Testfile

COLLABORATORS			
	TITLE : Testfile		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 26, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Testfile

1.1 Table Of Contents

rtgMaster.library AutoDOCS

LockRtgScreen

OpenClient

OpenRtgScreen

OpenServer

RtgAccept

RtgBlit

RtgBltClear

RtgClearPointer

RtgCloseFont

RtgInAdr

RtgInitRDCMP

RtgIoctl

RtgOpenFont

RtgRecv

RtgScreenAtFront

RtgScreenModeReq

RtgSend

RtgSetFont

RtgSetPointer

RtgGetMsg

RtgReplyMsg

RtgSetTextMode

RtgSetTextModeRGB

RtgText

RtgWaitRDCMP

RtgWaitTOF

RunServer

SetSegment

SwitchScreens

```
UnlockRtgScreen

WaitRtgBlit

WaitRtgSwitch

WriteRtgPixel

WriteRtgPixelArray

WriteRtgPixelRGB
```

1.2 rtgmaster.library/CallRtgC2P

```
              rtgmaster.library/CallRtgC2P

   NAME
        CallRtgC2P -- Perform c2p for Planar Screens, CopyRtgPixelArray for Chunky ←
            Screens

   SYNOPSIS
        int CallRtgC2P(RtgScreen,BufAdr,Array,signal,xpos,ypos,width,height,mode)
         D0              A0        A1      A2     D0      D1   D2   D3    D4      D5

        int CallRtgC2P(struct RtgScreen *,APTR,APTR,ULONG,ULONG,ULONG,ULONG,ULONG, ←
           ULONG)

   FUNCTION
        This function will look what the "standard c2p" for the system is up to  ←
           now
        (the standard c2p can be choose by a future version of the Rtgmaster  ←
           Screenmode
        requester, the available c2p algorithms are found in libs:rtgc2p, how own  ←
           c2p
        algorithms can be added to the system will be explained in the  ←
           documentation
        of the first version of rtgmaster.library that actually supports  ←
           CallRtgC2P).
        The function will, if the display is a Planar one, convert the Chunky Data ←
            in
        Array to Planar using the choosen c2p algorithm, and display it in the  ←
           choosen
        Buffer. For Chunky Displays it will instead do the same as  ←
           CopyRtgChunkyPixel.
        This way a very easy possibility to support both AGA and Graphics Boards  ←
           without
        having to do "special versions" will be available, if one uses a Fastram  ←
           Buffer.

        I am still looking for c2p algorithms for this function !!! All used c2p  ←
           algorithms
        should support AGA and additional, it would be fine, if they supported
        1x1,1x2,2x1 and 2x2. If you have fine c2p algorithms, mail me
```

```
      (MagicSN@birdland.es.bawue.de).

      NOTE : The Array should EXACTLY be as big as specified with Left, Top,  ↩
         Width
      and Height... it should *NOT* be bigger.

      NOTE: Currently you *HAVE TO* use xpos=0 ypos=0 width=<max x> height=<max  ↩
         y>
      Maybe this will change in the future !!!!!!!!!!!!!! This is only because i ↩
          do
      not have ANY c2p that supports that feature up to now...

      NOTE: Some c2p algorithms might do NOTHING in certain colour depths,  ↩
         chunky modes
      or for interleaved bitmaps. Be careful about this. If the c2p works,
      this function returns 0, otherwise an errorcode.

      Principially it COULD support 256, 64 (EHB), 32 or 16 colors and 1x1, 1x2, ↩
          2x1,
      2x2,4x2,2x4 and 4x4 (look at the includes). It is also possible to choose  ↩
         the
      FASTEST AVAILABLE, the BEST AVAILABLE mode or the mode that was selected  ↩
         from
      the user as standard mode for his system, using the Screenmode Requester.

      If the user did not specify a standard c2p, this function will use the  ↩
         fastest
      available mode.

      The signal indicates (for asynchrone c2p) that the c2p has done. For  ↩
         synchrone
      ones it is set after quitting the function.

      In mode you specify which c2p mode to use.

      For Graphics Boards, ALWAYS 1x1 is used.

   INPUTS
      RtgScreen - The RtgScreen to use.
      BufAdr - The address of the buffer to use
      Array - The fastram buffer
      Left - The x position on the Bitmap of RtgScreen where to put the stuff
      Top - The y position
      Width - The Width of the stuff
      Height - The Height of the stuff

   SEE ALSO

              CopyRtgPixelArray()
```

1.3 rtgmaster.library/CloseClient

```
                NAME
      CloseClient -- Closes the Client of a TCP/IP connection again
```

```
   SYNOPSIS
        CloseClient(SBase,Socket)
                    A0    A1

        CloseClient(struct Library *,struct TCP_Socket *)

   FUNCTION
        Terminates a "virtual connection" of TCP/IP and gives the Socket of the  ↩
           Client
        back to the system. (For UDP it only gives the socket back to the system,
        as there are no "virtual connections" in connectionless UDP).

        NOTE: It might appear strange to you, that you have to open bsdsocket. ↩
           library
        yourselves and provide it as parameter. This is needed because of some  ↩
           internal
        problems of AmiTCP, that make it IMPOSSIBLE opening it inside a library.  ↩
           Look
        at the Docs for more information.

        You do NOT have to use rtgmaster.library’s Graphics Board features to
        use rtgmaster.library’s TCP/IP features, if you do not WANT to...

   INPUTS
        SBase    - Result of the call (C Syntax here...)
                   SBase = OpenLibrary("bsdsocket.library",0);
        Socket   - The Socket of the Client you want to close.
                   You should ONLY use this function for Clients,
                   NOT FOR SERVERS !!!

 SEE ALSO

                 OpenClient()
                ,
                 OpenServer()
                ,
                 CloseServer()
                ,
                 RunServer()
                ,
                 RtgSend()

                 RtgRecv()
                ,
                 RtgAccept()
                ,
                 RtgIoctl()
                ,
                 GetUDPName()
                ,
                 RtgInAdr()
```

1.4 rtgmaster.library/CloseRtgScreen

```
                NAME
      CloseRtgScreen -- Close a screen previously opened with
                        OpenRtgScreen

   SYNOPSIS
      CloseRtgScreen(RtgScreen)
                      A0

      CloseRtgScreen(ULONG)

   FUNCTION
      Should close a RtgScreen opened by this sublibrary and free all
      of its resources.

   INPUTS
      RtgScreen - A handle for a valid screen previously opened by
                  this sublibrary's OpenRtgScreen() function.

   SEE ALSO

               OpenRtgScreen()
```

1.5 rtgmaster.library/CloseServer

```
                NAME
      CloseServer -- Closes the Server of a TCP/IP connection again

   SYNOPSIS
      CloseServer(SBase,Socket)
                   A0    A1

      CloseServer(struct Library *,struct TCP_Socket *)

   FUNCTION
      Terminates a "virtual connection" of TCP/IP and gives the Socket of the  ←
         Server
      back to the system. For UDP it only gives the socket back to the system,  ←
         as
      for UDP there is no connection to terminate.

      NOTE: It might appear strange to you, that you have to open bsdsocket. ←
         library
      yourselves and provide it as parameter. This is needed because of some  ←
         internal
      problems of AmiTCP, that make it IMPOSSIBLE opening it inside a library.  ←
         Look
      at the Docs for more information.

      You do NOT have to use rtgmaster.library's Graphics Board features to
      use rtgmaster.library's TCP/IP features, if you do not WANT to...
   INPUTS
        SBase    - Result of the call (C Syntax here...)
                   SBase = OpenLibrary("bsdsocket.library",0);
        Socket   - The Socket of the Client you want to close.
                   You should ONLY use this function for Servers,
                   NOT FOR CLIENTS !!!

 SEE ALSO

                 OpenClient()
                ,
                 OpenServer()
                ,
                 CloseClient()
                ,
                 RunServer()
                ,
                 RtgSend()

                 RtgRecv()
                ,
                 RtgAccept()
                ,
                 RtgIoctl()
                ,
                 GetUDPName()
                ,
                 RtgInAdr()
```

1.6 rtgmaster.library/CopyRtgPixelArray

```
                   NAME
        CopyRtgPixelArray -- Copy a rectangular array of pixels directly
                             to the graphics card memory without any
                             conversion

   SYNOPSIS
        CopyRtgPixelArray(RtgScreen, BufferAdr, Array, Left, Top, Width, Height, ←
           SrcX,SrcY);
                          A0         A1         A2     D0    D1   D2     D3      ←
                             D4   D5

        CopyRtgPixelArray(struct RtgScreen *, APTR, APTR, ULONG, ULONG,ULONG,  ←
           ULONG,ULONG,ULONG)

   FUNCTION
        Copies a rectangular array of pixels directly to the graphics card
         memory with no conversion.  The array of pixels is assumed to be
        in the correct native format so it can be copied at maximum speed.
        The copy routine however does take segment boundaries in account (if
        required).

        This routine is mainly intended for machines which have relatively
        fast FastRAM compared to the speed of the graphics card RAM.  This
        is usually on machines with a 32-bit accelerator card which have
        a Zorro-II graphics card installed.

        This function is not supported by the rtgAMI.library.

   INPUTS
        RtgScreen - A handle for a valid screen previously opened by
                    this sublibrary's OpenRtgScreen() function.
        BufferAdr - The address of the memory containing the actual
                    screen graphics
        Array     - Pointer to an array of pixels which is Width pixels wide,
                    and Height pixels high.  The size of the pixel is dependant
                    on the ScreenBuffer your copying to.  Make sure the array
                    is in the correct native format.
        Left      - X position of the top-left of the rectangular pixel array
        Top       - Y position of the top-left of the rectangular pixel array
        Width     - Width of the array in pixels
        Height    - Height of the array in pixels

   SEE ALSO

                 OpenRtgScreen()
                ,
                 WriteRtgPixelArray()
                ,WriteRtgPixelRGBArray()
```

