ReqTools

COLLABORATORS			
	TITLE : ReqTools		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		December 6, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ReqTools

1.1 Table Of Contents

1.2 rtallocrequesta()

```
  NAME  rtAllocRequestA()

    req = rtAllocRequestA (type, taglist);

    APTR rtAllocRequestA (ULONG, struct TagItem *);
    D0        D0   A0

    req = rtAllocRequest (type, tag1,...);
```

```
  APTR rtAllocRequest (ULONG, Tag,...);

DESCRIPTION
  Allocates a requester structure for you in a future compatible manner.
  This is the only way to properly allocate a rtFileRequester,
  rtFontRequester or rtReqInfo structure.  The structure will be
  initialized for you.

  Use rtFreeRequest() to free the requester structure when you no longer
  need it.

INPUTS
  type    - type of structure to allocate, currently RT_REQINFO,
      RT_FILEREQ, RT_FONTREQ or RT_SCREENMODEREQ.
  taglist - pointer to array of tags (currently always NULL).

TAGS
  no tags defined yet

RESULT
  req - pointer to the requester allocated or NULL if no memory.

BUGS
  none known

SEE ALSO
  rtFreeRequest()
```

1.3 rtchangereqattra()

```
NAME  rtChangeReqAttrA()

  [long =] rtChangeReqAttrA (req, taglist);

  [LONG] rtChangeReqAttrA (APTR, struct TagItem *);
        A1    A0

  [long =] rtChangeReqAttr (req, tag1,...);

  [LONG] rtChangeReqAttr (APTR, Tag,...);

DESCRIPTION
  Change requester attributes with supplied taglist.  This is the only
  correct way to change the attributes listed below.

  The return code from rtChangeReqAttrA() should be ignored unless stated
  otherwise.

  Don't pass the tags listed below to the requester itself (unless
  documented otherwise).  They will not be recognized.

INPUTS
  req     - pointer to requester.
  taglist - pointer to array of tags.
```

```
TAGS
  for the file requester:
RTFI_Dir   - (char *)
       Name of new directory to position file requester
       in. The requester's buffer will be deallocated.
RTFI_MatchPat  - (char *)
       New pattern string to match files on.
RTFI_AddEntry  - (BPTR)
       THIS *MUST* BE THE LAST TAG~(just before TAG_END)!
       Tagdata must hold a lock on a file or directory
       you want to add to the file requester's buffer.
       The lock should have been obtained using Lock(),
       and you must unlock this lock yourself.
       It is your responsibility to make sure the file
       or directory is indeed in the directory the file
       requester is in.
       If the entry is already in the file requester's
       buffer it will simply be updated.
       It is harmless to use this tag if the requester's
       buffer is not initialized. rtChangeReqAttr() will
       return a boolean to indicate success or failure
       (out of memory).
RTFI_RemoveEntry - (char *)
       Name of file or directory you want to remove from
       the file requester's buffer.
       It is your responsibility to make sure the file
       or directory is indeed in the directory the file
       requester is in.
       It is harmless use this tag if the requester's
       buffer is not initialized.
  for the font requester:
RTFO_FontName  - (char *)
       Set the name of the currently selected font.
RTFO_FontHeight  - (UWORD)
       Set the fontsize of the currently selected font.
RTFO_FontStyle   - (UBYTE)
       Set the style of the current font.
RTFO_FontFlags   - (UBYTE)
       Set the flags of the current font.
  for the screenmode requester [V38]:
RTSC_ModeFromScreen - (struct Screen *)
          Screen to get mode attributes from.
          NOTE: You must make sure the mode this screen
          is in will be accepted by the screen
          mode requester.  Otherwise it will auto-
          matically cancel.  For example, you use
          RTDI_ModeFromScreen on a HAM screen and
          you haven't set the SCREQF_NONSTDMODES
          flag.
RTSC_DisplayID      - (ULONG)
          Set 32-bit mode id of selected mode.  The width
          and height will be set to the default (visible)
          width and height, and the depth will be set to
          maximum.  Also read note above.
RTSC_DisplayWidth   - (UWORD)
          Set width of display.  Must come after
          RTSC_DisplayID or RTSC_ModeFromScreen tags.
```

```
RTSC_DisplayHeight  - (UWORD)
          Set height of display.  Must come after
          RTSC_DisplayID or RTSC_ModeFromScreen tags.
RTSC_DisplayDepth   - (UWORD)
          Set depth of display.  Must come after
          RTSC_DisplayID or RTSC_ModeFromScreen tags.
RTSC_OverscanType   - (ULONG)
          Set type of overscan.  Set to 0 for regular
          size, otherwise use OSCAN_... constants.
          See 'intuition/screens.[h|i]'.
RTSC_AutoScroll     - (BOOL)
          Boolean state of autoscroll checkbox.

RESULT
  none (except when RTFI_AddEntry tag is used, see above)

BUGS
  none known

SEE ALSO
  Lock()
```

1.4 rtclosewindowsafely()

```
NAME  rtCloseWindowSafely() [V38]

  rtCloseWindowSafely (window);

  void rtCloseWindowSafely (struct Window *);
          A0

DESCRIPTION
  Closes a window which shares its IDCMP port with another window.  All the
  pending messages (concerning this window) on the port will be removed and
  the window will be closed.

  Do not use this function to close windows which have an IDCMP port set up
  by Intuition.  If you do the port will be left in memory!

  If you intend to open a lot of windows all sharing the same IDCMP port it
  is easiest if you create a port yourself and open all windows with
  newwin.IDCMPFlags equal to 0 (this tells Intuition to NOT set up an IDCMP
  port).  After opening the window set the win->UserPort to your message
  port and call ModifyIDCMP to set your IDCMP flags.

  When you then receive messages from intuition check the imsg->IDCMPWindow
  field to find out what window they came from and act upon them.

  When closing your windows call rtCloseWindowSafely() for all of them and
  delete your message port.  Easy peasy :-)

INPUTS
  window - pointer to the window to be closed.

RESULT
```

```
  none

NOTE
  This function is for the advanced ReqTools user.

BUGS
  none known

SEE ALSO
  intuition.library/CloseWindow()
```

