SnoopDos

COLLABORATORS

	TITLE : SnoopDos	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 24, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

SnoopDos

1.1 main

SnoopDos 3.0 -- System and application monitor

SnoopDos is a utility that allows you to monitor a variety of system operations carried out by programs on your Amiga. This includes what files a program is trying to open, what fonts, libraries, devices and environment variables it is looking for, and so on.

Click on one of the following topics for more detailed information:

Click on the CONTENTS button at the top to return here at any time.

1.2 quickstart

A QUICK GUIDE TO USING SNOOPDOS

To start SnoopDos, just double-click on the SnoopDos icon from Workbench or type "Run SnoopDos" from the CLI.

The default options have been chosen to suit most configurations. You can click on the Functions button to choose what functions are being monitored, or the Setup button to change most other settings. Also check out the menus associated with the main window.

Here are the main things you need to know:

- Press HELP from inside SnoopDos to access this help file.
- The headings displayed in the main window can be dragged left and right using the mouse. Holding down shift moves all items to the right of the current item as well.
- Double-click on the header line in the main window to quickly call up the format editor. Use this editor to change the display layout by picking up a field with the left mouse button, dragging it to a new position, and releasing it.
- Changes made in the Function, Setup and Format Editor windows always take effect immediately. Click Use to make the change permament, or Undo or Cancel to restore the previous settings. You can have all three configuration windows open simultaneously.
- When you select Save Settings, SnoopDos saves the current configuration to ENVARC:SnoopDos.prefs (the SETTINGS tooltype and CLI option allows you to choose a different name). Type this file to see a partial list of supported ARexx commands, tooltypes, and CLI options. If you move this file to S:, SnoopDos will automatically use that copy instead (this saves memory in ENV:).
- If you want to monitor file operations performed by programs compiled using Gnu C (i.e those using ixemul.library) then ensure that the Monitor Packets option is turned on in the function window. This may slow down your system a little, however.
- The "Buffer->Copy Window to Clip" menu option lets you quickly copy SnoopDos output into your favourite editor using the clipboard.
- You can exclude any tasks you don’t want to monitor by entering an AmigaDOS pattern like "~(task1|task2|task3)" into the Match Name gadget in the function window.
- The Pause button in the main window lets you see what an application is trying to do before it actually does it. Any task that tries to execute a monitored function will be listed with a result of WAIT. Click on the down-arrow scroll gadget or press Space to allow the task to execute the function call.
- For DOS device writers, the Packet Debugger option in the function window lets you view all packets sent to a device. You’ll probably want to also turn on Monitor ROM Calls to view packets sent internally

by dos.library. You should also make the Action and Res. fields wider to ensure you can see all pertinent information.

- If you want SnoopDos to run automatically when you boot your system, simply drag the icon to your WBStartup drawer. Ensure the icon contains the tooltype HIDE=YES if you want SnoopDos to start up hidden in the background.

- If you want to access online help from within SnoopDos, make sure the file SnoopDos.guide is in the same directory as SnoopDos itself, or else copy it to the directory HELP:English.

See also: Contents Introduction What's New Main window

1.3 introduction

INTRODUCTION TO SNOOPDOS

How many times have you tried to install some big application program, only to find out when you run it that it doesn't seem to work properly? Often, it's looking for some configuration file, library, environment variable, or font which you've forgotten to install.

Version 1 of SnoopDos provided a simple no-frills approach to spying on a program's activities. Since its release, other programs have appeared which perform a similar task, including DosTrace by Peter Stuer, Snoopy by Gerson Kurz, and Woodward by Tomas Rokiki.

However, good as these programs are, I felt that they made it unnecessarily difficult for the user to view the information of interest. SnoopDos 3.0 aims to bring the original SnoopDos into the 90's by adding a full style guide compliant GUI and a host of powerful new features, whilst retaining the simplicity of its ancestor.

To get going with SnoopDos, simply double-click on the SnoopDos icon or type "Run SnoopDos" in a Shell window.

The first thing you'll see is the main SnoopDos window, which features a variety of buttons. At this point, SnoopDos is active, and monitoring a range of system operations. If you now run another application, you should see some lines of output displayed in the SnoopDos window. When there are too many lines to fit in the window, some will scroll off the top; you can use the scroll bar to bring them back into view.

In most cases, this is all you need to do to identify a problem. The SnoopDos window will list any files that can't be opened, fonts that can't be found, and so on, as well as all those operations that were successful.

However, you can customise SnoopDos to better suit your Amiga environment. By clicking on the Setup and Function gadgets in the main window, you can open two additional windows. The Settings window lets you change various preferences settings, such as the fonts used for the SnoopDos window, while the Function window lets you choose which events you want SnoopDos to display in the main window. If you're unsure of the meaning of any of the buttons, check out the Function window help for more details.