1.7 rtgmaster.library/DrawRtgLine

```
                   NAME
        DrawRtgLine - draws a line on a RtgScreen

   SYNOPSIS
        DrawRtgLine(RtgScreen, BufferAdr, Color, X1, Y1, X2, Y2)
                    A0         A1         D0     D1  D2  D3  D4

        DrawRtgLine(struct RtgScreen *, APTR, ULONG, LONG, LONG, LONG, LONG)

   FUNCTION
        Draws a line on the screen which will be clipped if necessary.
        NOTE : X1<=X2 AND Y1<=Y2 !!!

   INPUTS
        RtgScreen   - A handle for a valid screen previously opened by
                      this sublibrary's OpenRtgScreen() function.
        BufferAdr   - The address of the memory containing the actual
                      screen graphics
        Color       - Color number
        X1,Y1,X2,Y2 - Draws a line from (X1,Y1) to (X2,Y2)

   SEE ALSO

                 OpenRtgScreen()
                ,
                 DrawRtgLineRGB()
```

1.8 rtgmaster.library/DrawRtgLineRGB

```
                NAME
     DrawRtgLineRGB - draws a line on a RtgScreen

SYNOPSIS
     DrawRtgLineRGB(RtgScreen, BufferAdr, Color, X1, Y1, X2, Y2)
                    A0         A1         D0     D1  D2  D3  D4

     DrawRtgLineRGB(struct RtgScreen *, APTR, ULONG, LONG, LONG, LONG, LONG)

FUNCTION
     Draws a line on the screen which will be clipped if necessary.
     NOTE: X1<=X2 AND Y1<=Y2 !!!

INPUTS
     RtgScreen   - A handle for a valid screen previously opened by
                   this sublibrary's OpenRtgScreen() function.
     BufferAdr   - The address of the memory containing the actual
                   screen graphics
     Color       - A 32-bit value describing the color
     X1,Y1,X2,Y2 - Draws a line from (X1,Y1) to (X2,Y2)

SEE ALSO

              OpenRtgScreen()
             ,
              DrawRtgLine()
```

1.9 rtgmaster.library/FillRtgRect

```
                NAME
     FillRtgRect - draws a filled rectangle to a RtgScreen

SYNOPSIS
     FillRtgRect(RtgScreen, BufferAdr, Color, Left, Top, Width, Height)
                 A0         A1         D0     D1    D2   D3     D4

     FillRtgRect(struct RtgScreen *, APTR, ULONG, ULONG, ULONG, ULONG, ULONG)

FUNCTION
     Draws a filled rectangle at the specified position on a RtgScreen.
     The BufferAdr is the starting address of the buffer the users wants
     to draw the rectangle in.  The user has obtained this address using
     LockRtgScreen() and GetBufAdr().  The BufferAdr is needed to specify
     the correct buffer for screens which are double or triple buffered.

     This function should only work for Palette mapped modes, Color is
     the Color number of the palette.
```

```
INPUTS
     RtgScreen - A handle for a valid screen previously opened by
                 this sublibrary's OpenRtgScreen() function.
     BufferAdr - The address of the memory containing the actual
                 screen graphics
     Color     - Color number
     Left      - X position of the top-left of the rectangle
     Top       - Y position of the top-left of the rectangle
     Width     - Width of the rectangle in pixels
     Height    - Height of the rectangle in pixels

SEE ALSO

              OpenRtgScreen()
             ,
              FillRtgRectRGB()
```

1.10 rtgmaster.library/FillRtgRectRGB

```
                NAME
     FillRtgRectRGB - draws a filled rectangle to a RtgScreen

SYNOPSIS
     FillRtgRectRGB(RtgScreen, BufferAdr, Color, Left, Top, Width, Height)
                    A0         A1         D0     D1    D2   D3     D4

     FillRtgRectRGB(struct RtgScreen *, APTR, ULONG, ULONG, ULONG, ULONG, ULONG ↩
        )

FUNCTION
     Draws a filled rectangle at the specified position on a RtgScreen.
     The BufferAdr is the starting address of the buffer the users wants
     to draw the rectangle in.  The user has obtained this address using
     LockRtgScreen() and GetBufAdr().  The BufferAdr is needed to specify
     the correct buffer for screens which are double or triple buffered.

     This function should only work for True Color modes, Color is
     a 32 bit value which specifies what Color the pixel should be.
     The layout of this 32-bit value is as follows:

     %aaaaaaaa.rrrrrrrr.gggggggg.bbbbbbbb

     a = AlphaChannel (8-bits) which may or may not be ignored.  The
         user will set this to zero if the user doesn't want to use
         AlphaChannel.
     r = Red component (8-bits) of the 24-bit RGB value
     g = Green component (8-bits) of the 24-bit RGB value
     b = Blue component (8-bits) of the 24-bit RGB value


INPUTS
     RtgScreen - A handle for a valid screen previously opened by
                 this sublibrary's OpenRtgScreen() function.
```

```
      BufferAdr - The address of the memory containing the actual
                  screen graphics
      Color     - A 32-bit value describing the color (see above)
      Left      - X position of the top-left of the rectangle
      Top       - Y position of the top-left of the rectangle
      Width     - Width of the rectangle in pixels
      Height    - Height of the rectangle in pixels

   SEE ALSO

               OpenRtgScreen()
              ,
               FillRtgRect()
```

1.11 rtgmaster.library/FreeRtgScreenModeReq

```
                 NAME
      FreeRtgScreenModeReq - frees the ScreenReq structure again

   SYNOPSIS
      FreeRtgScreenModeReq(myreq)
                           A0

      FreeRtgScreenModeReq(struct ScreenReq *)

   FUNCTION
      This function frees the memory allocated by RtgScreenModeReq again.
      Should be called after you need that data not any longer. Note: This
      function caused a system crash with an early Beta Version of rtgmaster. ↩
         library.
      This does no longer happen, as this bug got fixed. This function is
      only in the master-library, not in the sublibraries.

   INPUTS
      myreq - The ScreenReq Structure returned by RtgScreenModeReq

   SEE ALSO

               RtgScreenModeReq()
```

1.12 rtgmaster.library/FreeScreenModes

```
                 NAME
      FreeScreenModes - frees a list of screenmodes

   SYNOPSIS
      FreeScreenModes(array of screenmodes)
                      A0

      FreeScreenModes(APTR)
```

```
FUNCTION
     This function should free a previously with GetScreenModes()
     allocated list of ScreenMode structures, including everything else
     GetScreenModes() allocated.  Be prepared to handle a NULL pointer.
     THIS FUNCTION IS ONLY IN SUBLIBRARIES, NOT IN THE MASTER-LIBRARY
     ITSELF. IT IS ONLY CALLED BY RTGMASTER.LIBRARY ITSELF.

INPUTS
     array - an array of ScreenMode structures or NULL

SEE ALSO

              GetScreenModes()
```

1.13 rtgmaster.library/GetBufAdr

```
               NAME
     GetBufAdr -- Get the address for one of the buffers from a
                  multi-buffered RtgScreen

SYNOPSIS
     address = GetBufAdr(RtgScreen, Buffer)
      D0                  A0         D0

     APTR GetBufAdr(ULONG, ULONG)

FUNCTION
     If the user is using multi-buffered screens, it might be usefull
     to know where the two buffers start in memory.  After a
     LockRtgScreen() the user can call this function with a RtgScreen
     handle and a number to get the address of the corresponding
     buffer.

     The address is only valid if the RtgScreen is currently locked
     using LockRtgScreen().

INPUTS
     RtgScreen - A handle for a valid screen previously opened by
                 this sublibrary's OpenRtgScreen() function.
     Buffer    - The buffer number the user wants the address of

RESULTS
     address   - the address of the buffer, or NULL for failure

SEE ALSO

              LockRtgScreen()
             ,
              OpenRtgScreen()
```

1.14 rtgmaster.library/GetRtgScreenData

```
                NAME
     GetRtgScreenData -- Fills a TagList with data about the RtgScreen

SYNOPSIS
     GetRtgScreenData(RtgScreen, TagList)
                      A0         A1

     GetRtgScreenData(ULONG, struct TagItem *)

FUNCTION
     This function should fill the TI_DATA fields of the passed in
     TagList with the requested information.

     See for available tags and descriptions in the .i/.h file.

     NOTE: Starting with sublibrary V2.2, this function can also
     be used to find out to what BUSSYSTEM a Graphics Board is
     connected. In the original design of rtgmaster an extra function
     was intended for that, but now it is the job of GetRtgScreenData.

INPUTS
     RtgScreen - A handle for a valid screen previously opened by
                 this sublibrary's OpenRtgScreen() function.
     TagList   - TagList which should be filled in with requested info.

SEE ALSO

             OpenRtgScreen()
            ,
             GetBufAdr()
```

1.15 rtgmaster.library/GetScreenModes

```
                NAME
     GetScreenModes - builds a list of available screenmodes

SYNOPSIS
     array of ScreenMode structures = GetScreenModes()
      D0

     APTR GetScreenModes()

FUNCTION
     This function should return a linked list of ScreenMode structures
     describing all the available ScreenModes available to this
     sublibrary.  If there aren't any, or you couldn't allocate the
     memory for the list then return 0. THIS FUNCTION IS ONLY IN SUBLIBRARIES,
     NOT IN THE MASTER-LIBRARY ITSELF. IT IS ONLY CALLED BY RTGMASTER.LIBRARY
     ITSELF.

RESULTS
```

```
     array - an array of ScreenMode structures or NULL

SEE ALSO

              FreeScreenModes()
```

1.16 rtgmaster.library/GetSegment

```
                NAME
     GetSegment - get the active segment or ~0

SYNOPSIS
     segnum = GetSegment()
      D0

     ULONG GetSegment()

FUNCTION
     If the graphic board works in segment mode -- with a
     memory window of 64 KByte -- you will get the number of the
     active segment after calling this function.

     If the graphic board works non segmented, you will get
     ~0 (= 0xFFFFFFFF) as a result to this call.

RESULTS
     segnum - number of active segment, or -1 if the board works
              non-segmented

SEE ALSO

              SetSegment()
```

