Scout

System Monitor
Scout 37.42 (Release 1.3)
Edition 1.3
September 1994

by Andreas Gelhausen

1 Introduction

What is Scout?

`Scout` is a tool that allows you to monitor your computer system. It displays many different things — like tasks, ports, assigns, expansion boards, resident commands, interrupts, etc. — and you can perform some certain actions on them.

For example you can freeze tasks, close windows and screens, release semaphores or remove locks, ports and interrupts.

An implemented ARexx interface makes you these actions available, too.

But this program is rather a tool for programmers and everybody who wants to watch a little bit deeper in his system than for 'normal' users.

2 Legalities

Copyright

No Warranty

No warranties are made for this program. All use is at your own risk. No liability or responsibility is assumed for any damages occured during the usage of `Scout`. You have been warned.

Giftware

`Scout` 37.42 is *giftware*. If you like and use this program, you are welcome to appreciate my programming efforts by sending me a little present — thanks a lot in advance! =:^)

3 Before Starting

What your system should have

`Scout` requires Amiga operating system version 2.04 and MUI version 2.1. See below. Of course you can use higher versions of these things, too.

MUI - MagicUserInterface

MUI is a system to generate and maintain graphical user interfaces. With the aid of a preferences program, the user of an application has the ability to customize the outfit according to his personal taste.

MUI is distributed as shareware. To obtain a complete package containing lots of examples and more information about registration please look for a file called '`muiXXusr.lha`' (XX means the latest version number) on your local bulletin boards or on public domain disks.

If you want to register directly, feel free to send DM 30.- or US$ 20.- to

Stefan Stuntz
Eduard-Spranger-Straße 7
80935 München
GERMANY

Installing Scout

You only have to copy the program `Scout` and the data file `scout.data` to your favourite directory and then you can start it. `Scout.data` includes data of expansion boards.

4 How to use Scout

If you start the program you will see the main window which includes many gadgets. Each of these gadgets represents a certain kind of system structure.

You can choose between:

Assigns, Devices, Expansions, Fonts, InputHandlers, Interrupts, Libraries, Locks, Memory, Mounted Devices, Ports, Resident Commands, Residents, Resources, Semaphores, Tasks, Vectors and Windows.

Click one of these gadgets and another window will be opened with a list of the structure type that is indicated on the pressed gadget.

Example: Press the task gadget and you will get a window with the list of tasks and processes.

You can also select these functions by pressing the underlined key you see on each gadget or by using the right mousebutton menu.

If you wish to handle/remove a given structure, you should know what you do.

> **Warning:** Wrong handling of the showed structures can crash your system. At the worst you will lose your data.

Please note: You should not be surprised, if you don't find a certain detail information in this manual, because it's to much work to explain each element of all the structures you could see in this program.

Many books are written about these things and if you want to have more information about them, you should have a look in the specialized literature.

4.1 Assigns

This type of structure assigns a logical name to a directory.

If you assign the directory '`dh0:data/documents`' the logical name '`texts:`', you will also be able to choose a file *filename* in that directory with the path 'texts:*filename*'.

Column items

'`Address`' Address of the assign structure.

'`Name`' Logical name of a directory

'`Path`' Here you will find the path of the directory.

Actions

'`Update`' Selecting this gadget updates the list of assigns.

'`Print`' This function allows you to send the list of '`Assigns`' to printer or a selected file.

'`Remove`' The selected assign will be removed with this function.

'`Exit`' The '`Assigns`' window will be closed.

4.2 Devices

A device is — like a library (see Section 4.7 [Libraries], page 12) — a collection of functions/procedures, which have to do certain jobs.

E.g. the 'trackdisk.device' includes functions for the floppy disk handling.

Column items

'`Address`' Address of the device structure

'`ln_Name`' Name of a device

'`ln_Pri`' Priority of a device

'`OpenC`' This element shows how often the device was opened.

'`RPC`' '`RPC`' means '`RAM Pointer Count`' and shows how many jump addresses of the device point into RAM. In this way many programs — like the `setpatch` command from Commodore — patch the system.

Many viruses patch the system in this way too, but don't panic now. If you check your system in regular intervals with a current virus killer, it should be out of danger.