1.5 rtezrequesta()

```
NAME  rtEZRequestA()

  ret = rtEZRequestA (bodyfmt, gadfmt, reqinfo, argarray, taglist);

  ULONG rtEZRequestA
  (char *, char *, struct rtReqInfo *, APTR, struct TagItem *);
  D0     A1  A2  A3        A4    A0

  ret = rtEZRequest (bodyfmt, gadfmt, reqinfo, taglist, arg1, arg2,...);

  ULONG rtEZRequest
    (char *, char *, struct rtReqInfo *, struct TagItem *,...);

  ret = rtEZRequestTags(bodyfmt, gadfmt, reqinfo, argarray, tag1,...);

  ULONG rtEZRequestTags(char *, char *, struct rtReqInfo *, APTR, Tag,...);

DESCRIPTION
  This function puts up a requester for you and waits for a response from
  the user. If the response is positive, this procedure returns TRUE.
  If the response is negative, this procedure returns FALSE.
  The function may also return an IDCMP flag or a value corresponding with
  one of other possible responses (see below).

  'gadfmt' may contain several possible responses.  Separate these
  responses by a '|'.  For example: "Yes|No", or 'Yes|Maybe|No".  The
  responses should be typed in the same order as they will appear on
  screen, from left to right.  There is no limit to the number of responses
  other than the width of the screen the requester will appear on.

  'bodyfmt' can contain newlines ('\n', ASCII 10). This will cause a new
  line to be started (surprise, surprise :-).
  You may also include 'printf' style formatting codes. The format
  arguments should be pointed to by 'argarray'.
  You can use formatting codes in 'gadfmt' as well.  The arguments for
  this format string should follow the ones for 'bodyfmt'.

  NOTE: The formatting is done by exec.library/RawDoFmt(), so be aware that
  to display a 32-bit integer argument you must use "%ld", not "%d",
  since RawDoFmt() is "word-oriented."

  The second and third function use a variable number of arguments. These
```

```
  functions can be found in 'reqtools[nb].lib'.
  The second function has the RawDoFmt arguments as variable args, the
  third the tags. If you need both this is what you can do:

  ...
     {
     ULONG tags[] = { RTEZ_ReqTitle, (ULONG)"mytitle", TAG_END };

     rtEZRequest ("String, num: %s, %ld", NULL, "Ok",
           (struct TagItem *)tags, "six", 6);
     }
  ...

  You can satisfy the requester with the following keyboard shortcuts:
    'Y' or Left Amiga 'V' for a positive response,
    ESC, 'N', 'R' or Left Amiga 'B' for a negative response.

  If EZREQF_NORETURNKEY is _not_ set (see RTEZ_Flags below) the RETURN key
  is also accepted as a shortcut for the positive response (can be changed
  using RTEZ_DefaultResponse, see below).  The response that will be
  selected when you press RETURN will be printed in bold.

  The EZREQF_LAMIGAQUAL flag should be used when you put up a requester
  for a destructive action (e.g. to delete something).  When it is set
  the keyboard shortcuts are limited to Left Amiga 'V' and 'B' so it is
  harder to accidently select something you will regret.
  Note that the RETURN and ESC key remain active!  To disable the RETURN
  key use the EZREQF_NORETURNKEY flag.  The ESC key cannot be disabled.

  You may pass a NULL for 'gadfmt', but make sure you know what you are
  doing.  Passing a NULL opens an EZRequester with NO responses, just a
  body text.  This implies the user has no means of "answering" this
  requester.  You must therefore use the RT_IDCMPFlags tag to allow some
  other events to end the requester (e.g. IDCMP_MOUSEBUTTONS,
  IDCMP_INACTIVEWINDOW,...) or you must make use of the ReqHandler feature.
  Using a requester handler you can end the requester by program control.
  This way you can e.g. put up a requester before you start loading a file
  and remove it after the file has been loaded.  Do not pass an empty
  string as 'gadfmt'!

  'reqinfo' can be used to customize the requester.  For greater control
  use the tags listed below.  The advantage of the rtReqInfo structure is
  that it is global, where tags have to be specified each function call.
  See libraries/reqtools.[hi] for a description of the rtReqInfo structure.

INPUTS
  bodyfmt  - requester body text, can be format string a la RawDoFmt().
  gadfmt   - text for gadgets (left to right, separated by '|') or NULL.
  argarray - pointer to array of arguments for format string(s).
  reqinfo  - pointer to a rtReqInfo structure allocated with
       rtAllocRequest() or NULL.
  taglist  - pointer to a TagItem array.

TAGS
  RT_Window     - (struct Window *)
        Window that will be used to find the screen to put the
        requester on.
```

```
      You *MUST* supply this if you are a task calling this
      function and not a process! This is because tasks
      don't have a pr_WindowPtr.
RT_IDCMPFlags   - (ULONG)
      Extra idcmp flags to return on.  If one these IDCMP
      flags causes the requester to abort the return code
      will equal the flag in question.
RT_ReqPos     - (ULONG)
      One of the following:
  REQPOS_POINTER    - requester appears where the mouse
          pointer is (default).
  REQPOS_CENTERSCR  - requester is centered on the
          screen.
  REQPOS_CENTERWIN  - requester is centered in the
          window (only works if the
          pr_WindowPtr of your process is
          valid or if you use RT_Window).
          If RT_Window is NULL the
          requester will be centered on
          the screen.
  REQPOS_TOPLEFTSCR - requester appears at the top left
          of the screen.
  REQPOS_TOPLEFTWIN - requester appears at the top left
          of the window (only works if the
          pr_WindowPtr of your process is
          valid or if you use RT_Window).
      The requester will always remain in the visible part of
      the screen, so if you use the Workbench 2.0 ScreenMode
      preferences editor to enlarge your Workbench screen and
      you scroll around, the requester will always appear in
      the part you can see.
      REQPOS_CENTERSCR and REQPOS_TOPLEFTSCR also apply to
      the visible part of the screen. So if you use one of
      these the requester will be appear in the center or the
      top left off what you can see of the screen as opposed
      to the entire screen.
      REQPOS_CENTERWIN and REQPOS_TOPLEFTWIN fall back to
      REQPOS_CENTERSCR or REQPOS_TOPLEFTSCR respectively
      when there is no parent window.  So you can safely use
      these without worrying about the existence of a window.
RT_LeftOffset   - (ULONG)
      Offset of left edge of requester relative to position
      specified with RT_ReqPos (does not offset the requester
      when RT_ReqPos is REQPOS_POINTER).
RT_TopOffset    - (ULONG)
      Offset of top edge of requester relative to position
      specified with RT_ReqPos (does not offset the requester
      when RT_ReqPos is REQPOS_POINTER).
RT_PubScrName   - (char *)
      Name of public screen requester should appear on. When
      this tag is used the RT_Window tag will be ignored.
      If the public screen is not found the requester will
      open on the default public screen.
      Only works on Kickstart 2.0!  reqtools.library does
      not check this, it is up to you *NOT* to use this tag
      on Kickstart 1.3 or below!
      Note that the 1.3 version of reqtools.library also
```

understands and supports this tag (on 2.0).

RT_Screen - (struct Screen *)
Address of screen to put requester on. You should never use this, use RT_Window or RT_PubScrName.

RT_ReqHandler - (struct rtHandlerInfo **)
Using this tag you can start an "asynchronous" requester. ti_TagData of the tag must hold the address of a pointer variable to a rtHandlerInfo structure. The requester will initialize this pointer and will return immediately after its normal initialization. The return code will not be what you would normally expect. If the return code is _not_ equal to CALL_HANDLER an error occurred and you should take appropriate steps. If the return code was CALL_HANDLER everything went ok and the requester will still be up! See the explanation for rtReqHandlerA() below for the following steps you have to take.

RT_WaitPointer - (BOOL)
If this is TRUE the window calling the requester will get a standard wait pointer set while the requester is up. This will happen if you used the RT_Window tag or if your process's pr_WindowPtr is valid. Note that after the requester has finished your window will be ClearPointer()-ed. If you used a custom pointer in your window you will have to re-set it, or not use the RT_WaitPointer tag and put up a wait pointer yourself. If your program requires ReqTools V38 it is advised you use RT_LockWindow instead. Defaults to FALSE.

RT_LockWindow - (BOOL) [V38]
If this is TRUE the window calling the requester will get locked. It will no longer accept any user input and it will get standard wait pointer set. This will happen only if you used the RT_Window tag or if your process's pr_WindowPtr is valid. RT_LockWindow will restore a custom pointer if you have used one (unlike RT_WaitPointer). So you do not have to worry about having to restore it yourself. It is advised you use this tag as much as possible. Defaults to FALSE.

RT_ScreenToFront - (BOOL) [V38]
Boolean indicating whether to pop the screen the requester will appear on to the front. Default is TRUE.

RT_ShareIDCMP - (BOOL) [V38]
Boolean indicating whether to share the IDCMP port of the parent window. Use this tag together with the RT_Window tag to indicate the window to share IDCMP with. Sharing the IDCMP port produces less overhead, so it is advised you use this tag. Defaults to FALSE.

RT_Locale - (struct Locale *) [V38]
Locale to determine what language to use for the requester text. If this tag is not used or its data is NULL, the system's current default locale will be used. Default NULL.

