ghostrider

COLLABORATORS			
	TITLE : ghostrider		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		November 28, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ghostrider

1.1 ghostrider.doc

```
     grClearBreakPoint()
     grEnterGR()
     grSetBreakPoint()
     grSetEntryQualifier()
```

1.2 ghostrider.library/grClearBreakPoint

```
   NAME
  grClearBreakPoint -- Remove breakpoint from address or clear table.

   SYNOPSIS
  error = grClearBreakPoint( Address )
  D0                         A0

  BYTE grClearBreakPoint( APTR )

   FUNCTION
  This function removes a breakpoint set with grSetBreakPoint() or
  clears the breakpoint table if called with the value -1.

   INPUTS
  Address - Address to be removed from the breakpoint table or -1 for
      clearing the breakpoint table.

   RESULT
  error - Result of operation. If not NULL it is identifying one of
      the situations below:

      gr_grnotfound - The library could not find GhostRider in
          the system. (Re)load GhostRider using the
          DeckRunner.
      gr_sbp_notset - Address did not have a breakpoint.

   NOTES
  If a breakpoint is executed (i.e. GhostRider is invoked) the
```

breakpoint will be removed from the list. Because of this you should be able to ignore the gr_sbp_notset error in normal situations.

SEE ALSO

grSetBreakPoint(), libraries/ghostrider.i

1.3 ghostrider.library/grEnterGR

NAME

grEnterGR -- Enter GhostRider by system call.

SYNOPSIS

```
error = grEnterGR( OrgPC, OrgStack )
D0       D0,    D1

BYTE grEnterGR( APTR, APTR );
```

FUNCTION

If your program's error handler call this function you might have a better chance of finding bugs in your software.
This function also gives you the possibility of having a look at structures/tables/registers etc., on the fly.
With the two parameters, OrgPC and OrgStack, you may pass "fake" stack and PC values to GhostRider. Now, why would you do this?
Here is an example; You are working on an application which have a handler in the input stream. The function of this handler is to notify the main program of a specific hot-key activation (quit, iconify or whatever). This can be done with standard signal notification (via Exec), and since you have the handler ready, you decide to use it for debugging purpose as well. By using the tc_ExceptCode field of your task, an extra signal and the Exec function SetExcept you get a _very_ powerfull debugging facility.
Now, the task exception is a lovely thing because it will break your task's execution and enter the exception code. Let this code retrieve the task's PC and the correct stack position and pass this in the call to GhostRider. When GR is invoked it's current address will be positioned at the location on which the task execution was halted. Also, the stackpointer will be correct. Unfortunately the contents of the other registers will not be correct. However, you will be able to find their correct contents at the negative side of the stackpointer.
If OrgPC or OrgStack is NULL, the PC/Stack addresses will be derived from the stack (this will be the actual caller and stack addresses).

EXAMPLES

This example show how to use the grEnterGR call in a task. With the setup below, it is possible to break the task execution even from another task or an interrupt (or an input event handler). If you use an interrupt/event handler you will be able to invoke GR asynchronously much like with a NMI-button. This entry method has the added feature of being more system friendly, though.

```
;---- This code initializes the exception
```

```
SetupSignalExceptions
    move.l  $4.w,a6
    sub.l a1,a1     ;Find task
    jsr _LVOFindTask(a6)
    move.l  d0,MyTask

    move.l  d0,a0     ;Set exception ptr
    move.l  #TaskExceptionCode,TC_EXCEPTCODE(a0)

    moveq #-1,d0
    jsr _LVOAllocSignal(a6) ;Allocate signal
    move.w  d0,GhostRiderSignal ;and store

    moveq #0,d0     ;Mark signal for
    move.w  GhostRiderSignal(pc),d1 ;exception
    bset  d1,d0
    move.l  d0,d1
    jsr _LVOSetExcept(a6)
    rts

;---- This code frees the signal
    move.l  $4.w,a6
    moveq #0,d0
    move.w  GhostRiderSignal(pc),d0
    jsr _LVOFreeSignal(a6)  ;Free signal
    rts


;---- This is the exception code
TaskExceptCode  movem.l d0-a6,-(a7)

    move.w  GhostRiderSignal(pc),d1 ;Does causing
    btst  d1,d0     ;signal match?
    beq.b .DontInvokeGR

    move.l  8+15*4(a7),d0   ;Get task PC
    move.l  a7,d1     ;Calculate
    add.l #8+4+2+15*4+15*4,d1 ;task SP
    move.l  _GRBase(pc),a6
    jsr _LVOgrEnterGR(a6) ;and call GR

.DontInvokeGR movem.l (a7)+,d0-a6
    rts

;---- Use the code below to invoke GhostRider
    ...
    move.l  $4.w,a6
    moveq #0,d0     ;Build signal
    move.w  GhostRiderSignal(pc),d1
    bset  d1,d0
    move.l  MyTask(pc),a1   ;Get task ptr
    jsr _LVOSignal(a6)    ;and signal
    ...   (I guess this is the PC you will get if
       this code is put in the task execution
       flow.)
```

```
 INPUTS
OrgPC - The value GhostRider should use to when you refer to the
    PC register.
OrgStack- The value GhostRider should use to when you refer to the
    stack register (see bugs note below).

 BUGS
Since this function use the system to startup GhostRider, some of
the register information will be destroyed. If you want a "clean"
entry you should use the grSetBreakPoint() function.

This function assumes that it has been called from user mode. If
it is called from supervisor mode the stackpointer and SR will not
reflect this - the OrgStack will always be copied to USP.

When GhostRider is invoked with fake PC/SP it is not possible to
change the exit address (well, it is, but you will have to do it
on the stack yourself).

 SEE ALSO
grSetBreakPoint()
```