If the whole program code of the device is located in RAM, you will find a dash (minus sign) here, because in this case it's unimportant how many jump addresses point into RAM.

'`ln_Type`' Type of this structure (usually '`device`')

Actions

'`Update`' If you select this gadget, the list of devices will be updated.

'`Print`' This function allows you to send the list of '`Devices`' to printer or a selected file.

'`Remove`' The selected device will be removed with this function provided that no program uses this device anymore and the '`OpenC`' is zero.

'`Priority`'
Herewith the priority of the device can be changed. A little window will be opened, that asks you for a new priority. Through the new priority it can happen that the device gets a new place in the device list.

'`More`' Another window will be opened and you will see more informations about the selected device.

You will have the same effect, if you doubleclick an element of the device list.

'`Exit`' The '`Devices`' window will be closed.

4.3 Expansions

In this list you will find all your expansion boards (graphic boards, memory expansions and so on).

Column items

'`BoardAddr`'
Usually you will find the ROM of the card here. If this address points into RAM, the card is a memory expansion.

'`BoardSize`'
If the entry belongs to a memory expansion, the size of the memory is displayed here. Otherwise it's the ROM size of the card.

'`Manufacturer`'
ManufacturerID, assigned by Commodore

'`Product`' Productnumber, assigned by the manufacturer of the board

'`Serial#`' Serialnumber of the card (usually unused)

Actions

'`Print`' This function allows you to send the list of '`Expansions`' to printer or a selected file.

'`More`' Now a window will be opened, that includes more informations about the selected expansion board.

Doubleclick an element of the '`Expansions`' list and you will have the same effect.

'Exit' The 'Expansions' window will be closed.

Unknown expansion boards

If you select an expansion board by selecting its list item, you will get the name of the manufacturer and the card in the textfield you find below the list, provided that I have known these data at compiling.

If no information is available in this textfield or the given information is wrong, you should send me the following data, please.

1. ManufacturerID (Manufacturer)
2. ProductID (Product)
3. Name of the company
4. Name of your expansion card

If you send me these data, the next version of `Scout.data` will know your expansion boards. Please be as precise you can.

4.4 Fonts

This function will show you all fonts existing in your system.

Column items

'YSize' Vertical size of the font

'Count' Here you can see how many programs use the font.

'Type' 'ROMFONT' means the font is located in ROM and 'DISKFONT' means the font was loaded from disk/harddisk.

'Name' Name of the font

Actions

'Update' The list of fonts will be updated.

'Print' This function allows you to send the list of 'Fonts' to printer or a selected file.

'Close' The font will be closed by using this function.

'Remove' It is possible to remove a font from system, provided that no program uses it and it's no 'ROMFONT'.

'Exit' The 'Fonts' window disappears.

4.5 InputHandlers

Input handlers take care of all user input arriving in system (pressed keys, mouseclicks, inserted disks, etc.). They stand one behind the other like on a production line and analyze the user input. The input handler with the highest priority gets the 'events' first and if it doesn't know how to react on these 'events', the second input handler gets them, and so on.

Usually the system input handler has a priority of 50. Every input handler, that wants to get the user input before the system, must have a higher priority.

Column items

'`ln_Name`' Name of the input handler

'`ln_Pri`' Its priority

'`is_Data`' This address points to some data needed by the input handler.

'`is_Code`' The program code starts here. If the code is located in RAM, the address is of different color. Otherwise you can find the code in ROM. Some viruses install an input handler in system. In this case the '`is_Code`' address points into RAM, but many other programs uses input handlers, too. Don't panic!

Actions

'`Update`' The list of input handlers will be updated when you select this gadget.

'`Print`' This function allows you to send the list of '`InputHandlers`' to printer or a selected file.

'`Remove`' Removes an input handler from system.

'`Priority`'
Changes the priority of an input handler.

'`Exit`' The window will be closed.

4.6 Interrupts

Interrupts are important events the computer system has to react on. It exists a list of interrupt routines for each interrupt type. If a certain interrupt occurs, all these interrupt routines will be called. During their execution the running program will be interrupted.

Column items

'`ln_Name`' Name of the interrupt

'`ln_Pri`' Its priority

'`is_Data`' At this address you find the data of the interrupt.