RT_IntuiMsgFunc - (struct Hook *) [V38]
The requester will call this hook for each IDCMP message it gets that doesn't belong to its window. Only applies if you used the RT_ShareIDCMP tag to share the IDCMP port with the parent window. Parameters are

```
        as follows:
    A0 - (struct Hook *) your hook
    A2 - (struct rtReqInfo *) your requester info
    A1 - (struct IntuiMessage *) the message
        After you have finished examining the message and your
        hook returns, ReqTools will reply the message.  So do
        not reply the message yourself!
  RT_Underscore   - (char) [V38]
        Indicates the symbol that precedes the character in the
        gadget label to be underscored.  This is to define a
        keyboard shortcut for this gadget.  Example: to define
        the key 'Q' as a keyboard shortcut for "Quit" and 'N'
        for "Oh, No!" you would use the tag RT_Underscore, '_'
        and pass as gadfmt "_Quit|Oh, _No!".  Do not use the
        symbol '%' as it is used for string formatting.  The
        usual character to use is '_' like in the example.
        IMPORTANT: the shortcuts defined using RT_Underscore
        take precedence of the default shortcuts!  It is for
        example not wise to use a 'N' for a positive response!
        Pick your shortcuts carefully!
  RT_TextAttr     - (struct TextAttr *) [V38]
        Use this font for the requester.  Default is to use the
        screen font.  Note that the font must already be
        opened by you.  ReqTools will call OpenFont() on this
        TextAttr, _not_ OpenDiskFont()!  If the font cannot be
        opened using OpenFont() the default screen font will
        be used.
  RTEZ_ReqTitle   - (char *)
        Title of requester window, default is "Request" unless
        the requester has less than 2 responses, then the
        default title is "Information".
  RTEZ_Flags      - (ULONG)
        Flags for rtEZRequestA():
    EZREQF_NORETURNKEY - turn off the RETURN key as
            shortcut for positive response.
    EZREQF_LAMIGAQUAL  - keyboard shortcuts are limited
            to Left Amiga 'V' and 'B', ESC
            and RETURN.
    EZREQF_CENTERTEXT  - centers each line of body text
            in the requester window. Useful
            for about requesters.
  RTEZ_DefaultResponse - (ULONG)
        Response value that will be returned when the user
        presses the return key.  Will be ignored if the
        EZREQF_NORETURNKEY flag is set.  The text for this
        response will be printed in bold. Default is 1.

RESULT
  ret - 1 (TRUE) for leftmost (positive) response, then each consecutive
  response will return 1 more, the rightmost (false) response will
  return 0 (FALSE), so 1,2,3,...,num-1,0 -- or idcmp flag.

NOTE
  Automatically adjusts the requester to the screen font.

  rtEZRequestA() checks the pr_WindowPtr of your process to find the
  screen to put the requester on.
```

```
BUGS
  none known

SEE ALSO
  exec.library/RawDoFmt(), rtReqHandlerA()
```

1.6 rtfilerequesta()

```
NAME  rtFileRequestA()

  ret = rtFileRequestA (filereq, filename, title, taglist);

  APTR rtFileRequestA
  (struct rtFileRequester *, char *, char *, struct TagItem *);
  D0     A1        A2    A3    A0

  ret = rtFileRequest (filereq, filename, title, tag1,...);

  APTR rtFileRequest (struct rtFileRequester *, char *, char *, Tag,...);

DESCRIPTION
  Get a directory and filename(s), or just a directory from the user.

  'filename' should point to an array of at least 108 chars.  The filename
  already in 'filename' will be displayed in the requester when it comes
  up.  When the requester returns 'filename' will probably have changed.

  Using certain tags may result in the calling of a caller-supplied hook.

  The hook will be called with A0 holding the address of your hook
  structure (you may use the h_Data field to your own liking), A2 a pointer
  to the requester structure calling the hook ('req') and A1 a pointer to
  an object.  The object is variable and depends on what your hook is for.

  This is an example of a hook suitable to be used with the RTFI_FilterFunc
  tag:

  SAS/C users can define their function thus:

  BOOL __asm __saveds filterfunc (register __a0 struct Hook *filterhook,
     register __a2 struct rtFileRequester *req,
     register __a1 struct FileInfoBlock *fib)
  {
     BOOL accepted = TRUE;

     /* examine fib to decide if you want this file in the requester */
     ...
     return (accepted);
  }

  Your hook structure should then be initialized like this:

     filterhook->h_Entry = filterfunc;
     /* in this case no need to initialize hook->h_SubEntry */
```

```
      filterhook->h_Data = your_userdata_if_needed;

    You can also use a stub written in machine code to call
    your function. (see 'utility/hooks.h')

  INPUTS
    filereq  - pointer to a struct rtFileRequester allocated with
         rtAllocRequestA().
    filename - pointer to an array of chars (must be 108 bytes big).
    title    - pointer to requester window title (null terminated).
    taglist  - pointer to a TagItem array.

  TAGS
    RT_Window        - see rtEZRequestA()
    RT_ReqPos        - see rtEZRequestA()
    RT_LeftOffset      - see rtEZRequestA()
    RT_TopOffset       - see rtEZRequestA()
    RT_PubScrName      - see rtEZRequestA()
    RT_Screen        - see rtEZRequestA()
    RT_ReqHandler      - see rtEZRequestA()
    RT_WaitPointer     - see rtEZRequestA()
    RT_LockWindow      - [V38] see rtEZRequestA()
    RT_ScreenToFront   - [V38] see rtEZRequestA()
    RT_ShareIDCMP      - [V38] see rtEZRequestA()
    RT_Locale        - [V38] see rtEZRequestA()
    RT_IntuiMsgFunc    - (struct Hook *) [V38]
       The requester will call this hook for each IDCMP
       message it gets that doesn't belong to its window.
       Only applies if you used the RT_ShareIDCMP tag to
       share the IDCMP port with the parent window.
       Parameters are as follows:
         A0 - (struct Hook *) your hook
         A2 - (struct rtFileRequester *) your requester
         A1 - (struct IntuiMessage *) the message
       After you have finished examining the message and
       your hook returns, ReqTools will reply the message.
       So do not reply the message yourself!
    RT_Underscore      - (char) [V38]
       Indicates the symbol that precedes the character in
       a gadget's label to be underscored.  This will also
       define the keyboard shortcut for this gadget.
       Currently only needed for RTFI_OkText.  Usually set
       to '_'.
    RT_DefaultFont     - (struct TextFont *)
       This tag allows you to specify the font to be used
       in the requester when the screen font is
       proportional. Default is GfxBase->DefaultFont.
    RT_TextAttr        - (struct TextAttr *) [V38]
       Use this font for the requester.  Must be a fixed
       width font, _not_ a proportional one.  Default is to
       use the screen font or the default font (if the
       screen font is proportional).  Note that the font
       must already be opened by you.  ReqTools will call
       OpenFont() on this TextAttr, _not_ OpenDiskFont()!
       If the font cannot be opened using OpenFont() or if
       the font is proportional the default screen font
       will be used (or the font set with RT_DefaultFont).
```

```
RTFI_Flags         - (ULONG)
   Several flags:
     FREQF_NOBUFFER    - do _not_ use a buffer to
             remember directory contents
             for the next time the file
             requester is used.
     FREQF_MULTISELECT - allow multiple files to be
             selected. rtFileRequest() will
             return a pointer to an
             rtFileList structure which
             will contain all selected
             files.  Use rtFreeFileList()
             to free the memory used by
             this file list.
     FREQF_SELECTDIRS  - set this flag if you wish to
             enable the selecting of dirs
             as well as files.  You *must*
             also set FREQF_MULTISELECT.
             Directories will be returned
             together with files in
             rtFileList, but with StrLen
             equal to -1.  If you need the
             length of the directory's name
             use strlen().
     FREQF_SAVE      - Set this if you are using the
             requester to save or delete
             something.  Double-clicking
             will be disabled so it is
             harder to make a mistake and
             select the wrong file.  If the
             user enters a non-existent
             directory in the drawer string
             gadget, a requester will
             appear asking if the directory
             should be created.
     FREQF_NOFILES     - Set this if you want to use
             the requester to allow the
             user to select a directory
             rather than a file.  Ideal for
             getting a destination dir. May
             be used with FREQF_MULTISELECT
             and FREQF_SELECTDIRS.
     FREQF_PATGAD      - When this is set a pattern
             gadget will be added to the
             requester.
RTFI_Height        - (ULONG)
   Suggested height of file requester window.
RTFI_OkText        - (char *)
   Replacement text for "Ok" gadget, max 6 chars long.
RTFI_VolumeRequest - (ULONG) [V38]
   The presence of this tag turns the file requester
   into a volume/assign disk requester.  This requester
   can be used to get a device name ("DF0:", "DH1:",..)
   or an assign ("C:", "FONTS:",...) from the user.
   The result of this requester can be found in the
   filereq->Dir field.  The volume can also be changed
   with rtChangeReqAttrA() and the RTFI_Dir tag.  Note
```

```
      that the user may edit the disk/assign names, or
      enter a new one.  Note also that the real device
      name is returned, not the name of the volume in the
      device.  For example "DH1:", not "Hard1:".
      The tag data (ULONG) is used to set following flags:
        VREQF_NOASSIGNS - Do not include the assigns in
              the list, only the real devices.
        VREQF_NODISKS   - Do not include devices, just
              show the assigns.
        VREQF_ALLDISKS  - Show _all_ devices.  Default
              behavior is to show only those
              devices which have valid disks
              inserted into them.  So if you
              have no disk in drive DF0: it
              will not show up.  Set this flag
              if you do want these devices
              included.
      NOTE: Do *NOT* use { RTFI_VolumeRequest, TRUE }!
            You are then setting the VREQF_NOASSIGNS flag!
            Use { RTFI_VolumeRequest, 0 } for a normal
            volume requester.
      NOTE: If you use the RTFI_FilterFunc described
            below the third parameter will be a pointer
            to a rtVolumeEntry structure rather than a
            pointer to a FileInfoBlock structure!
            Tech note: the DOS device list has been
            unlocked, so it is safe to e.g. Lock() this
            device and call Info() on this lock.
    RTFI_FilterFunc    - (struct Hook *) [V38]
      Call this hook for each file in the directory being
      read (or for each entry in the volume requester).
      Parameters are as follows:
        A0 - (struct Hook *) your hook
        A2 - (struct rtFileRequester *) your filereq
        A1 - (struct FileInfoBlock *) fib of file OR
       (struct rtVolumeEntry *) device or assign
       in case of a volume requester.
      If your hook returns TRUE the file will be accepted.
      If it returns FALSE the file will be skipped and
      will not appear in the requester.
      IMPORTANT NOTE: If you change your hook's behavior
          you _MUST_ purge the requester's
          buffer (using rtFreeReqBuffer())!
      IMPORTANT NOTE: When this callback hook is called
          from a volume requester the
          pr_WindowPtr of your process will
          be set to -1 so *no* DOS requesters
          will appear when an error occurs!
    RTFI_AllowEmpty    - (BOOL) [V38]
      If RTFI_AllowEmpty is TRUE an empty file string will
      also be accepted and returned.  Defaults to FALSE,
      meaning that if the user enters no filename the
      requester will be canceled.  You should use this tag
      as little as possible!

  RESULT
    ret - TRUE if the user selected a file (check 'filereq->Dir' for the
```

```
  directory and 'filename' for the filename) or FALSE if the
  requester was canceled -- or a pointer to a struct rtFileList
  (if FREQF_MULTISELECT was used).

NOTE
  You CANNOT call the file requester from a task because it uses DOS calls!

  Automatically adjusts the requester to the screen font.
  If the screen font is proportional the default font will be used.

  If the requester got too big for the screen because of a very large font,
  the topaz.font will be used.

  rtFileRequest() checks the pr_WindowPtr of your process to find the
  screen to put the requester on.

BUGS
  none known

SEE ALSO
```