An important feature of SnoopDos is that you can choose exactly how much or how little information about events you want to see. Select "Show Format" from the Windows menu to open up the format editor. This contains the current event format which controls how the information in the main window is displayed. To change this, simply pick up fields using the left mouse button and drag them to a new position.

For example, if you want to see what time each event occurred at, simply drag the Time field from the "Available" box into the "Current Format" box. As soon as you release it, the main window updates to show the time associated with each event in the window. SnoopDos always records all possible information about an event, even if it isn't all being displayed at the moment.

You can also modify the format directly in the main window, by clicking on a column title with the mouse and dragging it to a new position. Usually, dragging a title repositions it independently of those on either side; if you hold down shift as you drag, then all the columns to the right will move as well.

If you make any changes to the default SnoopDos configuration, you may wish to save the changes so that they will take effect the next time you run it. Simply click on the Save Settings button in the main window, and they will be saved to a text file called ENVARC:SnoopDos.prefs. You can view this file with any text reader.

Now go and play with SnoopDos a bit. At any time within SnoopDos, you can press the HELP key to call up context-sensitive help from this document. After this, you might like to return and read the help pages dealing with the main window, function window, and settings window.

See also: Contents Quick Start What's New Main Window

1.4 whatsnew

WHAT'S NEW IN SNOOPDOS 3.0

If you are one of the many users of SnoopDos 1.7, you will be pleased to see that SnoopDos 3.0 has a host of new features designed to make it more useful.

Alas, nothing is for free in this world. SnoopDos has grown from a lean and mean CLI-only utility into a rather large GUI-based application. Since most serious Amiga users these days have an accelerated system with plenty of memory and hard drive space, this seems like a reasonable trade off. If your Amiga has more modest capabilities, you may find one of the other SnoopDos-like utilities more suited to your system.

Now, without further ado, here's a list of the major new features in this version of SnoopDos:

- A fully font-sensitive GUI lets you control all options using the mouse or keyboard. The main window gadgets can be selectively disabled to allow as many lines of text as possible to be displayed.

- A memory buffer stores details of all function calls that scroll off the top of the window. These can then be reviewed using the scroll bar.

- Many new functions can be monitored, including Workbench tooltypes, ARexx messages sent by any ARexx script, and a variety of new 2.04 DOS functions.

- Programs compiled with GNU C, which bypass dos.library and talk directly to DOS devices using packet i/o, can now be monitored.

- Additional information is available for each function monitored, including the time the function was called, what program module made the call (if SegTracker is loaded), the process ID, and much more.

- You can choose exactly which programs to monitor or ignore, using a standard AmigaDOS pattern string.

- For DOS device programmers, a packet debugger allows you to monitor every packet sent to any currently mounted filesystem device.

- A convenient "drag & drop" format editor allows you to easily select exactly how much or little information you wish to see about each event. For even greater control, you can directly adjust the width of each column of output in the main window using the mouse.

- All of the sub windows used to control various settings are modeless. You can leave them open all the time if you like, and any changes you make take effect immediately.

- SnoopDos can now run as a commodity, continuing to monitor activity while hidden in the background.

- A new Pause option allows you to freeze any tasks being monitored by SnoopDos, or single step through the calls they make, one at a time.

- A comprehensive command language with over 100 commands lets you control SnoopDos from the CLI, Workbench, ARexx, or script files.

- SnoopDos can read an external language catalog file to allow it to be localised for any country.

- Context-sensitive AmigaGuide help is available from within the program, both for windows and menu options.

- The clipboard is fully supported, allowing you to quickly copy SnoopDos output into your favourite editor or wordprocessor.

- Function calls can now be simultaneously written to a disk file while being displayed in the SnoopDos window.

So what are you waiting for? Go and give it a try!

See also: Contents Quick Start Introduction Main Window

1.5 workbench

STARTING SNOOPDOS FROM WORKBENCH

SnoopDos can be easily started from Workbench by double-clicking on its icon, or on the icon of any settings file saved earlier. You can store any SnoopDos command as a tooltype in the icon and each command will be executed by SnoopDos when it loads.

For example, this allows you to automatically open a log file on disk each time you run SnoopDos (using the OpenLog or AppendLog commands). You could also automatically open the function window using the OpenFunction command, or arrange that the main window always opens in a particular position using the MainWindowPos command.

SnoopDos carries out the following actions when started from Workbench:

- Checks icon tooltypes for Language , PatchRamLib and Settings commands
- Loads the default settings file, as defined by the Settings command. If no name has been set, then checks PROGDIR:, ENVARC: and S: for a file called SnoopDos.prefs and reads that instead if it exists.
- Reads any settings files that were passed in as project icons when SnoopDos was loaded. These files can contain any SnoopDos command, not just settings commands.
- Scans the tooltypes of all selected icons, starting with the program icon, and executes any recognised commands contained therein.