'`is_Code`' Address of the interrupt code. If this address points into RAM, it's of a different color.

'`NUM`' This number represents the type of event the interrupt routine is called on. The '`IntName`' you find in the interrupt detail window gives you a little bit more information about it.

Example: Number 5 means that the interrupt is called at every vertical blank interval.

Actions

'`Update`' The list of interrupts will be updated.

'`Print`' This function allows you to send the list of '`Interrupts`' to printer or a selected file.

'`Remove`' If the interrupt is a server you can remove it from system. An interrupt handler can't be removed by `Scout`.

If you call `avail flush` and the audio.device isn't used, the interrupt handlers of the audio.device will be removed.

'`More`' Now a window will be opened that includes more details of the interrupt.

'`Exit`' Selecting this gadget will close the '`Interrupts`' window.

4.7 Libraries

A library is a collection of functions/procedures, which have to do certain jobs.

E.g. the 'graphics.library' includes routines for graphical display.

Column items

'`Address`' Adress of the library structure

'`ln_Name`' Name of a library

'`ln_Pri`' Priority of a library

'OpenC' Here you see, how often the library was opened.

'RPC' 'RPC' means 'RAM Pointer Count' and shows how many jump addresses of the library point into RAM. In this way many programs — like the setpatch command from Commodore — patch the system.

Many viruses patch the system in this way too, but don't panic now. If you check your system in regular intervals with a current virus killer, it should be out of danger.

If the whole program code of the library is located in RAM, you will find a dash (minus sign) here, because in this case it's unimportant how many jump addresses point into RAM.

'ln_Type' Type of this structure (usually 'library')

Actions

'Update' The list of libraries will be updated.

'Print' This function allows you to send the list of 'Libraries' to printer or a selected file.

'Remove' The selected library will be removed with this function provided that no program uses this library anymore and the 'OpenC' is zero.

Some libraries can't be removed from system without a reset. So you shouldn't wonder about it, if this happens.

'Close' A library must be closed by all programs, if you want to remove it from system. In this case the 'OpenC' is zero.

If you select this function, you will be asked, how often you want to close it. You can choose between 'Once' and 'All'.

Select 'All' and the library will so often be closed till the 'OpenC' is zero.

'Priority'

Herewith the priority of the library can be changed. A little window will be opened, that asks you for a new priority. Through the new priority it can happen that the library gets a new place in the list of libraries.

'More' A window will be opened that includes more details of the library.

'Exit' Selecting this gadget will close the 'library' window.

4.8 Locks

A lock structure shows you, that a program reads from or perhaps write into a file or a directory. With this type of structure the system prevents, that a file will be deleted while another program gets some data from it.

Column items

'`Access`' Here you can see the type of access. This could be '`READ`', '`WRITE`' or '`OWN`'. '`OWN`' stands for a lock `Scout` created to get the elements of this list.

'`Path`' Path of the file or directory

Actions

'`Update`' The list of '`Locks`' will be updated.

'`Print`' This function allows you to send the list of '`Locks`' to printer or a selected file.

'`Remove`' A lock will be removed through dos.library's 'UnLock()' function.

'`Pattern`' If you give `Scout` a pattern, only the locks with a matching path will be shown.

'`Exit`' The '`Locks`' window will be closed.

4.9 Memory

In this list you will find the segments of your memory. At least you will find an entry for your chip memory.

Column items

'`ln_Name`' Name of the memory segment (e.g. '`chip memory`')

'`ln_Pri`' Priority of memory

'`mh_Lower`'
First address of memory

'`mh_Upper`'
Last address of memory

Actions

'`Print`' This function allows you to send the list of the memory segments to printer or a selected file.

'`Priority`'
This function allows you to change the priority of a memory segment. The memory segment with the highest priority will be preferred from system, provided that no certain type of memory is demanded.

'`More`' Another window will be opened. This window includes more information about the memory segment.

'`Exit`' The window will be closed.

4.10 Mounted Devices

In this list you will find all your devices like disk drives, printer devices, etc.

Column items

'`Name`' Name of the device

'`Unit`' Unit number

'`Heads`' Number of heads

'`Cyl`' Number of cylinders

'`State`' The state shows you for example, if a disk is in drive.

