!  This function MAY do something in
```

```
  the future so don't use it now.

RESULT

  As this function does nothing, this always returns zero.

BUGS

  Dead function - do not use.
```

1.41 chk_drawchunkywindowarea

```
NAME

  CHK_DrawChunkyWindowArea -- draws part of a ChunkyPort into a window
                              (V3.2+)

SYNOPSIS

  void CHK_DrawChunkyWindowArea(struct ChunkyPort *cp,
                                        A0
    struct Window *win, WORD x, WORD y, UWORD w, UWORD h)
           A1            D0      D1      D2       D3

FUNCTION

  This routine just saves you a bit of extra work when dealing with
  Windows and not directly RastPorts.

  It will calculate the offsets required to draw into a window that has
  a title bar and/or a border.  It will then simply call
  CHK_DrawChunkyArea().

INPUTS

  cp -- pointer to a valid ChunkyPort
  win -- pointer to an open Window with a RastPort
  x,y -- coords to draw at in the Window (may be negative)
  w,h -- width and height of cp to draw

NOTES

  Since this calls CHK_DrawChunkyArea() no extra clipping is done under
  AGA, so make sure your window is large enough, or clip yourself.

BUGS

  This routine could do so much more, but doesn't :)

SEE ALSO

  CHK_DrawChunkyWindow(), CHK_DrawChunkyArea()
```

1.42 chk_drawchunkywindow

```
NAME

  CHK_DrawChunkyWindow -- draws an entire ChunkyPort into a Window
                          (v3.2+)

SYNOPSIS

  void CHK_DrawChunkyWindow(struct ChunkyPort *cp, struct Window *win,
                                    A0                      A1
    WORD x, WORD y, UWORD w, UWORD h)
      D0      D1      D2       D3

FUNCTION

  As CHK_DrawChunky(), but is tailored for Windows and not RastPorts.
  See CHK_DrawChunkyWindowArea().

INPUTS

  cp -- pointer to a valid ChunkyPort
  win -- pointer to an open Window with a RastPort
  x,y -- coords to draw at in the Window (may be negative)

SEE ALSO

  CHK_DrawChunkyWindowArea()
```

1.43 chk_queryuseos

```
NAME

  CHK_QueryUseOS -- asks chunky.library how it's drawing stuff
                    (v3.2+)

SYNOPSIS

  BOOL CHK_QueryUseOS(void)

FUNCTION

  After your program has called CHK_ChooseHardwareMode() there is no way
  how your program can find out if chunky.library is drawing using AGA
  C2P or CGFX routines.  This function asks the library if it is using
  CGFX routines, returning TRUE if it is, and FALSE for AGA C2P.

RESULT

  TRUE if chunky.library is using CGFX...
  FALSE if it is using AGA C2P.

NOTES
```

```
Though your program does not need to know how chunky.library is drawing
its data on your screen, it can be useful to know sometimes.

Note that this result can only be correct once this routine has been
called.  It may change in the future.
```