1.17 rtgmaster.library/GetUDPName

```
                NAME
     GetUDPName - Get the sockaddr_in structure of a UDP Client/Server

SYNOPSIS
     name = GetUDPName(SocketBase, sock)
                 A0          A1

     struct sockaddr_in *GetUDPName(struct Library *, struct RTG_Socket *)

FUNCTION
     If this is UDP, you will get the sockaddr_in structure of a Client/Server,
     else you will get 0. For what this is intended, read RtgRecv/RtgSend/ ↩
        RtgInAdr.
     And of course the Docs of rtgmaster.library where detailed information  ↩
        about
        TCP, IP and UDP is found.

        NOTE: It might appear strange to you, that you have to open bsdsocket. ↩
           library
        yourselves and provide it as parameter. This is needed because of some  ↩
           internal
        problems of AmiTCP, that make it IMPOSSIBLE opening it inside a library.  ↩
           Look
        at the Docs for more information.

        You do NOT have to use rtgmaster.library’s Graphics Board features to
        use rtgmaster.library’s TCP/IP features, if you do not WANT to...

   INPUTS
        SBase    - Result of the call (C Syntax here...)
                   SBase = OpenLibrary("bsdsocket.library",0);
        Socket   - The Socket of the Client you want to close.
                   You should ONLY use this function for Clients,
                   NOT FOR SERVERS !!!

   RESULTS
        name     - The sockaddr_in structure of the Client/Server

 SEE ALSO

                  OpenClient()
                 ,
                  OpenServer()
                 ,
                  CloseServer()
                 ,
                  RunServer()
                 ,
                  RtgSend()

                  RtgRecv()
                 ,
                  RtgAccept()
                 ,
                  RtgIoctl()
                 ,
                  RtgInAdr()
```

1.18 rtgmaster.library/LoadRGBRtg

```
                    NAME
        LoadRGBRtg - changes one or more colors of a RtgScreen

   SYNOPSIS
        LoadRGBRtg(RtgScreen, Table)
                   A0         A1

        LoadRGBRtg(ULONG, APTR)
```

```
FUNCTION
     Enables the user to change one or more colors of his/her screen.
     This function only works for RtgScreens which have a palette, and
     thus won't work for the True-color modes.

INPUTS
     RtgScreen - A handle for a valid screen previously opened by
                 this sublibrary's OpenRtgScreen() function.
     Table     - A pointer to a series of records which describe which
                 colors to modify

NOTES
     Passing a NULL Table must be ignored.  The format of the Table
     passed is a series of records, each with the this format:

        WORD  Count value: Number of colors to load
        WORD  Number of first color to be loaded

     After these two words, a list of 3 Longs follow as many times
     as specified by the first word.  These 3 longwords represent the
     left justified 32 bit RGB value.

     And then the list repeats until ended with a count value of 0.

     See for more information about the table graphics/LoadRGB32.
     This function must use the same format.

SEE ALSO

              OpenRtgScreen()
             , graphics/LoadRGB32()
```

1.19 rtgmaster.library/LockRtgScreen

```
                NAME
     LockRtgScreen -- Locks a RtgScreen (prevents it from being moved
                      in memory)

SYNOPSIS
     address = LockRtgScreen(RtgScreen)
      D0                     A0

     APTR LockRtgScreen(ULONG)

FUNCTION
     This function should make sure that the screen is not moved from
     it's current location in memory.  In other words, it will guarantee
     that the address you get back from this function remains valid
     until you call UnlockRtgScreen().

     The result from this function should be the address of the buffer
     associated with the screen either in the graphics cards own
```

```
        memory or the computers memory.

        For multi-buffered screens the return-address must point to buffer
        0 for this RtgScreen.  To get the addresses of the other
        buffers the user will use GetBufAdr().

        LockRtgScreen and UnlockRtgScreen functions must nest, which means
        you must call an UnlockRtgScreen for every LockRtgScreen.  The
        field rs_Locks in the RtgScreen structure should be used to keep
        track of the number of times a screen was locked.

        Note : On some Graphics Boards this function will take some
        CPU time to happen, so it is advised ONLY to call it *once* at the start
        of your code (And UnlockRtgScreen *once* at the end of your code,
        to be on the sure side...)

   INPUTS
        RtgScreen - A handle for a valid screen previously opened by
                    this sublibrary's OpenRtgScreen() function.

   RESULTS
        address   - The address of the (first) buffer of this screen.

   SEE ALSO

                 UnlockRtgScreen()
                ,
                 OpenRtgScreen()
                ,
                 GetBufAdr()
```

1.20 rtgmaster.library/OpenClient

```
                   NAME
        OpenClient    -- Open a TCP/IP Client

   SYNOPSIS
        Socket = OpenClient(SBase,host,port,mode,protocol)
         D0                 A0    A1   D0   D1   D2

        struct RTG_Socket *OpenClient(struct Library *,char *,int,int,int)

   FUNCTION
        For TCP, this function opens a "virtual connection" between two  ↩
           applications.
        For UDP it creates a socket that can be used by the application to
        transfer data connectionless to other applications.
        This function is the "Client part" of the connection. The protocol
        being used is TCP/IP.

        For more information, look at the docs. There is a chapter about
        "TCP/IP programming for newcomers", that shows you, how to support
        netework gaming for your computer game, even if you never heard of
        TCP/IP before :) Up to now rtgmaster.library only supports the
```

```
     "protocol stack" AmiTCP, no AS225 support up to now.  Runs for sure
     with AmiTCP 4.0 demo from Aminet, i do not know about earlier versions.

     NOTE: It might appear strange to you, that you have to open bsdsocket. ↩
        library
     yourselves and provide it as parameter. This is needed because of some  ↩
        internal
     problems of AmiTCP, that make it IMPOSSIBLE opening it inside a library.  ↩
        Look
     at the Docs for more information.

     You do NOT have to use rtgmaster.library's Graphics Board features to
     use rtgmaster.library's TCP/IP features, if you do not WANT to...

INPUTS
     SBase    - Result of the call (C Syntax here...)
                SBase = OpenLibrary("bsdsocket.library",0);
     host     - hostname of the "Server",  to which you want to connect your
                application (for example "194.55.101.26").
     port     - The port your application uses. For example 4000.
                Be sure to use a number bigger than 3000, small numbers
                are often used for different protocols in TCP/IP. For example
                21 is telnet.
     mode     - The mode of the connection. Up to now only SOCK_STREAM is
                supported ("virtual connection using a datastream").
                SOCK_DGRAM probably will give you a UDP connection, but
                I do not know enough about UDP to make this really working..
                maybe in a future version...
     protocol - The protocol To be used. Set this to 0 currently.
                mode SOCK_STREAM and protocol 0 will result in a TCP connection ↩
                   .


RESULTS
     Socket   - The "Socket" of the Application. See more in the docs.

SEE ALSO

             OpenServer()
            ,
             CloseClient()
            ,
             CloseServer()
            ,
             RunServer()
            ,
             RtgSend()

             RtgRecv()
            ,
             RtgAccept()
            ,
             RtgIoctl()
            ,
             GetUDPName()
            ,
             RtgInAdr()
```

1.21 rtgmaster.library/OpenRtgScreen

```
           NAME
      OpenRtgScreen -- Open a screen

   SYNOPSIS
      RtgScreen = OpenRtgScreen(ScreenReq, RtgTags)
       D0                       A0         A1

      struct RtgScreen *OpenRtgScreen(struct ScreenReq *, struct TagItem *)


   FUNCTION
      This function should open the screen which falls within the
      parameters specified by the user.  If this function can't
      deliver such a screen than it will fail and will return zero.

      Note that the Width and Height values you get from
      rtgmaster.library have been checked to see if they are valid
      for this screenmode.  Also note that RtgTags may be zero.

   INPUTS
      RtgTags   - Pointer to (an array of) TagItem structures,
                  terminated by the value TAG_END (0).
      ScreenReq - ScreenReq structure as returned by RtgScreenModeReq()
                  of rtgmaster.library, see rtg.i for more information

      Each TagItem is an optional tagged data structure which
      identifies a parameter to OpenRtgScreen().  The applicable tag ID
      values for TagItem.ti_Tag and their corresponding data can be
      found in the .i/.h file where the Tags for OpenRtgScreen() are
      specified.

   RESULTS
      RtgScreen - A handle for the screen you opened.  The user may
                  later use this handle to get information about
                  this screen or perform actions like setting the
                  palette or double/triple buffering.  You should
                  returns NULL if the screen couldn't be opened.

   SEE ALSO

            CloseRtgScreen
```

1.22 rtgmaster.library/OpenServer

```
           NAME
      OpenServer - Opens a TCP/IP Server
```

```
   SYNOPSIS
	Socket=OpenServer(SBase,port,mode,protocol)
	 D0                A0    D0   D1   D2

	struct TCP_Connect *OpenServer(struct Library *,int,int,int)

   FUNCTION
	For TCP this function opens a "virtual connection" between two  ↩
	   applications.
	For UDP it creates a server that UDP clients can access.

	This function is the "Server part" of the connection. The protocol
	being used is TCP/IP. Up to now, as to the "transport protocol", only
	TCP is supported, no UDP (maybe in a future version ???)
	For more information, look at the docs. There is a chapter about
	"TCP/IP programming for newcomers", that shows you, how to support
	netework gaming for your computer game, even if you never heard of
	TCP/IP before :) Up to now rtgmaster.library only supports the
	"protocol stack" AmiTCP, no AS225 support up to now.  Runs for sure
	with AmiTCP 4.0 demo from Aminet, i do not know about earlier versions.

	NOTE: It might appear strange to you, that you have to open bsdsocket. ↩
	   library
	yourselves and provide it as parameter. This is needed because of some  ↩
	   internal
	problems of AmiTCP, that make it IMPOSSIBLE opening it inside a library.  ↩
	   Look
	at the Docs for more information.

	You do NOT have to use rtgmaster.library's Graphics Board features to
	use rtgmaster.library's TCP/IP features, if you do not WANT to...

   INPUTS
	SBase    - Result of the call (C Syntax here...)
	           SBase = OpenLibrary("bsdsocket.library",0);
	port     - The port your application uses. For example 4000.
	           Be sure to use a number bigger than 3000, small numbers
	           are often used for different protocols in TCP/IP. For example
	           21 is telnet.
	mode     - The mode of the connection. Up to now only SOCK_STREAM is
	           supported ("virtual connection using a datastream").
	           SOCK_DGRAM probably will give you a UDP connection, but
	           i do not know enough about UDP to make this really working...
	           maybe in a future version...
	protocol - The protocol To be used. Set this to 0 currently.
	           mode SOCK_STREAM and protocol 0 will result in a TCP connection ↩
	               .

   RESULTS
	Socket   - The "Socket" of the Application. See more in the docs.

  SEE ALSO

                OpenClient()
               ,
                CloseClient()
               ,
                CloseServer()
               ,
                RunServer()
               ,
                RtgSend()

                RtgRecv()
               ,
                RtgAccept()
               ,
                RtgIoctl()
               ,
                GetUDPName()
               ,
                RtgInAdr()
```