'`DiskType`'
Type of a disk (e.g. OFS (OldFileSystem), FFS (FastFileSystem), ...)

'`Handler or Device`'
The handler or the device you find here has to manage the stream of data from and to the device.

Actions

'`Update`' The list will be updated.

'`Print`' This function allows you to send the list of '`Mounted Devs`' to printer or a selected file.

'`Exit`' The window will be closed.

4.11 Ports

Programs are able to communicate together through ports.

Column items

'`Address`' Here you will find the port structure.

'`ln_Name`' Name of port

'`ln_Pri`' Priority of port

'`mp_SigTask`'
The task is communicating through the port.

Actions

'`Update`' The ports list will be updated.

'`Print`' This function allows you to send the list of '`Ports`' to printer or a selected file.

'`Remove`' The port will be removed.

'`Priority`'
Herewith the port priority can be changed.

'`Exit`' The '`Ports`' window will be closed.

4.12 Resident Commands

This list includes all resident commands. That means all commands you find in ROM and the commands you made 'resident' through the `resident` command.

Positions and sizes of their hunks you will find here, too.

Column items

'`Name`' Name of the command

'`UseCount`'
Here you can see, how often a command was being executed at the time the list was build.

'`Lower`' First address of hunk in memory

'`Upper`' Last address of hunk in memory

'`Size`' Size of hunk (upper - lower - 8 bytes overhead)

Actions

'`Update`' The list of '`Resident Commands`' will be updated.

'`Print`' This function allows you to send the list of '`Resident Commands`' to printer or a selected file.

'`Remove`' The selected command will be removed with this function provided that no program uses this command anymore and the '`UseCount`' is zero.

'`Exit`' The window disappears.

4.13 Residents

Resident modules are reset-protected segments (code and data). In the list of '`Residents`' you usually find libraries, devices and resources. A programmer has the possibility to make his own programs reset-protected. He has to initialize a resident structure for it and then he can link the program through the kick-vectors (see Section 4.17 [Vectors], page 21) to the list of the resident modules. The residents you linked to system are usually located in RAM and are of a different color.

If you find a resident module that points into RAM and you don't know which program has created it, you should start your favourite virus detector and let it check your memory. Many viruses prefer this way to travel around.

Column items

'`Address`' At this address the resident module is located.

'`ln_Name`' Name of the resident module

'`rt_Pri`' Priority

'`rt_IdString`'
Identity string of the resident module.

Actions

'`Update`' The list of '`Residents`' will be updated.

'`Print`' This function allows you to send the list of '`Residents`' to printer or a selected file.

'`More`' Selecting this gadget opens a new window with more information about the selected resident module.

'`Exit`' The '`Residents`' window will be closed.

4.14 Resources

Usually a resource is — like a library (see Section 4.7 [Libraries], page 12) — a collection of functions/procedures, which have to do certain jobs.

E.g. the 'filesystem.resource' includes functions for the filesystem handling.

Column items

'`Address`' Address of the resource structure

'`ln_Name`' Name of a resource

'`ln_Pri`' Priority of a resource

'`OpenC`' This element shows how often the resource was opened.

'`RPC`' '`RPC`' means '`RAM Pointer Count`' and shows how many jump addresses of the resource point into RAM. In this way many programs — like the `setpatch` command from Commodore — patch the system.

Many viruses patch the system in this way too, but don't panic now. If you check your system in regular intervals with a current virus killer, it should be out of danger.

If the whole program code of the resource is located in RAM, you will find a dash (minus sign) here, because in this case it's unimportant how many jump addresses point into RAM.

'`ln_Type`' Type of this structure (usually '`resource`')

Actions

'`Update`' The list of '`Resources`' will be updated.

'`Print`' This function allows you to send the list of '`Resources`' to printer or a selected file.

'`Remove`' The selected resource will be removed with this function, provided that no program uses it anymore and the '`OpenC`' is zero.

'`Priority`'

Herewith the priority of the resource can be changed. A small window will be opened, that asks you for a new priority. Through the new priority it can happen that the resource gets a new position in the list of resources.

'`More`' Select this gadget and you get a new window with more information about the selected resource.