1.7 rtfontrequesta()

```
NAME  rtFontRequestA()

  bool = rtFontRequestA (fontreq, title, taglist);

  BOOL rtFontRequestA (struct rtFontRequester *, char *, struct TagItem *);
  D0       A1        A3    A0

  bool = rtFontRequest (fontreq, title, tag1,...);

  BOOL rtFontRequest (struct rtFontRequester *, char *, Tag,...);

DESCRIPTION
  Let the user select a font and a style (optional).

INPUTS
  fontreq  - pointer to a struct rtFontRequester allocated with
       rtAllocRequestA().
  title    - pointer to requester window title (null terminated).
  taglist  - pointer to a TagItem array.

TAGS
  RT_Window       - see rtEZRequestA()
  RT_ReqPos       - see rtEZRequestA()
  RT_LeftOffset     - see rtEZRequestA()
  RT_TopOffset      - see rtEZRequestA()
  RT_PubScrName     - see rtEZRequestA()
  RT_Screen       - see rtEZRequestA()
  RT_ReqHandler     - see rtEZRequestA()
  RT_WaitPointer    - see rtEZRequestA()
  RT_LockWindow     - [V38] see rtEZRequestA()
  RT_ScreenToFront  - [V38] see rtEZRequestA()
  RT_ShareIDCMP     - [V38] see rtEZRequestA()
```

```
RT_Locale       - [V38] see rtEZRequestA()
RT_IntuiMsgFunc   - (struct Hook *) [V38]
  The requester will call this hook for each IDCMP
  message it gets that doesn't belong to its window.
  Only applies if you used the RT_ShareIDCMP tag to
  share the IDCMP port with the parent window.
  Parameters are as follows:
    A0 - (struct Hook *) your hook
    A2 - (struct rtFontRequester *) your requester
    A1 - (struct IntuiMessage *) the message
  After you have finished examining the message and
  your hook returns, ReqTools will reply the message.
  So do not reply the message yourself!
RT_Underscore     - (char) [V38]
  Indicates the symbol that precedes the character in
  a gadget's label to be underscored.  This will also
  define the keyboard shortcut for this gadget.
  Currently only needed for RTFO_OkText.  Usually set
  to '_'.
RT_DefaultFont    - (struct TextFont *)
  This tag allows you to specify the font to be used in
  the requester when the screen font is proportional.
  Default is GfxBase->DefaultFont.
RT_TextAttr       - [V38] see rtFileRequestA()
  Remember: font cannot be proportional!
RTFO_Flags        - (ULONG)
  Several flags:
    FREQF_NOBUFFER      - do not buffer the font list
        for subsequent calls to
        rtFontRequestA().
    FREQF_FIXEDWIDTH    - only show fixed-width fonts.
    FREQF_COLORFONTS    - show color fonts also.
    FREQF_CHANGEPALETTE - change the screen's palette
        to match that of a selected
        color font.
    FREQF_LEAVEPALETTE  - leave the palette as it is
        when exiting rtFontRequestA()
        Useful in combination with
        FREQF_CHANGEPALETTE.
    FREQF_SCALE       - allow fonts to be scaled
        when they don't exist in the
        requested size.
        (works on Kickstart 2.0 only,
         has no effect on 1.2/1.3).
    FREQF_STYLE       - include gadgets so the user
        may select the font's style.
RTFO_Height       - (ULONG)
  Suggested height of font requester window.
RTFO_OkText       - (char *)
  Replacement text for "Ok" gadget.  Maximum 6 chars.
  (7 is still ok, but not esthetically pleasing)
RTFO_SampleHeight - (ULONG)
  Height of font sample display in pixels (default 24).
RTFO_MinHeight    - (ULONG)
  Minimum font size displayed.
RTFO_MaxHeight    - (ULONG)
  Maximum font size displayed.
```

```
  RTFO_FilterFunc   - (struct Hook *) [V38]
    Call this hook for each available font.
    Parameters are as follows:
      A0 - (struct Hook *) your hook
      A2 - (struct rtFontRequester *) your filereq
      A1 - (struct TextAttr *) textattr of font
    If your hook returns TRUE the font will be accepted.
    If it returns FALSE the font will be skipped and
    will not appear in the requester.
    IMPORTANT NOTE:  If you change your hook's behavior
    you _MUST_ purge the requester's buffer (using
    rtFreeReqBuffer())!

RESULT
  bool - TRUE if the user selected a font (freq->Attr holds the font),
   FALSE if the requester was canceled.

NOTE
  You CANNOT call the font requester from a task because it may use DOS
  calls!

  Automatically adjusts the requester to the screen font.
  If the screen font is proportional the default font will be used.

  If the requester got too big for the screen because of a very large font,
  the topaz.font will be used.

  rtFontRequest() checks the pr_WindowPtr of your process to find the
  screen to put the requester on.

BUGS
  none known

SEE ALSO
```

1.8 rtfreefilelist()

```
NAME  rtFreeFileList()

  rtFreeFileList (filelist);

  void rtFreeFileList (struct rtFileList *);
     A0

DESCRIPTION
  Frees a filelist returned by rtFileRequest() when the FREQF_MULTISELECT
  flag was set.  Call this after you have scanned the filelist and you no
  longer need it.

INPUTS
  filelist - pointer to rtFileList structure, returned by rtFileRequest()
       (may be NULL).

RESULT
  none
```

```
BUGS
  none known

SEE ALSO
  rtFileRequest()
```