1.23 rtgmaster.library/RtgAccept

```
                  NAME
        RtgAccept - Let the server accept a connection deminded by a Client

   SYNOPSIS
        Socket=RtgAccept(SBase,Socket)
         D0               A0    A1

        struct RTG_Socket *RtgAccept(struct Library *,struct RTG_Socket *)

   FUNCTION
        If you do not use the RunServer function (you do not use it, if you only
        do a point-to-point connection), you have to do this call on Server side
        to wait for the Client to connect. If you use RunSercer, DO NOT USE IT. It
        is only for connecting exactly TWO systems (one being  the server, one the
        client), not for connecting ONE server with SEVERAL clients...

        Also do not use it for UDP connection. RtgAccept is only needed for TCP.
        It does not work with UDP.

        NOTE: It might appear strange to you, that you have to open bsdsocket. ↩
            library
        yourselves and provide it as parameter. This is needed because of some  ↩
            internal
        problems of AmiTCP, that make it IMPOSSIBLE opening it inside a library.  ↩
            Look
        at the Docs for more information.

        You do NOT have to use rtgmaster.library's Graphics Board features to
        use rtgmaster.library's TCP/IP features, if you do not WANT to...


   INPUTS
        SBase    - Result of the call (C Syntax here...)
                   SBase = OpenLibrary("bsdsocket.library",0);
        Socket   - the Socket of the Server
```

```
    RESULTS
        Socket   - The "Socket" of the Client, that connected. See more in the
docs.

    SEE ALSO

                 OpenClient()
                ,
                 CloseClient()
                ,
                 CloseServer()
                ,
                 RunServer()
                ,
                 RtgSend()

                 RtgRecv()
                ,
                 RtgIoctl()
                ,
                 GetUDPName()
                ,
                 RtgInAdr()
```

1.24 rtgmaster.library/RtgBlit

```
                   NAME
        RtgBlit - Performs a Blit without waiting

   SYNOPSIS
        RtgBlit(RtgScreen,SrcBuf,DstBuf,SrcX,SrcY,DstX,DstY,Width,Height,Minterm)
                 A0        a1     a2     d0   d1   d2   d3   d4    d5     d6

        void RtgBlit(struct RtgScreen *, ULONG, ULONG, ULONG, ULONG, ULONG, ULONG, ↩
            ULONG, ULONG,UBYTE)

   FUNCTION
        This function blits the rectangle at (SrxX,SrcY) in the Buffer with the
        NUMBER SrcBuf (0-2) to the position (DstX,DstY) in the Buffer with the
        NUMBER DstBuf (0-2). The Blit has Width Width and Height Height.

        For most GFX Boards this function is the fastest way to move
        graphics data.

        Note : On some boards (for example EGS Boards) this function might
        wait on the Blitter to be finished as this can't be done in an
        other way with these boards. For these boards WaitRtgBlit simply
        does nothing.

        Note : The source and the destination rectangle should NOT OVERLAP !!!

        Valid minterms : $30,$50,$60,$80, $C0. NO OTHER MINTERMS ARE VALID.
        OTHER MINTERMS MIGHT WORK WITH SOME SUBLIBRARIES, BUT PROBABLY NOT WITH
        ALL SUBLIBRARIES.
```

```
     Note: This function MIGHT or MIGHT NOT work with some of the Minterms
     on rtgEGS.library... at least for $C0 it works for all... for the rest...
     i do not see myself as Betatester of half-finished WB-Emulations...

INPUTS
     RtgScreen - The RtgScreen where the Blit should happen
     SrcBuf    - The Buffer NUMBER (not address !!!) of the Source Buffer
     DstBuf    - The Buffer NUMBER (not address !!!) of the Destination Buffer
     SrcX      - The X coordinate of the source Rectangle
     SrcY      - The Y coordinate of the source Rectangle
     DstX      - The X coordinate of the Destination Rectangle
     DstY      - The Y coordinate of the Destination Rectangle
     Width     - The Width of the Blit
     Height    - The Height of the Blit
     minterm   - the minterm of the Blit, defined as usual

SEE ALSO

              OpenRtgScreen()
             ,
              WaitRtgBlit()
             ,
              SwitchScreens()
```

1.25 rtgmaster.library/RtgBltClear

```
                SYNOPSIS
     RtgBltClear(RtgScreen,BufNum,xpos,ypos,width,height)

     RtgBltClear(struct RtgScreen *,ULONG,ULONG,ULONG,ULONG,ULONG)

FUNCTION
     This function clears a rectangular area using the GFX Board
     blitter. For people who wonder, why i did not implement that
     the "usual" way, like done in graphics.library : The graphics.library
     function would not be possible under EGS, therefor i did it
     this way. The function usually does not wait for the Blitter,
     use WaitRtgBlit for this (unless under EGS... like explained
     in RtgBlit and WaitRtgBlit...)

INPUTS
     RtgScreen - The RtgScreen
     BufNum    - The NUMBER of the concerned Buffer, between 0 and 2 (NOT the
                 buffer address !!!)
     xpos      - the start x position of the rectangle to be cleared
     ypos      - the start y position of the rectangle to be cleared
     width     - the width of the rectangle
     height    - the height of the rectangle

SEE ALSO

              RtgBlit()
             ,
                 WaitRtgBlit()
```

1.26 rtgmaster.library/RtgClearPointer

```
                   NAME
         RtgClearPointer - resets the pointer to its default image

    SYNOPSIS
         RtgClearPointer(RtgScreen)
                          A0

         void RtgClearPointer(struct RtgScreen *)

    FUNCTION
         This restores the default image of the mousepointer. This is
         very useful, if you changed it with RtgSetPointer, but sometimes
         want the default pointer image, too. The pointer is only changed
         on THIS RtgScreen.

    INPUTS
         RtgScreen - The RtgScreen, which pointer should be resetted...

    NOTES
        Not implemented yet on rtgPICA.library and rtgEGS.library

    SEE ALSO

                 RtgSetPointer()
```

1.27 rtgmaster.library/RtgCloseFont

```
                   NAME
         RtgCloseFont - closes an AmigaFont

    SYNOPSIS
         RtgCloseFont(RtgScreen,font)
                       A0        A1

         void RtgCloseFont(struct RtgScreen *,void *)

    FUNCTION
         This function closes an AmigaFont on a RtgScreen, much the
         same way, like OpenDiskFont does for Intuition Screens.
         The font parameter of the call is not for all WB Emulations
         a TextFont pointer. Don't use CloseFont with rtgmaster.library,
         use RtgCloseFont, for the best possible compatibility with
         all Sublibraries !!!

    INPUTS
         RtgScreen - an RtgScreen
```

```
      font      - a Font pointer. the structure of it is PRIVATE
                  to rtgmaster.library (and not the same for all
                  sublibraries...)

   NOTES
      Not yet implemented for rtgPICA.library

   SEE ALSO

               RtgOpenFont()
              ,
               RtgSetFont()
              ,
               RtgSetTextMode()
              ,
               RtgText()
              ,
               RtgSetTextModeRGB()
```

1.28 rtgmaster.library/RtgInAdr

```
                 NAME
      RtgInAdr - Find out the IP Address of a Receiver/Sender

   SYNOPSIS
      ip = RtgInAdr(SBase,si)
      D0             A0   A1

      char *RtgInAdr(struct Library *,struct sockaddr_in *)

   FUNCTION
      This function finds out the IP Address of a Receiver/Sender.
      You get back the sockaddr_in structure of a receiver/sender by
      RtgRecv/RtgSend or by using GetUDPName. This function ONLY
      works for UDP, not for TCP !!! It is used to differentiate
      Clients running on different machines from each other, if
      one does a Multiple Client "connection" (should not be called
      like that, as UDP is connectionless, but do you know a better
      term ?), without using RunServer...

      Of course this function can't differentiate multiple Clients
      running on the same machine...

      NOTE: It might appear strange to you, that you have to open bsdsocket. ←
         library
      yourselves and provide it as parameter. This is needed because of some  ←
         internal
      problems of AmiTCP, that make it IMPOSSIBLE opening it inside a library.  ←
         Look
      at the Docs for more information.

      You do NOT have to use rtgmaster.library's Graphics Board features to
      use rtgmaster.library's TCP/IP features, if you do not WANT to...
```

```
INPUTS
       SBase    - Result of the call (C Syntax here...)
                  SBase = OpenLibrary("bsdsocket.library",0);
       si       - Special structure returned by above mentioned calls

RESULTS
       ip       - IP Address as string (for example "194.55.101.26")

SEE ALSO

              OpenServer()
             ,
              OpenClient()
             ,
              CloseClient()
             ,
              CloseServer()
             ,
              RunServer()

              RtgSend()
             ,
              RtgAccept()
             ,
              GetUDPName()
```