After all of this, the main window is automatically opened, unless a Hide or CX_Popup=No command was encountered.

If SnoopDos was already running when the icon is loaded, then the commands in the settings files and tooltypes will be executed by the running version instead. In this case, the Hide and CX_Popup keywords will be ignored, and the SnoopDos window will be automatically re-opened if it was hidden. This allows you to always show the SnoopDos window by clicking on a SnoopDos icon, even if that icon contains a Hide command in its tooltypes.

See also: Command Index Command syntax Settings files
CLI usage ARexx support

1.6 cli

STARTING SNOOPDOS FROM THE CLI

When you run SnoopDos from the CLI, it carries out the following actions:

- Checks the command line for Language , PatchRamLib and Settings options
- Loads the default settings file, as defined by the Settings command.

If no name has been set, then checks PROGDIR:, ENVARC: and S: for
a file called SnoopDos.prefs and reads that instead if it exists.

- Executes all other SnoopDos commands on the command line.

If SnoopDos was already running, then any commands on the command line will be executed by that version instead. For example, you could use the command SNOOPDOS OPENLOG "ram:SnoopDos.log" to make the current copy of SnoopDos open a new log file.

Typing SNOOPDOS HELP will display a summary of all the commands recognised by SnoopDos. Commands that need a parameter will be marked with an '*'.

If you intend to control SnoopDos from the CLI like this, you may want to investigate its ARexx capabilities. Using the RX command to send a command to SnoopDos will usually be much quicker than waiting for the SnoopDos executable to load a second time. An alias such as the following can be useful:

```
ALIAS SNOOPTELL "RX *"ADDRESS SNOOPDOS '[]'*""
```

This allows you to issue SnoopDos commands by simply typing SnoopTell <cmd>. If you do this, you'll need to double-up any quotes that might appear on the command line, so that SnoopTell OpenLog "ram:test file" would become SnoopTell OpenLog ""ram:testfile"".

If you accidentally forget to type "RUN SNOOPDOS" the first time you load the program, that CLI will be unavailable until SnoopDos exits. If this happens, you can tell SnoopDos to quit by typing CTRL-C inside the CLI window. This is particularly useful if you are running SnoopDos remotely via modem or over a network.

See also: Command Index Command syntax Settings files
Workbench usage ARexx support

1.7 arexx

SNOOPDOS AREXX SUPPORT

When SnoopDos is running, it creates an ARexx port called (predictably) SNOOPDOS. This allows you to control almost every aspect of the program from within ARexx scripts.

Here's an example of a simple ARexx script that configures SnoopDos to monitor one particular function, and then captures all calls to that function in a log file.

```
/* Sample ARexx script to log all calls to LoadSeg */

/* Check if SnoopDos is running; if not, then start it */
if ~show('P', 'SNOOPDOS') then do
    address command "run >nil: SnoopDos"
    address command "waitforport SNOOPDOS"
    end
```

```
   address SNOOPDOS                          /* Commands now go to SnoopDos  */
   logformat "%t %15p %6a %37n %6o %r"       /* Install a custom log format  */
   functions none                            /* Disable all functions        */
   loadseg on                                /* Re-enable LoadSeg function   */
   appendlog "ram:SnoopDos.txt"              /* Open log file                */
   addlog "(Monitoring only LoadSeg)"        /* Add info message to log file */
   logformat none                            /* Remove our custom format     */
```

(Even though we remove the custom log format at the end of the script, it will remain in use until the log file is closed. The next time a log file is opened, it will use the main window format instead.)

If this code is stored in a file called LoadSeg.Rexx, you can execute it by typing RX LOADSEG.

You can tell if a particular SnoopDos command succeeded or not by checking the RC variable afterwards. This will be set as follows:

```
    RC = 0    Command completed successfully
    RC = 10   Command requires a parameter
    RC = 20   Command could not be completed (parameter may be invalid)
    RC = 30   Command was not recognised by SnoopDos
```

You can use the ARexx command SIGNAL ON ERROR to cause ARexx to stop if it encounters a SnoopDos error (usually, it will quietly ignore all errors).

If you want to get a feel for using SnoopDos commands, you can use the LoadSettings command with a filename of "CON:////SnoopDos/CLOSE" to open a command window that lets you interactively enter commands and see their effect.