1.9 rtfreereqbuffer()

```
NAME  rtFreeReqBuffer()

  rtFreeReqBuffer (req);

  void rtFreeReqBuffer (APTR);
      A1

DESCRIPTION
  Frees the buffer associated with 'req'.  In case of a file requester this
  function will deallocate the directory buffer, in case of a font
  requester the font list.

  It is safe to call this function for requesters that have no buffer, so
  you may call this for all requesters to free as much memory as possible.

INPUTS
  req - pointer to requester.

RESULT
  none

BUGS
  none known

SEE ALSO
  rtFileRequest(), rtFontRequest()
```

1.10 rtfreerequest()

```
NAME  rtFreeRequest()

  rtFreeRequest (req);

  void rtFreeRequest (APTR);
    A1

DESCRIPTION
  Free requester structure previously allocated by rtAllocRequestA().
  This will also free all buffers associated with the requester, so there
  is no need to call rtFreeReqBuffer() first.

INPUTS
  req - pointer to requester (may be NULL).
```

```
RESULT
  none

BUGS
  none known

SEE ALSO
  rtAllocRequestA()
```

1.11 rtgetlonga()

```
NAME  rtGetLongA()

  ret = rtGetLongA (&longvar, title, reqinfo, taglist);

  ULONG rtGetLongA (ULONG *, char *, struct rtReqInfo *, struct TagItem *);
  D0          A1       A2      A3      A0

  ret = rtGetLong (&longvar, title, reqinfo, tag1,...);

  ULONG rtGetLong (ULONG *, char *, struct rtReqInfo *, Tag,...);

DESCRIPTION
  Puts up a requester to get a signed long (32-bit) number from the user.

  'reqinfo' can be used to customize the requester.  For greater control
  use the tags listed below.  The advantage of the rtReqInfo structure is
  that it is global, where tags have to be specified each function call.
  See libraries/reqtools.[hi] for a description of the rtReqInfo structure.

INPUTS
  &longvar - address of long (32 bit!) variable to hold result.
  title    - pointer to null terminated title of requester window.
  reqinfo  - pointer to a rtReqInfo structure allocated with
       rtAllocRequest() or NULL.
  taglist  - pointer to a TagItem array.

TAGS
  RT_Window      - see rtEZRequestA()
  RT_IDCMPFlags    - see rtEZRequestA()
  RT_ReqPos      - see rtEZRequestA()
  RT_LeftOffset    - see rtEZRequestA()
  RT_TopOffset     - see rtEZRequestA()
  RT_PubScrName    - see rtEZRequestA()
  RT_Screen      - see rtEZRequestA()
  RT_ReqHandler    - see rtEZRequestA()
  RT_WaitPointer   - see rtEZRequestA()
  RT_Underscore    - [V38] see rtEZRequestA()
         Only when you also use the RTGL_GadFmt tag.
  RT_LockWindow    - [V38] see rtEZRequestA()
  RT_ScreenToFront - [V38] see rtEZRequestA()
  RT_ShareIDCMP    - [V38] see rtEZRequestA()
  RT_Locale      - [V38] see rtEZRequestA()
  RT_IntuiMsgFunc  - [V38] see rtEZRequestA()
```

```
  RT_TextAttr      - [V38] see rtEZRequestA()
         Note that under 1.2/1.3 the string gadget's font
         will remain the screen font.
  RTGL_Min         - (ULONG)
         Minimum allowed value. If the user tries to enter a
         smaller value the requester will refuse to accept it.
  RTGL_Max         - (ULONG)
         Maximum allowed value, higher values are refused.
  RTGL_Width       - (ULONG)
         Width of requester window in pixels.  This is only a
         suggestion. rtGetLongA() will not go below a
         certain width.
  RTGL_ShowDefault - (BOOL)
         If this is TRUE (default) the value already in
         'longvar' will be displayed in the requester when it
         comes up. If set to FALSE the requester will be empty.
  RTGL_GadFmt      - (char *) [V38]
         Using this tag you can offer the user several
         responses.  See rtEZRequestA() for more information.
         Note that selecting this gadget is considered a
         positive response so the integer in the gadget is
         copied to '&longvar'.
  RTGL_GadFmtArgs  - (APTR) [V38]
         If you used formatting codes with RTGL_GadFmt use this
         tag to pass the arguments.
  RTGL_Invisible   - (BOOL) [V38]
         Using this tag you can switch on invisible typing.
         Very useful if you need to get something like a
         code number from the user.  It is strongly advised to
         use { RTGL_ShowDefault, FALSE } or the user may get
         very confused!  Default is FALSE.
  RTGL_Backfill    - (BOOL) [V38]
         Backfill requester window with pattern.  Default TRUE.
  RTGL_TextFmt     - (char *) [V38]
         Print these lines of text above the gadget in the
         requester.  Very useful to inform the user of what
         he should enter.  Most of the time you will also want
         to set the GLREQF_CENTERTEXT flag.  If you set the
         RTGL_Backfill tag to FALSE _no_ recessed border will
         be placed around the text.  Formatting codes may be
         used in the string (see RTGL_TextFmtArgs tag).
  RTGL_TextFmtArgs - (APTR) [V38]
         If you used formatting codes with RTGL_TextFmt use
         this tag to pass the arguments.
  RTGL_Flags       - (ULONG) [V38]
     GLREQF_CENTERTEXT - centers each line of text above
              gadget in the requester window.
              Should be generally set.

RESULT
  ret - TRUE if user entered a number, FALSE if not. If one of your idcmp
  flags caused the requester to end 'ret' will hold this flag.
  If you used the RTGL_GadFmt tag the return code will hold the
  value of the response as with rtEZRequestA().

NOTE
  'longvar' will NOT change if the requester is aborted.
```

```
  Automatically adjusts the requester to the screen font.

  rtGetLongA() checks the pr_WindowPtr of your process to find the
  screen to put the requester on.

BUGS
  none known

SEE ALSO
```