1.29 rtgmaster.library/RtgInitRDCMP

```
                NAME
     RtgInitRDCMP - Inits the rtgmaster direct communication message port
                    (RDCMP)

SYNOPSIS
     result = RtgInitRDCMP(RtgScreen)
      d0                    a0

     struct RDCMPData *RtgInitRDCMP(struct RtgScreen *)

FUNCTION
     Inits the input port of rtgmaster. Returns 0, if initialization failed ( ←
        for example,
     if sublibrary does not support RDCMP), something >0 else. The port has to  ←
        be initialized
     once after Screen-Opening. The result will be a pointer to :

     struct RDCMPData
     {
      struct MsgPort *port;
      ULONG signal;
      WORD *MouseX;
      WORD *MouseY;
     };

     Port is the MessagePort of the RtgScreen. NOTE: Better do NOT use
```

```
        that port, the way it handles events, might differ according to the
        WB Emulation you use, better use the RtgGetMsg function to get
        Messages !!!
        The Port was only included, as the coder of a certain game wanted this.

        Signal will contain the 1<<mp_SigBit of the MessagePort of the RtgScreen
        for fast Input-Handling.

        MouseX and MouseY contain POINTERS to the current mouse position. This ←
           sort
        of checking is faster than using GetRtgScreenData for the mouse position.

        NOTE: All RDCMP/Font/Text functions won't work on the rtgPICA.library,due
        to the Original Picasso II WB Emulation having some limitations (you only ←
           get
        Direct Video RAM Access *OR* IDCMP/Font/Text).

   INPUTS
        RtgScreen - The Screen, which port is to init

   RESULTS
        result - 0, if failed, something else, if succeeded

   NOTES
       RDCMP supports both waiting and polling !!!
       As to my experiences, better use RDCMP than anything else...
       other methods i tried tended to lose mouseclicks, if
       they came very fast, and if the application took a lot
       of processing time. RDCMP does not lose data.

   SEE ALSO

                 RtgWaitRDCMP()
                ,
                 RtgGetMsg()
                ,
                 RtgReplyMsg()
```

1.30 rtgmaster.library/RtgIoctl

```
                    NAME
        RtgIoctl - Set a Socket to "Blocking" or to "Non-Blocking" mode

   SYNOPSIS
        result = RtgIoctl(SBase,Socket,arg)
         D0                 A0    A1     A2

        int RtgIoctl(struct Library *,struct RTG_Socket *,long *)

   FUNCTION
        This function determins, if RtgRecv and RtgSend will WAIT will the data
        was transmitted, or if they fail, if the data currently could not be
        transmitted. If arg POINTS to the VALUE 1, we have "non-blocking" (it
        does not wait), if it POINTS to 0, we have "blocking" (it waits). Default
        (if you do not call RtgIoctl at all) is "blocking".

        NOTE: It might appear strange to you, that you have to open bsdsocket. ↩
           library
        yourselves and provide it as parameter. This is needed because of some  ↩
           internal
        problems of AmiTCP, that make it IMPOSSIBLE opening it inside a library.  ↩
           Look
        at the Docs for more information.

        You do NOT have to use rtgmaster.library’s Graphics Board features to
        use rtgmaster.library’s TCP/IP features, if you do not WANT to...

        RtgIoctl is a VERY CPU TIME INTENSIVE FUNCTION !!!
        Only call it during the INITIALIZATION of the network !!!

        My suggestion : Run the server in "blocking" mode, the Clients in
        "nonblocking" mode... seems to be the fastest...

   INPUTS
        SBase    - Result of the call (C Syntax here...)
                   SBase = OpenLibrary("bsdsocket.library",0);
        Socket   - The Socket of the application, which socket is to be modified
        arg      - "Non-Blocking" or "Blocking" (a pointer)

   RESULTS
        result   - 0 on success, -1 on fail (should not fail,normally...)

   SEE ALSO

              OpenServer()
             ,
              OpenClient()
             ,
              CloseClient()
             ,
              CloseServer()
             ,
              RunServer()
             ,

              RtgSend()
             ,
              RtgRecv()
             ,
              GetUDPName()
             ,
              RtgInAdr()
```

1.31 rtgmaster.library/RtgOpenFont

```
                NAME
        RtgOpenFont - opens an AmigaFont
```

```
SYNOPSIS
     font = RtgOpenFont(RtgScreen,ta)
      D0                 A0       A1

     void * RtgOpenFont(struct RtgScreen *,struct TextAttr *)

FUNCTION
     This function loads an AmigaFont to memory, for the usage on
     a RtgScreen, much the same way, like OpenDiskFont does for
     Intuition Screens. ta is a normal TextAttr pointer, like
     for OpenDiskFont for diskfont.library, but the RESULT
     is not for all WB Emulations a TextFont pointer. Don't use
     OpenDiskFont or OpenFont with rtgmaster.library, use
     RtgOpenFont, for the best possible compatibility with
     all Sublibraries !!!

INPUTS
     RtgScreen - an RtgScreen
     ta        - a TextAttr structure, like defined in graphics/text.i (or .h)

RESULTS
     font      - A pointer to a font pointer. It's internal structure is
                 PRIVATE to rtgmaster.library, and NOT THE SAME for all
                 sublibraries

NOTES
     Not yet implemented for rtgPICA.library

SEE ALSO

              RtgCloseFont()
             ,
              RtgSetFont()
             ,
              RtgSetTextMode()
             ,
              RtgText()
             ,
              RtgSetTextModeRGB()
```

1.32 rtgmaster.library/RtgRecv

```
                NAME
     RtgRecv - The Socket of this application receives data from a connected
socket

SYNOPSIS
     length = RtgRecv(SBase,Socket,message,sender,len)
      D0               A0    A1      A2      A3     D0

     int RtgRecv(struct Library *,struct RTG_Socket *,struct char *,struct  ↩
        sockaddr_in *,int)

FUNCTION
```

```
This function is no longer compatible to rtgmaster.library V6 and below ↩
   !!!

This function gets data to the application that is sent to its socket by a ↩
     socket
that is connected to the socket of the Application (it is not that ↩
   difficult to
understand like it sounds... read the docs :) )

If the "virtual connection" (or the "connectionless connection" or what ↩
   this is called
for UDP... :) ) is "blocking", it WAITS, if there is no message available
on the socket, till one is available. If it is "non-blocking", it returns ↩
   -1, if no
message is available (see RtgIoctl for more details about "blocking" and " ↩
   Non-Blocking".
The default is "Blocking", BTW...)

Sender is a special structure that you can use to find out from what IP ↩
   Address
the message was sent. You can convert it to an IP Address using RtgInAdr. ↩
   This
only works with UDP (with TCP, you differentiate Clients from each other ↩
   with the
socket number...). Appearently, you CAN'T differentiate Clients that run ↩
   on the
same machine !!! Sender is NOT allocated by the function, you have to ↩
   provide
the structure...

For TCP you simple provide a 0 for Sender. It will be ignored.

You can use this feature to do multiple connection without using RunServer ↩
     (that
does not support UDP up to now anyways...).

NOTE: For *TCP* you provide the Socket of the Application which you want ↩
   to
contact. For *UDP* you provide your OWN'S Socket. Important difference !!!

NOTE: It might appear strange to you, that you have to open bsdsocket. ↩
   library
yourselves and provide it as parameter. This is needed because of some ↩
   internal
problems of AmiTCP, that make it IMPOSSIBLE opening it inside a library. ↩
   Look
at the Docs for more information.

You do NOT have to use rtgmaster.library's Graphics Board features to
use rtgmaster.library's TCP/IP features, if you do not WANT to...

NOTE: It is NOT possible to give Socket->s (the Socket Number) of a Client
to the Server using RtgRecv or RunServer !!! You will have to examine
inbuffer->num[x] to find out which Socket was the Sender !!! Also len
should NEVER be bigger than the actual message, or you might get a lot
of strange results !!!
```

```
   INPUTS
        SBase    - Result of the call (C Syntax here...)
                   SBase = OpenLibrary("bsdsocket.library",0);
        Socket   - The Socket of THIS application
        message  - The buffer, to which supplied messages will be put
        Sender   - The Function will fill in data about the Sender to this ↩
           structure.
        len      - The length of the message to wait for

   RESULTS
        length   - The length of the message received. If it is smaller
                   than the message, you waited for, do a RtgRecv once more...
                   NOTE: If a message was received that is too long to fit to
                   the buffer, some Bytes might be discarded. So you should
                   not send more Bytes than you want to receive... use a standard
                   Package Size at best...
    SEE ALSO

                 OpenClient()
                ,
                 CloseClient()
                ,
                 CloseServer()
                ,
                 RunServer()
                ,
                 RtgSend()

                 RtgRecv()
                ,
                 RtgIoctl()
                ,
                 GetUDPName()
                ,
                 RtgInAdr()
```

1.33 rtgmaster.library/RtgScreenAtFront

```
                   NAME
        RtgScreenAtFront - determines if a RtgScreen is at front

   SYNOPSIS
        boolean = RtgScreenAtFront(RtgScreen)
                                     A0

        RtgScreenAtFront(struct RtgScreen *)

   FUNCTION
        This function should determine if this RtgScreen is currently at
        front.  It should return TRUE (0xfffffff) if the screen is in front
        of all other screens, and FALSE (0) if the screen is behind (partially
        or completely) an other screen.

        Note that this function will not be heavily reliable, since the user
```

```
     might switch screens at any time.

   INPUTS
     RtgScreen - A handle for a valid screen previously opened by
                 this sublibrary's OpenRtgScreen() function.

   RESULTS
     boolean - TRUE if screen is at front, FALSE otherwise.

   SEE ALSO

                OpenRtgScreen()
```

1.34 rtgmaster.library/RtgScreenModeReq

```
                   NAME
     RtgScreenModeReq -- Opens a ScreenMode requester

   SYNOPSIS
     ScreenReq = RtgScreenModeReq(ScreenModeTags)
      D0                           A0

     struct ScreenReq *RtgScreenModeReq(struct TagItem *)

   FUNCTION
     Opens a ScreenMode requester which displays all available
     ScreenModes to the user, depending on the Tags which are passed
     to this function.  The functions returns a pointer to a
     ScreenReq structure or NULL for failure or if the user
     cancelled the requester.

     The ScreenReq holds various information which the user selected,
     like width, height, screenmode and depth.

     THIS FUNCTION IS ONLY IN THE MASTER-LIBRARY. It handles the Screenmodes
     for ALL sublibraries.

     Note: The Screenmode-Requester provided with an early Beta of the
     rtgmaster.library was VERY buggy. It got completely replaced by
     a new one for this version of the library.

   INPUTS
     ScreenModeTags - Pointer to (an array of) TagItem structures,
                      terminated by the value TAG_END (0).

     The description for the various tags can be found in the .i/.h
     file (smr_Tags). NOTE: The Tags changed a lot since the early
     Beta release of the library.

   RESULTS
     ScreenReq - A pointer to a ScreenReq structure or NULL for
                 failure

   SEE ALSO
```

```
             OpenRtgScreen()
            ,
             FreeRtgScreenModeReq()
```