1.4 ghostrider.library/grSetBreakPoint

```
 NAME
grSetBreakPoint -- Add breakpoint to address.

 SYNOPSIS
error = grSetBreakPoint( Address )
D0                       A0

BYTE grSetBreakPoint( APTR )

 FUNCTION
This function makes it possible to use GhostRider as a debugger
while you are developing a new program. By calling this function
in the start of the program the breakpoint(s) will always be
correctly positioned.
Also, by using this function you can keep all GhostRider interfacing
in one part of your program (as opposed to using grEnterGR()), making
it easier to remove later.

 INPUTS
Address - Address to be added to the breakpoint table.

 RESULT
error - Result of operation. If not NULL it is identifying one of
    the situations below:

    gr_grnotfound - The library could not find GhostRider in
        the system. (Re)load GhostRider using the
        DeckRunner.
    gr_sbp_fail - Address could not be accessed (actually a
        verify error)
    gr_sbp_full - Breakpoint table is full.
    gr_sbp_isset  - Address already have a breakpoint.
```

EXAMPLES

This example shows how to have a quick look at the result of a function call by invoking GhostRider immediately after the call returns:

```
  ...
  lea LetsSeeWhatWeHaveGot(pc),a0 ;This could be in the
  Call  grSetBreakPoint   ;very beginning of the code.
  ...
  Call  SomethingUtterlyBoringAndVeryConfuzing
LetsSeeWhatWeHaveGot:     ;When the CPU reach this
  ...       ;point, GhostRider will be
          ;invoked.
```

This example show how to deal with TRAP command conflicts (see the BUGS section):

```
  ...
  lea -1,a0     ;This will clear the
  Call  grClearBreakPoint ;breakpoint table, thus
  move.l  #MyTRAPHandler,$80+VBR  ;forcing GhostRider to fetch
  lea SomeBreakPoint(pc),a0 ;the pointer to MyTRAPHandler
  Call  grSetBreakPoint   ;when the first breakpoint is
  ...       ;set.
```

NOTES

Remember: GhostRider is not system dependant, so you are able to use this function to set breakpoints in parts of your code which suffer from... er, attitude problems :^)

BUGS

Since GhostRider use one of the TRAP commands for breakpoint setting you might get a conflict if your program also uses TRAP commands. You can circumvent this problem in two ways:

- By not using the same TRAP command used by GhostRider (defaults to TRAP #0, but may be changed from within GhostRider)
- By clearing all breakpoints, set your TRAP vector and then set the needed breakpoints. GhostRider will only act upon TRAPs that match a breakpoint. Others are parsed through to the original TRAP vector. The "original TRAP vector" is fetched whenever a "first" breakpoint is set. That is why you must clear the breakpoint table before setting your own vector.

SEE ALSO

grEnterGR(), grClearBreakPoint(), libraries/ghostrider.i

1.5 ghostrider.library/grSetEntryQualifier

NAME

grSetEntryQualifier -- Set qualifier for hot start.

SYNOPSIS

```
error = grSetEntryQualifier( Type, Code, Qualifier )
D0                           D0,   D1,   D2
```

```
BYTE grSetEntryQualifier( ULONG, UBYTE, UWORD )

 FUNCTION
This function will initialize a "hot starter", patched into the
input event stream, which will activate GhostRider if the specified
qualifiers occur.

 INPUTS
Type -  Type of entry qualifier:
   - NULL    : This will remove the handler.
   - GRETB_mbutton : The middle mouse button must be pressed
         for activation.
   - GRETB_rbutton : The right mouse button must be pressed
         for activation.
   - GRETB_lbutton : The left mouse button must be pressed
         for activation.
  ; The three mouse button flags function as a mask.
  ; [GRETF_rbutton!GRETF_lbutton] will invoke GhostRider if
  ; left AND right, but NOT the middle mouse button is pressed.
  ; These qualifiers may not be mixed with the RAWKEY qualifier.

   - GRETB_rawkey  : ie_Code and ie_Qualifier of a RAWKEY event
         must match the Code and Qualifier codes
         specified for activation.
         The CapsLock flag is ignored.
Code -  RAWKEY code for RAWKEY hot start.
Qualifier -  Qualifier settings for RAWKEY hot start.

 RESULT
error - Result of operation. Not used.

 EXAMPLES
moveq #%0001,d0   ;Invoke with middle mouse button.
Call  grSetEntryQualifier

moveq #%1000,d0   ;Type = RAWKEY event
moveq #$5F,d1     ;Invoke with Ctrl+HELP
moveq #$08,d2
Call  grSetEntryQualifier

 NOTES
The RAWKEY codes may be found in various litterature. The qualifier
bits are described in the system includes (IEQUALIFIERB_XXXXXXXXX).

 BUGS
Since a GhostRider entry by hot key would cause the system to miss
the key releases, I have included a highly illegal fix to this
problem in the GhostRider. This fix will find the keyboard.device
in the system’s device list and clear the keyboard matrix at exit.
Since the base of keyboard.device is not public, I have had to do
some research myself. The problem is that the fix may not work on
all KICKSTART versions, but I have checked it to do on versions
37.12 - 40.xx.
Oh, almost forgot; you can disable this fix in the preferences
(default is clearing the keyboard matrix)
```

```
  SEE ALSO
 libraries/ghostrider.i
```