'`Exit`' The '`Resources`' window will be closed.

Please note: If you should find three dashes (minus signs) at '`OpenC`' and/or '`RPC`', the resource has no typical library structure. This happens for example at the 'FileSystem.resource'.

4.15 Semaphores

The use of semaphores is a way of single-threading critical sections. For example only one program is allowed to use the printer at one time, otherwise the texts would be mixed.

Column items

'`ln_Name`' Name of a semaphore

'`NestCnt`' This element shows how often the semaphore is used.

'`ln_Type`' Type of the structure (usually '`signalsem`')

Actions

'`Update`' The list of '`Semaphores`' will be updated.

'`Print`' This function allows you to send the list of '`Semaphores`' to printer or a selected file.

'`Obtain`' This function is used to gain access to a semaphore. The '`NestCnt`' will be increased at one by this call.

'`Release`' Herewith you can make a signal semaphore available to others.

'`Exit`' The '`Semaphores`' window will be closed.

4.16 Tasks

In this window you find a list of all tasks and processes being in system. Each program you start will be executed as a task or process.

Column items

'`ln_Name`' Name of the task/process

'`ln_Type`' Type of the structure ('`task`' or '`process`')

'`ln_Pri`' Priority of the task/process

'NUM' If a non detaching program was started from shell, you will find here the number of the process. Programs you started from Workbench have a dash here.

'State' Here you see the state of the task or process. You will find Scout's own process on the top of the list with a 'run' at this place, because this process is always running when it gets the task list.

'ready' means the task wants to work, but it's interrupted by the execution of another task.

A task that is waiting for a certain signal is in the state 'wait'. In this case it doesn't need processing time.

'SigWait' Signalmask the task is waiting for.

Actions

'Update' The list will be updated.

'Print' This function allows you to send the list of 'Tasks' to printer or a selected file.

'Remove' A task will be removed from the list. You should prefer the freeze function, if you perhaps need this task again.

See also 'Break'!

'Freeze' With this function you freeze the selected task. It can still be found in the list of tasks, but it gets no processing time from system.

> **Warning:** If you try to freeze tasks essential to the system like 'input.device', you should have saved all important data, cause a RESET is the only way out!

'Activate'
A frozen task can be activated here.

'Secs' This string gadget allows you to set the intervall time for the CPU usage display.

'CPU/No CPU'
This cycle gadget let you choose between CPU usage display and not CPU usage display. At the first time of selecting 'CPU' some system functions will be patched. Scout uses these patches to calculate the percents of CPU time each task needs.

'Signal' If you select a signal mask, it will be send to the task.

'Break' A signal mask that includes the signals CTRL-C and CTRL-D will be send to the task you selected. Many tasks and processes end, if they receive these signals.

'`Priority`'
: The priority of a task can be changed with this function.

'`More`'
: Selecting this gadget will open another window that displays more informations about the task or the process.

'`Exit`'
: The window will be closed.

4.17 Vectors

Actions

'`Update`'
: The displayed vectors will be updated.

'`Print`'
: This function allows you to send the list of '`Vectors`' to printer or a selected file.

'`Exit`'
: The window will be closed.

Reset Vectors

A program can make itself reset-protected by using the reset vectors. If the vectors are unused, they have a value of zero. The programs which use the Kick-Vectors (KickTagPtr, KickMemPtr and KickCheckSum) can also be found in the list of resident structures. See also Section 4.13 [Residents], page 17.

Auto Vector Interrupts

In a computer system with a MC68000 processor you will find the seven '`Auto Vector Interrupts`' from address $64 to address $7c. Higher processors (MC68010, etc.) have the VBR (Vector Base Register) that allows you to move the interrupt table to FAST-MEM. The system will be a little bit faster then. `Scout` uses the VBR if it exists.

Interrupt Vectors

Here you see 16 interrupt vectors (IntVecs). These vectors are located in the 'ExecBase' (base structure of the exec.library).

4.18 Windows

All screens with the windows opened on them are listed here. Screens are of a different color as windows.

Column items

'`Pos(x,y)`'
: x and y position of the screen/window

'`Size(x,y)`'
: x and y size of the screen/window

'`Title`'
: Title of the screen/window

Actions

'`Update`'
: The list will be updated.