1.12 rtgetstringa()

```
NAME  rtGetStringA()

  ret = rtGetStringA (buffer, maxchars, title, reqinfo, taglist);

  ULONG rtGetStringA
      (UBYTE *, ULONG, char *, struct rtReqInfo *, struct TagItem *);
  D0         A1 D0     A2      A3      A0

  ret = rtGetString (buffer, maxchars, title, reqinfo, tag1,...);

  ULONG rtGetString (UBYTE *, ULONG, char *, struct rtReqInfo *, Tag,...);

DESCRIPTION
  Puts up a string requester to get a line of text from the user.
  The string present in 'buffer' upon entry will be displayed, ready to
  be edited.

  'reqinfo' can be used to customize the requester.  For greater control
  use the tags listed below.  The advantage of the rtReqInfo structure is
  that it is global, where tags have to be specified each function call.
  See libraries/reqtools.[hi] for a description of the rtReqInfo structure.

INPUTS
  buffer   - pointer to buffer to hold characters entered.
  maxchars - maximum number of characters that fit in buffer (EX-cluding
       the 0 to terminate the string !).
  title    - pointer to null terminated title of requester window.
  reqinfo  - pointer to a rtReqInfo structure allocated with
       rtAllocRequest() or NULL.
  taglist  - pointer to a TagItem array.

TAGS
  RT_Window      - see rtEZRequestA()
  RT_IDCMPFlags    - see rtEZRequestA()
  RT_ReqPos      - see rtEZRequestA()
  RT_LeftOffset    - see rtEZRequestA()
  RT_TopOffset     - see rtEZRequestA()
  RT_PubScrName    - see rtEZRequestA()
  RT_Screen      - see rtEZRequestA()
  RT_ReqHandler    - see rtEZRequestA()
  RT_WaitPointer   - see rtEZRequestA()
  RT_Underscore    - [V38] see rtEZRequestA()
```

```
        Only when you also use the RTGS_GadFmt tag.
   RT_LockWindow    - [V38] see rtEZRequestA()
   RT_ScreenToFront - [V38] see rtEZRequestA()
   RT_ShareIDCMP    - [V38] see rtEZRequestA()
   RT_Locale      - [V38] see rtEZRequestA()
   RT_IntuiMsgFunc  - [V38] see rtEZRequestA()
   RT_TextAttr      - [V38] see rtEZRequestA()
        Note that under 1.2/1.3 the string gadget's font
        will remain the screen font.
   RTGS_Width       - (ULONG)
        Width of requester window in pixels.  This is only a
        suggestion. rtGetStringA() will not go below a certain
        width.
   RTGS_AllowEmpty  - (BOOL)
        If RTGS_AllowEmpty is TRUE an empty string will also
        be accepted and returned.  Defaults to FALSE, meaning
        that if the user enters an empty string the requester
        will be canceled.
   RTGS_GadFmt      - (char *) [V38]
        Using this tag you can offer the user several
        responses.  See rtEZRequestA() for more information.
        Note that selecting this gadget is considered a
        positive response so the string in the gadget is
        copied to 'buffer'.
   RTGS_GadFmtArgs  - (APTR) [V38]
        If you used formatting codes with RTGS_GadFmt use this
        tag to pass the arguments.
   RTGS_Invisible   - (BOOL) [V38]
        Using this tag you can switch on invisible typing.
        Very useful if you need to get something like a
        password from the user.  It is strongly advised to use
        an empty initial string or the user may get very
        confused!  Default is FALSE.
   RTGS_Backfill    - (BOOL) [V38]
        Backfill requester window with pattern.  Default TRUE.
   RTGS_TextFmt     - (char *) [V38]
        Print these lines of text above the gadget in the
        requester.  Very useful to inform the user of what
        he should enter.  Most of the time you will also want
        to set the GSREQF_CENTERTEXT flag.  If you set the
        RTGS_Backfill tag to FALSE _no_ recessed border will
        be placed around the text.  Formatting codes may be
        used in the string (see RTGS_TextFmtArgs tag).
   RTGS_TextFmtArgs - (APTR) [V38]
        If you used formatting codes with RTGS_TextFmt use
        this tag to pass the arguments.
   RTGS_Flags       - (ULONG) [V38]
      GSREQF_CENTERTEXT - centers each line of text above
              gadget in the requester window.
              Should be generally set.

RESULT
   ret - TRUE if user entered something, FALSE if not. If one of your idcmp
   flags caused the requester to end 'ret' will hold this flag.
   If you used the RTGS_GadFmt tag the return code will hold the
   value of the response as with rtEZRequestA().
```

```
NOTE
  The contents of the buffer will NOT change if the requester is aborted.

  Automatically adjusts the requester to the screen font.

  rtGetStringA() checks the pr_WindowPtr of your process to find the
  screen to put the requester on.

BUGS
  none known

SEE ALSO
```

1.13 rtgetvscreensize()

```
NAME  rtGetVScreenSize()

  rtGetVScreenSize (screen, widthptr, heightptr);

  ULONG rtGetVScreenSize (struct Screen *, ULONG *, ULONG *);
  D0          A0         A1       A2

DESCRIPTION
  Use this function to get the size of the visible portion of a screen.

  The value returned by rtGetVScreenSize() can be used for vertical
  spacing.  It will be larger for interlaced and productivity screens.
  Using this number for spacing will assure your requester will look
  good on an interlaced and a non-interlaced screen.

  Current return codes are 2 for non-interlaced and 4 for interlaced.
  These values may change in the future, don't depend on them too much.
  They will in any case remain of the same magnitude.

INPUTS
  screen    - pointer to the screen.
  widthptr  - address of an ULONG variable to hold the width.
  heightptr - address of an ULONG variable to hold the height.

RESULT
  none

NOTE
  This function is for the advanced ReqTools user.

BUGS

SEE ALSO
```

1.14 rtlockwindow()

```
NAME  rtLockWindow() [V38]

  windowlock = rtLockWindow (window);

  APTR rtLockWindow (struct Window *);
  D0           A0

DESCRIPTION
  Lock a window so it will no longer accept any user input.  The only
  functions left to the user are depth arrangement and window dragging.
  All gadgets will be un-selectable and the window can not be resized.
  It will also get the standard wait pointer set.  The pointer at the
  time of locking will be restored when the window is unlocked.

  You may nest calls to rtLockWindow() and rtUnlockWindow().  Just make
  sure you unlock the window in the correct (opposite) order.

  See the RT_LockWindow tag for an automatic way of locking your window.

  Use this function (and rtUnlockWindow()) instead of rtSetWaitPointer().

INPUTS
  window - pointer to the window to be locked.

RESULT
  windowlock - a pointer to a (private) window lock.  You must pass this
   to rtUnlockWindow() to unlock the window again.
   Never mind if this is NULL.  This means there was not enough
   memory and the window will not be locked.  There is no
   sense in reporting this, just carry on and pass the NULL
   window lock to rtUnlockWindow().

NOTE
  The wait pointer will look exactly like the standard Workbench 2.0
  wait pointer.  In combination with PointerX, ClockTick or LacePointer
  the handle will turn.

BUGS
  none known

SEE ALSO
```

1.15 rtpalletterequesta()

```
NAME  rtPaletteRequestA()

  color = rtPaletteRequestA (title, reqinfo, taglist);

  LONG rtPaletteRequestA (char *, struct rtReqInfo *, struct TagItem *);
  D0          A2      A3      A0

  color = rtPaletteRequest (title, reqinfo, tag1,...);

  LONG rtPaletteRequest (char *, struct rtReqInfo *, Tag,...);
```

```
DESCRIPTION
  Put up a palette requester so the user can change the screen's colors.

  The colors are changed in the viewport of the screen the requester will
  appear on, so that is where you will find them after the palette
  requester returns.

  The selected color is returned, so you can also use this requester to let
  the user select a color.

  'reqinfo' can be used to customize the requester.  For greater control
  use the tags listed below.  The advantage of the rtReqInfo structure is
  that it is global, where tags have to be specified each function call.
  See libraries/reqtools.[hi] for a description of the rtReqInfo structure.

INPUTS
  title   - pointer to requester window title (null terminated).
  reqinfo - pointer to a rtReqInfo structure allocated with
      rtAllocRequest() or NULL.
  taglist - pointer to a TagItem array.

TAGS
  RT_Window      - see rtEZRequestA()
  RT_ReqPos      - see rtEZRequestA()
  RT_LeftOffset    - see rtEZRequestA()
  RT_TopOffset     - see rtEZRequestA()
  RT_PubScrName    - see rtEZRequestA()
  RT_Screen      - see rtEZRequestA()
  RT_ReqHandler    - see rtEZRequestA()
  RT_WaitPointer   - see rtEZRequestA()
  RT_LockWindow    - [V38] see rtEZRequestA()
  RT_ScreenToFront - [V38] see rtEZRequestA()
  RT_ShareIDCMP    - [V38] see rtEZRequestA()
  RT_Locale      - [V38] see rtEZRequestA()
  RT_IntuiMsgFunc  - [V38] see rtEZRequestA()
  RT_DefaultFont   - (struct TextFont *)
         This tag allows you to specify the font to be used in
         the requester when the screen font is proportional.
         Default is GfxBase->DefaultFont.
  RT_TextAttr      - [V38] see rtFileRequestA()
         Remember: font cannot be proportional!
  RTPA_Color       - (ULONG)
         Initially selected color of palette.  Default is 1.

RESULT
  color - the color number of the selected color or -1 if the user
    canceled the requester.

NOTE
  Automatically adjusts the requester to the screen font.
  If the screen font is proportional the default font will be used.

  If the requester got too big for the screen because of a very large font,
  the topaz.font will be used.

  rtPaletteRequestA() checks the pr_WindowPtr of your process to find the
```

```
    screen to put the requester on.

  BUGS
    none known

  SEE ALSO
```