1.35 rtgmaster.library/RtgSend

```
               NAME
      RtgSend - The Socket of this application sends data to a connected socket

   SYNOPSIS
      length = RtgSend(SBase,Socket,message,Receiver,len)
       D0               A0    A1     A2      A3       D0

      int RtgSend(struct Library *,struct RTG_Socket *,struct char *,struct  ↩
         sockaddr_in *,int)

   FUNCTION
      This function is no longer compatible to rtgmaster.library V6 and below  ↩
         !!!

      This function puts data from the application to the socket of an  ↩
         application
      that is connected to the socket of the Application (it is not that  ↩
         difficult to
      understand like it sounds... read the docs :) )

      If the "virtual connection" (well, this term does not fit for UDP, as it  ↩
         is
      connectionless, but i do not know how to call it else... :) ) is "blocking ↩
         ",
      it WAITS, if there is no message available on the socket, till one is  ↩
         available.
      If it is "non-blocking", it returns -1, if no message is available (see  ↩
         RtgIoctl
      for more details about "blocking" and "Non-Blocking".
      The default is "Blocking", BTW...)

      Receiver is a special structure that you can use to tell the receiver your ↩
          IP Address.
      You get your IP Address with GetUDPName (works only for UDP... for TCP  ↩
         this returns
      0...). For TCP you provide simply 0 for Receiver, it will be ignored.
      You can convert it to an IP Address using RtgInAdr. This
      only works with UDP (with TCP, you differentiate Clients from each other  ↩
         with the
      socket number...). Appearently, you CAN'T differentiate Clients that run  ↩
         on the
      same machine !!!

      You can use this feature to do multiple connection without using RunServer ↩
          (that
      does not support UDP up to now anyways...).
```

```
     NOTE: For *TCP* you provide the Socket of the Application which you want  ←
        to
     contact. For *UDP* you provide your OWN'S Socket. Important difference !!!

     NOTE: It might appear strange to you, that you have to open bsdsocket. ←
        library
     yourselves and provide it as parameter. This is needed because of some  ←
        internal
     problems of AmiTCP, that make it IMPOSSIBLE opening it inside a library.  ←
        Look
     at the Docs for more information.

     You do NOT have to use rtgmaster.library's Graphics Board features to
     use rtgmaster.library's TCP/IP features, if you do not WANT to...

     NOTE: It is NOT possible to give Socket->s (the Socket Number) of a Client
     to the Server using RtgRecv or RunServer !!! You will have to examine
     inbuffer->num[x] to find out which Socket was the Sender !!! Also len
     should NEVER be bigger than the actual message, or you might get a lot
     of strange results !!!

INPUTS
     SBase    - Result of the call (C Syntax here...)
                SBase = OpenLibrary("bsdsocket.library",0);
     Socket   - The Socket of THIS application
     message  - The message to be sent (an ASCII string, actually...)
     Receiver - The data you got from GetUDPName, tells the receiver your IP  ←
        address...
     len      - The length of the string to send ...

RESULTS
     length   - The length of the message sent. If it is smaller
                than the message, you sent, do a RtgSend once more...
                Probably the socket you sent to was quite busy with other
                messages currently...

                NOTE: If the send fails at all, maybe your message was too
                long for TCP/IP ? (As to the allowed package sizes, 1 KB works
                for sure... i do not know exactly how much more is possible...)

 SEE ALSO
```

OpenClient()
,
CloseClient()
,
CloseServer()
,
RunServer()
,
RtgSend()

RtgRecv()
,
RtgIoctl()
,
GetUDPName()

```
             ,
              RtgInAdr()
```

1.36 rtgmaster.library/RtgSetFont

```
              NAME
        RtgSetFont - sets an AmigaFont to an RtgScreen

   SYNOPSIS
        RtgSetFont(RtgScreen,font)
                    A0        A1

        void RtgSetFont(struct RtgScreen *,void *)

   FUNCTION
        This function sets an AmigaFont (that was opened using
        RtgOpenFont tomemory before) to an RtgScreen. Following
        RtgText() calls will use this font, now.

   INPUTS
        RtgScreen - an RtgScreen
        font      - a Font pointer. the structure of it is PRIVATE
                    to rtgmaster.library (and not the same for all
                    sublibraries...)

   NOTES
        Not yet implemented for rtgPICA.library

   SEE ALSO

              RtgCloseFont()
             ,
              RtgSetFont()
             ,
              RtgSetTextMode()
             ,
              RtgText()
             ,
              RtgSetTextModeRGB()
```

1.37 rtgmaster.library/RtgSetPointer

```
              NAME
        RtgSetPointer - sets the pointer to a new image

   SYNOPSIS
        RtgSetPointer(RtgScreen,pointer,Width,Height,OffsetX,OffsetY)
                       A0        A1      D0     D1     D2      D3

        void RtgSetPointer(struct RtgScreen *,UWORD *,WORD,WORD,WORD,WORD)
```

```
FUNCTION
     This function sets the mousepointer to a new image for this RtgScreen.
     The structure pointer is exactly the same like taken from intuition. ↩
        library
     command SetPointer (the data definition of a Simple Sprite). Different
     from this only is that the image HAS TO BE 16x18 pixels size. If you
     want a smaller pointer, modify the Width/Height values and set that
     pixels that you do not need to 0. But the SIZE of the structure has
     to be 16x18 pixels (72 Bytes). An example :

     UWORD Hoehe[2]
     {
         0x...,0x... // first line
         ...
         0x...,0x... // 18th line
     }

     Look at graphics.library SimpleSprite documentation for more information
     (01,10,11 are the three pointer colors,00 is transperent, each of
     the two words of a line determines ONE BIT of the 2-Bit mousepointer).

INPUTS
     RtgScreen - The RtgScreen, which pointer should be resetted...
     pointer   - pointer image, see above
     Width     - Actual Width of the pointer
     Height    - Actual Height of the pointer
     OffsetX   - Display Offset for the pointer, from the mouseposition
     OffsetY   - Display Offset for the pointer, from the mouseposition

NOTES
    Not implemented yet on rtgPICA.library and rtgEGS.library

    Do NOT specifically demand Chipram for the pointer array.
    Else your program won't run on the DraCo. rtgmaster.library will
    handle this itself, that the pointer-image will end in
    Chipram for ECS/AGA, in ANYTHING AVAILABLE on GFX Board system.
    So simply allocate RAM for the pointer image, without simply
    demanding Chipram or Fastram (as DraCo needs FastRam, and
    ECS/AGA need Chipram here... but well... rtgmaster.library
    takes care of this :) )

SEE ALSO

              RtgClearPointer()
```

1.38 rtgmaster.library/RtgGetMsg

```
                NAME
     RtgGetMsg - replies the message on an RDCMP

SYNOPSIS
     imsg = RtgGetMsg(RtgScreen)
      d0              a0
```

```
        void *RtgGetMsg(struct RtgScreen *)

   FUNCTION
        Gets the latest message of a RDCMP. RDCMP uses the
        structures of the IntuiMessage of intuition (but note,
        this is NOT an IDCMP... it only simulates the structures
        of the IDCMP !!!)
        List of the structures for those guys without OS includes
        (look at intuition/intuition.h or .i) (ASM notation, as
        C guys usually HAVE OS includes...)

        STRUCTURE IntuiMessage,0

         STRUCT im_ExecMessage,mn_SIZE

         LONG im_Class

         // For rtgmaster this is IDCMP_MOUSEBUTTONS or
         // IDCMP_RAWKEY, as RDCMP only supports mousebuttons
         // or keyboard events... look at this field to examine
         // which event happened...

         WORD im_Code

         // For Keyboard events, here the Rawkey value of the pressed
         // key is found... Bit 7 handles Keydown/Keyup, like usual...

         // For mouse : SELECTUP,SELECTDOWN,...

         WORD im_Qualifier

         // The Qualifiers for CTRL,SHIFT,... each qualifier has a bit...

         APTR im_IAddress

         WORD im_MouseX
         WORD im_MouseY

         // Mouseposition, when the event happened...

         LONG im_Seconds
         LONG im_Micros

         // systemtime, when the event happened

         APTR im_IDCMPWindow

         // Well, undefined for RDCMP, as rtgmaster not
         // always uses Intuition :)

         APTR im_SpecialLink

        // Well... usually undefined for RDCMP... don't acces
        // it...
};
```

```
        IDCMP_MOUSEBUTTONS EQU 8
        IDCMP_RAWKEY EQU 1024
SELECTUP        EQU     (IECODE_LBUTTON+IECODE_UP_PREFIX)
SELECTDOWN      EQU     (IECODE_LBUTTON)
MENUUP          EQU     (IECODE_RBUTTON+IECODE_UP_PREFIX)
MENUDOWN        EQU     (IECODE_RBUTTON)
MIDDLEUP        EQU     (IECODE_MBUTTON+IECODE_UP_PREFIX)
MIDDLEDOWN      EQU     (IECODE_MBUTTON)

out of devices/inputevents.i :

IECODE_LBUTTON                  EQU     $68     ; also uses IECODE_UP_PREFIX
IECODE_RBUTTON                  EQU     $69     ;
IECODE_MBUTTON                  EQU     $6A     ;

IEQUALIFIER_LSHIFT              EQU     $0001
IEQUALIFIER_RSHIFT              EQU     $0002
IEQUALIFIER_CAPSLOCK            EQU     $0004
IEQUALIFIER_CONTROL             EQU     $0008
IEQUALIFIER_LALT                EQU     $0010
IEQUALIFIER_RALT                EQU     $0020
IEQUALIFIER_LCOMMAND            EQU     $0040
IEQUALIFIER_RCOMMAND            EQU     $0080
IEQUALIFIER_MIDBUTTON           EQU     $1000
IEQUALIFIER_RBUTTON             EQU     $2000
IEQUALIFIER_LEFTBUTTON          EQU     $4000

but well, simply include the two OS includes, and you won't have to
bother about this stuff... :)

   INPUTS
        RtgScreen - The Screen, which port is to be used

   NOTES
       RDCMP supports both waiting and polling !!!
       As to my experiences, better use RDCMP than anything else...
       other methods i tried tended to lose mouseclicks, if
       they came very fast, and if the application took a lot
       of processing time. RDCMP does not lose data.
       Information about polling : See docs of RtgWaitRDCMP

   SEE ALSO

                 RtgWaitRDCMP()
                ,
                 RtgInitRDCMP()
                ,
                 RtgReplyMsg()
```