'`Print`'
: This function allows you to send the list of '`Windows`' to printer or a selected file.

'`Close`'
: With this function it is possible to close screens and/or windows. If you close a screen, all windows on it will be closed too.

'`More`'
: If you select this gadget another window will be opened that displays more informations about the window or the screen.

'`Exit`'
: The window will be closed.

5 Options

There are some options for `Scout` which you can use, when you start the program. The following options are available from shell and as tool types from Workbench.

Example:

```
Scout TOOLPRI=1 ICONIFIED
```

starts `Scout` iconified with a priority of one.

'`ICONIFIED`'

Usage: `ICONIFIED`

If this option is activ, `Scout` starts iconified.

'`PORTNAME`'

Usage: `PORTNAME`=*portname*

The name of Scout's ARexx port can be changed into *portname*. Without this option the ARexx port is called 'SCOUT.X'. The X stands for a decimal number that will be incremented, if a so called port already exists.

'`TOOLPRI`'

Usage: `TOOLPRI`=*value*

This option allows you to change the priority of Scout's process into *value*.

'`STARTUP`'

Usage: `STARTUP`=*scriptname*

You can choose an ARexx script *scriptname*, that will be executed at the start of `Scout`. In this way you can open more than only the main window. If for example the ARexx script includes the command '`OpenWindow Tasks`', the task list window will always be opened when the program starts.

(See also Chapter 6 [ARexx Port], page 25.)

6 Scout's ARexx interface

It's a feature of MUI to give each application its own ARexx port. Therefore `Scout` also has an ARexx port that usually has the name '`SCOUT.X`'. The X stands for a decimal number that will be incremented, if a so called port already exists.

You will find the name of Scout's ARexx port in the window you get, if you select the '`Project/About`' menu.

Using tasknames:

If a task or a process was started from shell and hasn't detached itself, you will find the name of the command being executed, where usually the taskname is displayed. The real name of those tasks usually is something like '`Background CLI`', but such a taskname isn't useful.

Example: If you start a non detaching task like '`DH0:Debug/Sushi`' from shell, you will see '`DH0:Debug/Sushi`' as taskname.

Some ARexx commands need a taskname as parameter. You have to select those from CLI started self detaching tasks by using their command names like `Scout` displays them in the lists of tasks.

Scout supports following ARexx commands:

'`FindTask`'

Usage: `FindTask` *taskname*

This command returns the address of the task *taskname*, if it is in system.

'`FreezeTask`'

Usage: `FreezeTask` *taskname*

The task *taskname* will be frozen.

'`ActivateTask`'

Usage: `ActivateTask` *taskname*

The frozen task *taskname* will be activated.

'`RemoveTask`'

Usage: `RemoveTask` *taskname*

This command removes the task *taskname*.

'`SendBreak`'

Usage: `SendBreak` *taskname*

`Scout` sends the task *taskname* a certain signal mask that includes the signals CTRL-C and CTRL-D. Many programs support these signals and finish themselves, if they receive one of them.

'SendSignal'

Usage: `SendSignal` *taskname hexsignal*

This command allows you to send a signal *hexsignal* to the task *taskname*. The signal must specified as a hexadecimal number.

Example:

```
SendSignal 'scout' 0x001000
```

sends task '`scout`' a CTRL-C and after that `Scout` ends.

'SetTaskPri'

Usage: `SetTaskPri` *taskname priority*

The task *taskname* gets a new priority (*priority*).

'RemovePort'

Usage: `RemovePort` *portname*

The port *portname* will be removed from `Scout`.

'GetLockNumber'

Usage: `GetLockNumber` *lockpattern*

This command returns the number of locks which have paths matching to the pattern *lockpattern*.

Example: Use the command

```
GetLockNumber 'WORK:Utilities/#?'
```

and you will know, how many locks are currently used for files in the directory '`WORK:Utilities/`'.

'RemoveLocks'

Usage: `RemoveLocks` *lockpattern*

Use this command and all locks which have paths matching to the pattern *lockpattern* will be removed. (See also `GetLockNumber`.)