1.16 rtreqhandlera()

```
  NAME  rtReqHandlerA()

    ret = rtReqHandlerA (handlerinfo, sigs, taglist);

    ULONG rtReqHandlerA (struct rtHandlerInfo *, ULONG, struct TagItem *);
    D0       A1      D0 A0

    ret = rtReqHandler (handlerinfo, sigs, tag1,...);

    ULONG rtReqHandler (struct rtHandlerInfo *, ULONG, Tag,...);

  DESCRIPTION
    This function should be called if you used the RT_ReqHandler tag with a
    requester function.

    The requester you used the tag with will have returned immediately after
    its initialization and will have initialized a pointer to a rtHandlerInfo
    structure for you.
    You should now do the following:

    Check the DoNotWait field. If it is FALSE you have to wait for the
    signals in the WaitMask field (plus your own signals if you like).
    If any of the signals in WaitMask are received or DoNotWait was not FALSE
    you have to call rtReqHandlerA() and check its return value for one of
    the following values:

      CALL_HANDLER    - Check DoNotWait again, Wait() if you have to
      and call rtReqHandlerA() again. In other words, loop.
      everything else - normal return value, requester has finished. This
      return value will be the same as if the requester
      had run normally.

    You must pass the signals you received to rtReqHandlerA().

    NOTE: if you want to wait for your own signals do not do so if
    DoNotWait is TRUE.  Call rtReqHandlerA() and if you must know
    if one of your signals arrived use SetSignal() to find this out.
    If you are waiting for a message to arrive at a message port you
    can simple call GetMsg() and check if it is non-null.
    DoNotWait will naturally only be TRUE when it absolutely,
    positively has to be.  A multitasking machine as the Amiga should
    use Wait() as much as possible.

    This is an example of a "requester loop":

    ...
```

```
  struct rtHandlerInfo *hinfo;
  ULONG ret, mymask, sigs;

  ...
     /* calculate our mask */
     mymask = 1 << win->UserPort->mp_SigBit;

     /* We use the RT_ReqHandler tag to cause the requester to return
  after initializing.
  Check the return value to see if this setup went ok. */
     if (rtFontRequest (req, "Font", RT_ReqHandler, &hinfo, TAG_END)
               == CALL_HANDLER) {
  do {
    /* Wait() if we can */
    if (!hinfo->DoNotWait)
       sigs = Wait (hinfo->WaitMask | mymask);

    /* check our own message port */
    while (msg = GetMsg (win->UserPort)) {
       ...
       /* here we handle messages received at our windows IDCMP         ...
    port */
       ...
       }

    /* let the requester do its thing (remember to pass 'sigs') */
    ret = rtReqHandler (hinfo, sigs, TAG_END);

    /* continue this loop as long as the requester is up */
    } while (ret == CALL_HANDLER)

  /* when we get here we know the requester has finished, 'ret'
     is the return code. */
  ...
  }
     else notify ("Error opening requester!");
  ...

INPUTS
  handlerinfo - pointer to handler info structure initialized by using
    the RT_ReqHandler tag when calling a requester function.
  sigs  - the signals received by previous wait, will be ignored if
    hinfo->DoNotWait was TRUE.
  taglist - pointer to a TagItem array.

TAGS
  RTRH_EndRequest - supplying this tag will end the requester. The return
        code from rtReqHandlerA() will _not_ be CALL_HANDLER,
        but the requester return code.  If the tagdata of this
        tag is REQ_CANCEL the requester will be canceled, if it
        is REQ_OK the requester will be ok-ed.
        In case of an EZRequest tagdata should be the return
        code of the requester (TRUE, FALSE or 2,3,4,...).

RESULT
  ret - CALL_HANDLER if you have to call rtReqHandlerA() again,
  or the normal return value from the requester.
```

```
BUGS
  none known

SEE ALSO
  rtEZRequest() (RT_ReqHandler explanation)
```

1.17 rtscreenmoderequesta()

```
NAME  rtScreenModeRequestA() [V38]

  ret = rtScreenModeRequestA (screenmodereq, title, taglist);

  BOOL rtScreenModeRequestA
    (struct rtScreenModeRequester *, char *, struct TagItem *);
  D0       A1          A3    A0

  ret = rtScreenModeRequest (screenmodereq, title, tag1,...);

  BOOL rtScreenModeRequest
       (struct rtScreenModeRequester *, char *, Tag,...);

DESCRIPTION

  *IMPORTANT* THIS REQUESTER IS ONLY AVAILABLE FROM KICKSTART 2.0 ONWARDS!
  The 1.3 version of ReqTools also contains the screenmode
  requester, but unless you are running 2.0 or higher it will
  not come up.  So what you essentially have to do is NOT
  call rtScreenModeRequestA() if your program is running on
  a machine with Kickstart 1.2/1.3.  You can safely call
  rtScreenModeRequestA() if you are running on a 2.0 machine,
  even if the user has installed the 1.3 version of ReqTools.

  Get a screen mode from the user.

  The user will be able to pick a screen mode by name, enter the size and
  the number of colors (bitplane depth).

  rtScreenModeRequestA() will call the appropriate 2.0 functions to get all
  the mode's information.  If no name has been assigned to the mode one
  will be constructed automatically.

INPUTS
  screenmodereq - pointer to a struct rtScreenModeRequester allocated with
      rtAllocRequestA().
  title   - pointer to requester window title (null terminated).
  taglist   - pointer to a TagItem array.

TAGS
  RT_Window        - see rtEZRequestA()
  RT_ReqPos        - see rtEZRequestA()
  RT_LeftOffset      - see rtEZRequestA()
  RT_TopOffset       - see rtEZRequestA()
  RT_PubScrName      - see rtEZRequestA()
  RT_Screen        - see rtEZRequestA()
```

```
RT_ReqHandler      - see rtEZRequestA()
RT_WaitPointer     - see rtEZRequestA()
RT_LockWindow      - see rtEZRequestA()
RT_ScreenToFront   - see rtEZRequestA()
RT_ShareIDCMP      - see rtEZRequestA()
RT_Locale        - see rtEZRequestA()
RT_IntuiMsgFunc    - (struct Hook *) [V38]
   The requester will call this hook for each IDCMP
   message it gets that doesn't belong to its window.
   Only applies if you used the RT_ShareIDCMP tag to
   share the IDCMP port with the parent window.
   Parameters are as follows:
     A0 - (struct Hook *) your hook
     A2 - (struct rtScreenModeRequester *) your req
     A1 - (struct IntuiMessage *) the message
   After you have finished examining the message and
   your hook returns, ReqTools will reply the message.
   So do not reply the message yourself!
RT_Underscore      - (char) [V38]
   Indicates the symbol that precedes the character in
   a gadget's label to be underscored.  This will also
   define the keyboard shortcut for this gadget.
   Currently only needed for RTSC_OkText.  Usually set
   to '_'.
RT_DefaultFont     - (struct TextFont *)
   This tag allows you to specify the font to be used
   in the requester when the screen font is
   proportional.  Default is GfxBase->DefaultFont.
RT_TextAttr        - [V38] see rtFileRequestA()
   Remember: font cannot be proportional!
RTSC_Flags         - (ULONG)
   Several flags:
     SCREQF_OVERSCANGAD - Add an overscan cycle
          gadget to the requester.
          After the requester returns
          you may read the overscan
          type in 'rq->OverscanType'
          If this is 0 no overscan is
          selected (Regular Size), if
          non-zero it holds one of
          the OSCAN_... values
          defined in the include file
          'intuition/screens.[h|i]'.
     SCREQF_AUTOSCROLLGAD - Add an autoscroll checkbox
          gadget to the requester.
          After the requester returns
          read 'smreq->AutoScroll' to
          see if the user prefers
          autoscroll to be on or off.
     SCREQF_SIZEGADS  - Add width and height
          gadgets to the requester.
          If you do not add these
          gadgets the width and
          height returned will be the
          default width and height
          for the selected overscan
          type.
```

```
      SCREQF_DEPTHGAD  - Add a depth slider gadget
           to the requester.  If you
           do not add a depth gadget,
           the depth returned will be
           the maximum depth this mode
           can be opened in.
      SCREQF_NONSTDMODES - Include all modes. Unless
           this flag is set
           rtScreenModeRequestA()
           will exclude nonstandard
           modes.  Nonstandard modes
           are presently HAM and EHB
           (ExtraHalfBrite). So unless
           you are picking a mode to
           do some rendering in leave
           this flag unset.  Without
           this flag set the mode
           returned will be a normal
           bitplaned mode.
      SCREQF_GUIMODES  - Set this flag if you are
           getting a screen mode to
           open a user interface
           screen in.  The modes
           shown will be standard
           modes with a high enough
           resolution (minumum 640
           pixels).  If this flag is
           set the SCREQF_NONSTDMODES
           flag is ignored.
RTSC_Height        - (ULONG)
   Suggested height of screenmode requester window.
RTSC_OkText        - (char *)
   Replacement text for "Ok" gadget, max 6 chars long.
RTSC_MinWidth      - (UWORD)
   The minimum display width allowed.
RTSC_MaxWidth      - (UWORD)
   The maximum display width allowed.
RTSC_MinHeight     - (UWORD)
   The minimum display height allowed.
RTSC_MaxHeight     - (UWORD)
   The maximum display height allowed.
RTSC_MinDepth      - (UWORD)
   The minimum display depth allowed.  Modes with a
   minimum display depth lower than this value will not
   be included in the list.
RTSC_MaxDepth      - (UWORD)
   The maximum display depth allowed.
RTSC_PropertyFlags - (ULONG)
   A mode must have these property flags to be
   included.  Only bits set in RTSC_PropertyMask are
   considered.
RTSC_PropertyMask  - (ULONG)
   Mask to apply to RTSC_PropertyFlags to determine
   which bits to consider.  See use of 'newsignals' and
   'signalmask' in exec.library/SetSignal().  Default
   is to consider all bits in RTSC_PropertyFlags as
   significant.
```

```
  RTSC_FilterFunc    - (struct Hook *)
     Call this hook for each display mode id in the
     system's list.
     Parameters are as follows:
       A0 - (struct Hook *) your hook
       A2 - (struct rtScreenModeRequester *) your req
       A1 - (ULONG) 32-bit extended mode id
     If your hook returns TRUE the mode will be accepted.
     If it returns FALSE the mode will be skipped and
     will not appear in the requester.

RESULT
  ret - FALSE if the requester was canceled or TRUE if the user selected a
  screen mode (check 'smreq->DisplayID' for the 32-bit extended
  display mode, 'smreq->DisplayWidth' and 'smreq->DisplayHeight' for
  the display size, 'smreq->DisplayDepth' for the screen's depth) or
  FALSE if the requester was canceled.

NOTE
  Automatically adjusts the requester to the screen font.
  If the screen font is proportional the default font will be used.

  If the requester got too big for the screen because of a very large font,
  the topaz.font will be used.

  rtScreenModeRequest() checks the pr_WindowPtr of your process to find the
  screen to put the requester on.

BUGS
  none known

SEE ALSO
  graphics/GetDisplayInfoData() graphics/displayinfo.h
  exec.library/SetSignal()
  Intuition/SA_DisplayID screen tag
```