1.39 rtgmaster.library/RtgReplyMsg

```
                    NAME
        RtgReplyMsg - replies the message on an RDCMP
```

```
SYNOPSIS
     RtgReplyMsg(RtgScreen, imsg)
                 a0        a1

     void RtgReplyMsg(struct RtgScreen *,void *)

FUNCTION
     Replies the message on a RDCMP and tells the port, that the
     message can be deleted now. Save all values of the message that
     you need before this !!! Don't access the structure itself
     after the reply !!!

INPUTS
     RtgScreen - The Screen, which port is to be used
     imsg - the messages to be replied

NOTES
    DOES NOT RUN ON rtgPICA.library up to now !!!
    RDCMP supports both waiting and polling !!!
    As to my experiences, better use RDCMP than anything else...
    other methods i tried tended to lose mouseclicks, if
    they came very fast, and if the application took a lot
    of processing time. RDCMP does not lose data.

SEE ALSO

              RtgWaitRDCMP()
             ,
              RtgInitRDCMP()
             ,
              RtgGetMsg()
```

1.40 rtgmaster.library/RtgSetTextMode

```
                NAME
     RtgSetTextMode - sets text color and drawing mode

SYNOPSIS
     RtgSetTextMode(RtgScreen,fgcolor,bgcolor,drmode)
                    A0        D0      D1      D2

     void RtgSetTextMode(struct RtgScreen *,UBYTE,UBYTE,UBYTE)

FUNCTION
     Sets the foreground color, the background color and the drawing
     mode for Text on this RtgScreen.  drmodes are defined as usual
     in graphics/rastport.i (or .h) : JAM1, JAM2, COMPLEMENT.
     INVERSVID is not valid...

     This function should only be used on Displays with depth <=8.

INPUTS
     RtgScreen - an RtgScreen
     fgcolor   - Foreground color
     bgcolor   - Background color
     drwmode   - Drawing mode, defined in graphics/rastport.i (or .h)

NOTES
     Not yet implemented for rtgPICA.library

SEE ALSO
```

RtgOpenFont()
,
RtgSetFont()
,
RtgCloseFont()
,
RtgText()
,
RtgSetTextModeRGB()

1.41 rtgmaster.library/RtgSetTextModeRGB

```
           NAME
     RtgSetTextModeRGB - sets text color and drawing mode for depths >8

SYNOPSIS
     RtgSetTextModeRGB(RtgScreen,fgcolor,bgcolor,drmode)
                       A0        D0      D1      D2

     void RtgSetTextModeRGB(struct RtgScreen *,ULONG,ULONG,UBYTE)

FUNCTION
     Sets the foreground color, the background color and the drawing
     mode for Text on this RtgScreen.  drmodes are defined as usual
     in graphics/rastport.i (or .h) : JAM1, JAM2, COMPLEMENT.
     INVERSVID is not valid...

     Should only be used on Displays with depth >8.

NOTES
     Due to a bug in CyberGraphX, this function does not work
     very well with rtgCGX.library (strange colors...). Not yet
     implemented in rtgPICA.library.

INPUTS
     RtgScreen - an RtgScreen
     fgcolor   - Foreground color
     bgcolor   - Background color
     drwmode   - Drawing mode, defined in graphics/rastport.i (or .h)


SEE ALSO
```

RtgOpenFont()
,
RtgSetFont()

```
,
 RtgCloseFont()
,
 RtgText()
,
 RtgSetTextMode()
```

1.42 rtgmaster.library/RtgText

```
        NAME
     RtgText - displays Text on an RtgScreen

  SYNOPSIS
     RtgText(RtgScreen,buffer,string,length,xpos,ypos)
              A0        A1     A2     D0     D1   D2

     void RtgText(struct RtgScreen *,void *,char *,WORD,SHORT,SHORT)

  FUNCTION
     Displays the string "string" with the chosen font parameters (see
     RtgSetFont(), RtgSetTextMode(), RtgSetTextModeRGB() ) at position
     xpos,ypos much the same way like Text() of graphics.library does.

  INPUTS
     RtgScreen - an RtgScreen
     buffer    - The buffer address of the buffer where to display the text
     string    - pointer to a string
     length    - length of the string in characters
     xpos      - the x-pos
     ypos      - the y-pos

  NOTES
     Not yet implemented for rtgPICA.library.

  SEE ALSO

              RtgOpenFont()
             ,
              RtgSetFont()
             ,
              RtgCloseFont()
             ,
              RtgText()
             ,
              RtgSetTextModeRGB()

              RtgSetTextMode()
```

1.43 rtgmaster.library/RtgWaitRDCMP

```
                NAME
        RtgWaitRDCMP - Waits on a message on the RDCMP of the Screen

   SYNOPSIS
        RtgWaitRDCMP(RtgScreen)
                     a0

        void RtgWaitRDCMP(struct RtgScreen *)

   FUNCTION
        Waits on a message on the RDCMP. If you don't want your
        application to WAIT while the user is giving no inputs
        (well, it usually should NOT wait for games...), use polling :

        1. Open the Screen and init its RDCMP
        2. Inside the loop, do RtgGetMsg
        3. If imsg->Class is RDCMP_MOUSEBUTTONS or
           RDCMP_RAWKEY, handle the message and reply it
           using RtgReplyMsg
        4. Else don't reply it (if you reply a message when there
           in fact is NO MESSAGE, you might crash the system !!!)

   INPUTS
        RtgScreen - The Screen, which port is to be used

   NOTES
       DOES NOT RUN ON rtgPICA.library up to now !!!

   SEE ALSO

                 RtgInitRDCMP()
                ,
                 RtgGetMsg()
                ,
                 RtgReplyMsg()
```

1.44 rtgmaster.library/RtgWaitTOF

```
   NAME
        RtgWaitTOF - Wait for the top of the next video frame.

   SYNOPSIS
        WaitTOF(RtgScreen)
                A0

        WaitTOF(struct RtgScreen *)

   FUNCTION
        Wait  for vertical blank to occur and all vertical blank
        interrupt routines to complete before returning to caller.

        Does not do anything at all with some sublibraries... (CyberGraphX does  ←
           not
        support TOF-Waiting up to now...)

   INPUTS
        The Screen Handle of the GFX Board Screen (only in FACT needed for rtgEGS. ↩
           library,
        and probably nothing much is done about it anyway... but be nice... give  ↩
           this
        parameter... to stay compatible :) )

   RESULTS
        Places this task on the TOF wait queue. When the vertical blank
        interrupt comes around, the interrupt service routine will fire off
        signals to all the tasks doing WaitTOF. The highest priority task
        ready will get to run then.

   SEE ALSO
       graphics.library/WaitTOF()
```

1.45 rtgmaster.library/RunServer

```
                   NAME
        RunServer -- Handle all the messaging for a server and several clients

   SYNOPSIS
        New_Socket = RunServer(SBase,Socket,in_buffer,out_buffer,maxplayers)
         D0                    A0    A1     A2        A3         D0

        struct RTG_Socket *RunServer(struct Library *,struct RTG_Socket *,struct  ↩
           RTG_Buff *,struct RTG_Buff *,int)

   FUNCTION
        You will have to run this fine
        function in a loop. Every time it returns, it gives you the Socket of a  ↩
           new connected
        client or 0, if no new Client connected. Also, in  in_buffer, you will  ↩
           have all new
        messages sent from  already connected Clients to the server, and all  ↩
           messages you filled
        in in out_buffer before calling this function, will be sent to the Clients ↩
           .
        If nothing happened, this function returns at once, with consuming nearly  ↩
           no CPU time.

        You SHOULD initialize the in_buffer.num values with -1 EACH TIME, before
        RunServer is run, and the out_buffer.num values once before the FIRST TIME
        RunServer is called !!!

        NOTE: If you (later...) use RunServer, the Server can't handle a Player.  ↩
           You need
        ONE CLIENT PER PLAYER AND AN ADDITIONAL SERVER WITHOUT A PLAYER. The  ↩
           Server can run
        on a system, where a Client is also running, though (should be the fastest ↩
            system in
        the connection, probably, as it will have to do all that messaging to the  ↩
           Client ...)
```

```
      NOTE: It might appear strange to you, that you have to open bsdsocket. ↩
         library
      yourselves and provide it as parameter. This is needed because of some  ↩
         internal
      problems of AmiTCP, that make it IMPOSSIBLE opening it inside a library.  ↩
         Look
      at the Docs for more information.

      You do NOT have to use rtgmaster.library’s Graphics Board features to
      use rtgmaster.library’s TCP/IP features, if you do not WANT to...

      NOTE: It is NOT possible to give Socket->s (the Socket Number) of a Client
      to the Server using RtgRecv or RunServer !!! You will have to examine
      inbuffer->num[x] to find out which Socket was the Sender !!! Also len
      should NEVER be bigger than the actual message, or you might get a lot
      of strange results !!!

      NOTE: UP TO NOW (rtgmaster Version 7) ONLY SUPPORTS TCP... NO UDP SUPPORT  ↩
         UP TO NOW !!!

   INPUTS
      SBase      - Result of the call (C Syntax here...)
                   SBase = OpenLibrary("bsdsocket.library",0);
      Socket     - The Socket of THIS application (the Server...)
      in_buffer  - messages that arrived during the call of RunServer
      out_buffer - messages that Run_Server should deliver
      maxplayers - The Maximum of Clients allowed (CAN’T BE BIGGER THAN 12 !!!)

   RESULTS
      New_Socket - The Socket of a newly connected Client. Save it somewhere...

   SEE ALSO

               OpenClient()
              ,
               CloseClient()
              ,
               CloseServer()
              ,
               RunServer()
              ,
               RtgSend()

               RtgRecv()
              ,
               RtgIoctl()
              ,
               GetUDPName()
              ,
               RtgInAdr()
```