'OpenWindow'

Usage: `OpenWindow` *windowid*

All windows you get if you select a gadget of the main window, can be opened with this command. The *windowid* is the same text you find on the main window gadgets.

Example:

```
OpenWindow 'Mounted Devs'
```

will open the window with the list of mounted devices.

'FindName'

Usage: `FindName` *nodetype nodename*

This command allows you to find a certain node. You only have to know its name (*nodename*) and its type (*nodetype*).

Nodetype can have following values: '`LIBRARY`', '`DEVICE`', '`RESOURCE`', '`MEMORY`', '`SEMAPHORE`', '`PORT`' or '`INPUTHANDLER`'.

Example: If you want to get the address of the 'disk.resource' you must use:

```
FindName RESOURCE 'disk.resource'
```

'GetPriority'

Usage: GetPriority *nodeaddress*

This command allows you to check the priority of a certain node structure. This includes all following structure types: tasks, libraries, devices, resources, ports, residents, input handlers, interrupts, semaphores and the elements of the memory list.

You only have to know the address (*nodeaddress*) of that structure.

Example: The following ARexx commands store the priority of your chip memory in the variable 'pri':

```
FindName MEMORY 'chip memory'
addr = result
GetPriority addr
pri = result
```

'SetPriority'

Usage: SetPriority *nodetype nodename*

If you want to change the priority of the node *nodename*, you can use this command. Again *nodetype* can have following values: 'LIBRARY', 'DEVICE', 'RESOURCE', 'MEMORY', 'SEMAPHORE', 'PORT' or 'INPUTHANDLER'.

'FindResident'

Usage: FindResident *residentname*

This command returns the address of the resident structure *residentname*.

'FindInterrupt'

Usage: FindInterrupt *interruptname*

The address of the interrupt *interruptname* will be returned.

'FlushDevs'

Usage: FlushDevs

All not used devices will be removed. The used memory will be freed.

'FlushFonts'

Usage: FlushFonts

If a diskfont is in memory, but no program uses it, it will be removed.

'FlushLibs'

Usage: FlushLibs

All not used libraries will be removed. The used memory will be freed.

'`FlushAll`'

Usage: `FlushAll`

This function includes `FlushDevs`, `FlushFonts` and `FlushLibs`. All not used devices, libraries and fonts will be removed and the used memory will be freed.

'`ClearResetVectors`'

Usage: `ClearResetVectors`

The six reset vectors will be cleared, if you select this function (see Section 4.17 [Vectors], page 21).

Appendix A

How to get updates

The newest version of `Scout` should always be available in the "DEEP THOUGHT BBS" (see below), on AmiNet or Public Domain collections, which are up-to-date.

Support BBS

```
DEEP THOUGHT Bulletin Board System, Oldenburg, Germany

Node 1
+49-(0)441-383365   1200-21600 bps  v.32terbo, v.42bis

Node 2
+49-(0)441-383839   1200-19200 bps  v.32bis, v.42bis, ZyXEL

                Node 1                Node 2
FidoNet         2:2426/2020.0         2:2426/2021.0
AmigaNet        39:170/204.0          39:170/205.0

InterNet        cosinus@deepthought.north.de


Both Nodes are 24 hours online every day.
A FidoNet Mailer is running on both Nodes which accepts
FidoNet Filerequests.

Use the magic   SCOUT for the newest version of SCOUT
or              FILES for a complete filelist
```

Credits

Now I have to thank:

Klaus 'gizmo' Weber, he tested this piece of software and was always available to me and my many questions (not a few) during the programming of `Scout`.

Christian 'cosinus' Stelter, he gave me the permission to use his many manuals.

Stefan Stuntz for his great '`MagicUserInterface`'

Kai ‘wusel’ Siering for testing, reporting bugs, etc.

and last but not least

all the others I’ve forgotten for reporting bugs, sending expansion boards data and so on.

How to reach the author

If you have questions, suggestions, bug reports or anything else, you can send electronic mails to:


```
atte@crash.north.de (Andreas Gelhausen)
                or
      2:2426/2020.24  (on FidoNet)
```


If it is not possible for you to use this way, you can send letters to:


```
Andreas Gelhausen
Graf Spee Str. 23b
26123 Oldenburg
   - Germany -
```

Index

Table of Contents