1.18 rtscreentofrontsafely()

```
NAME  rtScreenToFrontSafely()

  rtScreenToFrontSafely (screen);

  void rtScreenToFrontSafely (struct Screen *);
       A0

DESCRIPTION
  Brings the specified screen to the front of the display, but only after
  checking it is still in the list of currently open screens.

  This function can be used to bring a screen back to the front of the
  display after bringing another screen to the front.  If the first screen
  closed while you where busy it is harmless to call this function, unlike
  calling the normal ScreenToFront().

INPUTS
```

```
  screen - pointer to the screen.

RESULT
  none

NOTE
  This function is for the advanced ReqTools user.

BUGS
  none known

SEE ALSO
  intuition.library/ScreenToFront()
```

1.19 rtsetreqposition()

```
NAME  rtSetReqPosition()

  rtSetReqPosition (reqpos, newwindow, screen, window);

  void rtSetReqPosition
       (ULONG, struct NewWindow *, struct Screen *, struct Window *);
  D0     A0      A1       A2

DESCRIPTION
  Sets newwindow->LeftEdge and newwindow->TopEdge according to reqpos.

  Except for the left- and topedge 'newwindow' must already be completely
  initialized.

  The newwindow->LeftEdge and newwindow->TopEdge already in the NewWindow
  structure will be used as offsets to the requested position.  If you'd
  like a window at position (25,18) from the top left of the screen you
  would fill newwindow->LeftEdge with 25, newwindow->TopEdge with 18 and
  call rtSetReqPosition() with reqpos equal to REQPOS_TOPLEFTSCR.

  Don't forget to make sure newwindow->LeftEdge and newwindow->TopEdge
  are 0 if you don't want to offset your window.

  In case of REQPOS_POINTER you can use them to point to your window's
  hotspot, where the pointer should point.  If you call rtSetReqPosition()
  with the left- and topedge equal to 0 you'd get a window appearing with
  its top- and leftedge equal to the current pointer position.

  Note that the screen pointer may _NOT_ be NULL.  If you have your own
  window open you can supply yourwindow->WScreen to this function.

  The window pointer is only required if reqpos is REQPOS_CENTERWIN or
  REQPOS_TOPLEFTWIN.  Even in this case you may call rtSetReqPosition()
  with a NULL window pointer.  The positions will simply fall back to
  REQPOS_CENTERSCR and REQPOS_TOPLEFTSCR respectively.

INPUTS
  reqpos    - one of the REQPOS_... constants usable with RT_ReqPos.
  newwindow - pointer to your (already initialized) NewWindow structure.
```

```
  screen    - pointer to screen the requester will appear on.
  window    - pointer to parent window or NULL.

RESULT
  none

NOTE
  This function is for the advanced ReqTools user.

BUGS
  none known

SEE ALSO
  RT_ReqPos tag
```

1.20 rtsetwaitpointer()

```
NAME  rtSetWaitPointer()

  rtSetWaitPointer (window);

  void rtSetWaitPointer (struct Window *);
       A0

DESCRIPTION
  Change the window's pointer image to that of a wait pointer.  Call this
  function whenever your program will be busy doing something for a lengthy
  period of time.

  It is recommended you call this function before calling any of the
  requester functions.  This way if the user clicks in your window he will
  know he must respond to the requester before doing anything else.  Also
  see the RT_WaitPointer tag for an automatic way of setting the wait
  pointer.  If you are using ReqTools V38+ check out the RT_LockWindow tag!

INPUTS
  window - pointer to the window to receive the wait pointer.

RESULT
  none

NOTE
  The wait pointer will look exactly like the standard Workbench 2.0
  wait pointer.  In combination with PointerX, ClockTick or LacePointer
  the handle will turn.

BUGS
  none known

SEE ALSO
```

1.21 rtspread()

```
NAME  rtSpread()

  rtSpread (posarray, sizearray, totalsize, min, max, num);

  void rtSpread (ULONG *, ULONG *, ULONG, ULONG, ULONG, ULONG);
     A0     A1       D0     D1     D2   D3

DESCRIPTION
  Evenly spread a number of objects over a certain length.
  Primary use is for arrangement of gadgets in a window.

  Example:

  'sizearray' holds following values: 4, 6, 4, 2 and 8,
  'totalsize' is 24 (= 4 + 6 + 4 + 2 + 8),
  'min' is 3, 'max' is 43,
  and finally, 'num' is 5.

  After calling rtSpread() 'posarray' would hold the following
  values: 3, 11, 19, 26 and 31.

  My attempt at a visual representation:

|              |
|  |            |  |
|  OOOO    OOOOOO    OOOO    OO    OOOOOOOO  |
|  |            |  |
|   1    1    2  2    3    3  4    4
0----5----0----5----0----5----0----5----0----5

INPUTS
  posarray  - pointer to array to be filled with positions.
  sizearray - pointer to array of sizes.
  totalsize - total size of all objects (sum of all values in sizearray).
  min       - first position to use.
  max       - last position, first _NOT_ to use.
  num       - number of objects (size of posarray and sizearray).

RESULT
  none

NOTE
  This function is for the advanced ReqTools user.

BUGS
  none known

SEE ALSO
```

1.22 rtunlockwindow()

```
NAME  rtUnlockWindow() [V38]

  rtUnlockWindow (window, windowlock);
```

```
  void rtUnlockWindow (struct Window *, APTR);
     A0     A1

DESCRIPTION
  Unlock a window previously locked with rtLockWindow().  The window will
  once again accept user input and will get its original mouse pointer
  back (default or custom).

INPUTS
  window     - pointer to the window to be unlocked.
  windowlock - the windowlock pointer returned by rtLockWindow(), may
   be NULL.

RESULT
  none

BUGS
  none known

SEE ALSO
```