1.46 rtgmaster.library/SetSegment

```
                NAME
     SetSegment -- set the active segment

SYNOPSIS
     SetSegment(segnum)
                D0

     SetSegment(ULONG)

FUNCTION
     If the graphic board works in segment mode -- with a
     memory window of 64 KByte -- a call to SetSegment() sets
     the active segment to the supplied number.

     If the graphic board works non-segmented, a call to this
     function has no effect.

INPUTS
     segnum - number of segment

SEE ALSO

              GetSegment()
```

1.47 rtgmaster.library/SwitchScreens

```
                NAME
     SwitchScreens -- Perform doublebuffering

SYNOPSIS
     SwitchScreens(RtgScreen, Buffer)
                   A0         D0

     SwitchScreens(ULONG, ULONG)

FUNCTION
     RtgScreen passed in A0 is a handle of a screen previously opened
     by OpenRtgScreen().

     This functions is used to specify the buffer which should
     be displayed starting from the next Vertical Blank.  The buffer
     supplied is a simple number (0 = first buffer, 1 = 2nd buffer
     etcetera).

     If the same buffer is being specified as is being displayed then
     this function should do nothing.

     This function will never be called from interupts.

INPUTS
     RtgScreen - A handle for a valid screen previously opened by
                 this sublibrary's OpenRtgScreen() function.
     Buffer - The buffer number the user wishes to display
```

```
SEE ALSO

              OpenRtgScreen()
             ,
              WaitRtgSwitch()
```

1.48 rtgmaster.library/UnlockRtgScreen

```
                NAME
     UnlockRtgScreen -- Unlocks a RtgScreen

SYNOPSIS
     UnlockRtgScreen(RtgScreen)
                     A0

     UnlockRtgScreen(ULONG)

FUNCTION
     Unlocks a previously locked RtgScreen.  If this screen hasn't
     been locked before this function will do nothing.

     LockRtgScreen() and UnlockRtgScreen() functions nest, which means
     the user must call an UnlockRtgScreen() for every LockRtgScreen().
     If not the user will end up with a permenantly locked screen.

INPUTS
     RtgScreen - A handle for a valid screen previously opened by
                 this sublibrary's OpenRtgScreen() function.

SEE ALSO

              LockRtgScreen()
```

1.49 rtgmaster.library/WaitRtgBlit

```
                NAME
     WaitRtgBlit - Waits on the Blitter to be finished

SYNOPSIS
     WaitRtgBlit(RtgScreen)
                 A0

     WaitRtgBlit(struct RtgScreen *)

FUNCTION
     Waits for the GFX Board Blitter to be finished. For those who wonder,
     why RtgScreen has to be given as parameter... it probably won't be used...
     just to be on the sure side :)
```

```
          Does not do anything at all with some sublibraries. Some of them ALWAYS ↩
             wait...

     INPUTS
          RtgScreen - The RtgScreen structure

     SEE ALSO

                   OpenRtgScreen()
                  ,
                   RtgBlit()
```

1.50 rtgmaster.library/WaitRtgSwitch

```
                     NAME
          WaitRtgSwitch - Waits on Doublebuffering having happened

     SYNOPSIS
          WaitRtgSwitch(RtgScreen)
                        A0

          WaitRtgSwitch(struct RtgScreen *)

     FUNCTION
          As SwitchScreens does not wait till the Doublebuffering has happened,
          but returns AT ONCE, it might be that the program wants to access the
          video memory BEFORE the change has happened. In this case you can use
          WaitRtgSwitch to be sure the change really happened. If you do not
          want to wait, simply do not use this call :)

          Does not do much for some sublibraries. Some always wait...

     INPUTS
          The Screenhandle of the Screen, where the Wait should happen...

     SEE ALSO

                   SwitchScreens()
```

1.51 rtgmaster.library/WriteRtgPixel

```
                     NAME
          WriteRtgPixel - plots a single pixel to a RtgScreen

     SYNOPSIS
          WriteRtgPixel(RtgScreen, BufferAdr, XPos, YPos, Color)
                           A0          A1          D0    D1    D2

          WriteRtgPixel(struct RtgScreen *, APTR, ULONG, ULONG, UBYTE)
```

```
FUNCTION
     Draws a single pixel at the specified position on a RtgScreen.
     The BufferAdr is the starting address of the buffer the users wants
     to draw the pixel in.  The user has obtained this address using
     LockRtgScreen() and GetBufAdr().  The BufferAdr is needed to specify
     the correct buffer for screens which are double or triple buffered.

     This function should only work for Palette mapped modes, Color is
     the Color number of the palette.

     This function is not supported by rtgAMI.library

     DO NOT USE FOR SPEED RELEVANT STUFF. THIS FUNCTION MAY HAVE
     SOME OVERHEAD. FOR FAST OPERATIONS USE COPYRTGPIXELARRAY OR
     DO THE STUFF YOURSELVES.

INPUTS
     RtgScreen - A handle for a valid screen previously opened by
                 this sublibrary's OpenRtgScreen() function.
     BufferAdr - The address of the memory containing the actual
                 screen graphics
     XPos - X position of the pixel the user wants to plot
     YPos - Y position of the pixel the user wants to plot
     Color - Color number

SEE ALSO

              OpenRtgScreen()
             ,
              WriteRtgPixelRGB()
             ,
              WriteRtgPixelArray()
```

1.52 rtgmaster.library/WriteRtgPixelArray

```
               NAME
     WriteRtgPixelArray - writes an array of pixels to a RtgScreen

SYNOPSIS
     WriteRtgPixelArray(RtgScreen, BufferAdr, Array, Left, Top, Width, Height)
                        A0         A1         A2     D0    D1   D2     D3

     WriteRtgPixelArray(struct RtgScreen *, APTR, APTR, ULONG, ULONG, ULONG,  ↩
        ULONG)

FUNCTION
     Draws an rectangular array of pixels to the specified position on a
     RtgScreen.  The BufferAdr is the starting address of the buffer the
     user wants to draw this array of pixels in.  The user has obtained
     this address using LockRtgScreen() and GetBufAdr().  The BufferAdr is
     needed to specify the correct buffer for screens which are double or
     triple buffered.

     This function should only work for Palette mapped modes.  The array
```

```
      consists of one byte per pixel, each byte specifying a Color number.

      This function is many times faster than writing each pixel seperately
      to the screen using WriteRtgPixel().

      This function is not supported by rtgAMI.library

      DO NOT USE FOR SPEED RELEVANT STUFF. THIS FUNCTION MAY HAVE
      SOME OVERHEAD. FOR FAST OPERATIONS USE COPYRTGPIXELARRAY OR
      DO THE STUFF YOURSELVES.

   INPUTS
      RtgScreen - A handle for a valid screen previously opened by
                  this sublibrary's OpenRtgScreen() function.
      BufferAdr - The address of the memory containing the actual
                  screen graphics
      Array - Pointer to an array of pixels which is Width pixels wide,
              and Height pixels high.  Each pixel is one byte in size.
      Left - X position of the top-left of the rectangular pixel array
      Top - Y position of the top-left of the rectangular pixel array
      Width - Width of the array in pixels
      Height - Height of the array in pixels

   SEE ALSO

               OpenRtgScreen()
              ,
               WriteRtgPixel()
              ,WriteRtgPixelRGBArray()
```

1.53 rtgmaster.library/WriteRtgPixelRGB

```
                NAME
      WriteRtgPixelRGB - plots a single pixel to a RtgScreen

   SYNOPSIS
      WriteRtgPixelRGB(RtgScreen, BufferAdr, XPos, YPos, Color)
                       A0         A1         D0    D1    D2

      WriteRtgPixelRGB(struct RtgScreen *, APTR, ULONG, ULONG, ULONG)

   FUNCTION
      Draws a single pixel at the specified position on a RtgScreen.
      The BufferAdr is the starting address of the buffer the users wants
      to draw the pixel in.  The user has obtained this address using
      LockRtgScreen() and GetBufAdr().  The BufferAdr is needed to specify
      the correct buffer for screens which are double or triple buffered.

      This function should only work for True Color modes, Color is
      a 32 bit value which specifies what Color the pixel should be.
      The layout of this 32-bit value is as follows:
```

```
    %aaaaaaaa.rrrrrrrr.gggggggg.bbbbbbbb

    a = AlphaChannel (8-bits) which may or may not be ignored.  The
        user will set this to zero if the user doesn't want to use
        AlphaChannel.
    r = Red component (8-bits) of the 24-bit RGB value
    g = Green component (8-bits) of the 24-bit RGB value
    b = Blue component (8-bits) of the 24-bit RGB value

    DO NOT USE FOR SPEED RELEVANT STUFF. THIS FUNCTION MAY HAVE
    SOME OVERHEAD. FOR FAST OPERATIONS USE COPYRTGPIXELARRAY OR
    DO THE STUFF YOURSELVES.

INPUTS
    RtgScreen - A handle for a valid screen previously opened by
                this sublibrary's OpenRtgScreen() function.
    BufferAdr - The address of the memory containing the actual
                screen graphics
    XPos - X position of the pixel the user wants to plot
    YPos - Y position of the pixel the user wants to plot
    Color - A 32-bit value describing the color (see above)

SEE ALSO
```

OpenRtgScreen()
,
WriteRtgPixel()
,WriteRtgPixelRGBArray()