See also: Command Index Command syntax Settings files
Workbench usage CLI usage

1.8 settingsfiles

SNOOPDOS SETTINGS FILES

A SnoopDos settings file is any text file which begins with the line "<SnoopDos Settings>" and contains SnoopDos commands. While settings files are usually generated automatically using the SaveDefSettings command, they can also be created using any standard text editor.

In fact, the term "settings file" is slightly misleading, since such a file can contain any SnoopDos command, not just those that control program settings. For example, a settings file can carry out actions like opening a new log file, opening a particular arrangement of windows, or adding custom output to an existing log file.

The format of the file is fairly free form. There should be one command per line, and anything following a semicolon is treated as a comment. See the section on Command syntax for more details.

When you run SnoopDos, it looks for a default settings file and executes all the commands it contains. You can specify the name of this file

explicitly using the Settings command.

If you don't specify a default filename, then SnoopDos will look in the PROGDIR:, ENVARC: and S: directories for a file called SnoopDos.prefs. If it can't find a file in any of those three directories, then it will use the name ENVARC:SnoopDos.prefs when you ask it to save default settings.

Note that this gives you some flexibility as to where you store your default settings. If you don't like the idea of the SnoopDos settings file wasting RAM by being stored in ENVARC: (and thus ENV:) then you can simply move the file to S: or the program directory instead, and SnoopDos will always use the new directory with no additional work needed on your part.

Since SnoopDos erases the old defaults file whenever you save default settings, it's not a good idea to include commands other than those written by SnoopDos itself in the defaults file -- they would get overwritten the next time you saved your settings. Instead, you could consider adding such commands as tooltypes in the SnoopDos icon.

However, there is another slightly more convoluted way to execute additional commands while SnoopDos is initialising. You can create a default settings file (e.g. ENVARC:SnoopDos.prefs) which contains something like the following:

```
<SnoopDos Settings>
;
Settings "S:SnoopDos.set"   ; This is where the defaults will be stored
LoadDefSettings             ; Now load in the defaults
;
;  Additional commands go below here
;
OpenLog "ram:SnoopDos.log"  ; Let's start a new log file
FormatWindowPos=400,300     ; Ensure window position is always the same
OpenFormat                  ; Open the format window for easy access
; etc.
```

As you can see, the first thing this command file does is to change the name of the default settings file. This ensures that if you click on Save Settings from within SnoopDos, the new file will be overwritten rather than this file.

Next, those settings are loaded in. (SnoopDos won't automatically load the settings unless you tell it to.)

After this, any additional commands can appear. For example, you could make SnoopDos always start up Paused or Disabled. If there are certain settings that you always want set to a particular value, overriding any changes made via Save Settings, then this is a good place to put them.

Note that while it is perfectly valid for one settings file to execute another (using the LoadSettings or LoadDefSettings commands) this oly works up to three levels of nesting. This prevents an infinite loop from occurring if you accidentally try and load a file from within itself.

There is one final trick you can do with settings files. If you load a settings file with a filename that corresponds to a valid console window, such as "CON:////SnoopDos/CLOSE", then SnoopDos will allow you to type in

commands interactively and see their effect. See the LoadSettings command for more information.

See also: Command Index Command syntax ARexx support
Workbench usage CLI usage

1.9 distribution

SNOOPDOS DISTRIBUTION

My distribution policy for SnoopDos is simple: it may be freely distributed for non-commercial purposes, as long as all the files in the original archive are present and have not been modified in any way.

No charge for SnoopDos may be made, other than a reasonable cost to cover the media and copying time.

If you wish to include SnoopDos on a magazine cover disk, you may do so as long as you send a complimentary copy of the magazine issue in which it appears to my home address. The entire package must be included on the disk, not just the executable. Please contact me in advance to ensure you have the latest version.

If you wish to supply SnoopDos as part of a commercial product, please contact me to discuss it -- I'm easy to please.

If you wish to supply SnoopDos as part of a CD-ROM compilation of freeware, you may do so as long as the disc contains at least 20 Mb of other freeware. Otherwise, contact me first please. Explicit permission is granted to distribute SnoopDos on any CD-ROM produced by Fred Fish / Amiga Library Services, and on any officially supported Aminet compilation CD-ROM.

I reserve the right to publically poke fun at anyone who fails to abide by these distribution rules, along with more severe action if appropriate.

Note: The source code for SnoopDos is available in a separate archive. You should be able to find it on Aminet in the util/moni directory, along with the main SnoopDos archive.

See also: Contents History Credits Author

1.10 history

SNOOPDOS HISTORY

The very first version of SnoopDos was written back in May 1990 at the suggestion of a friend. It proved popular, and was officially released to the world as Version 1.0 in September 1990.

Over the next few years, SnoopDos received a number of minor upgrades, with the final public release being V1.7 in December 1992. A further upgrade, V1.7a in May 1993, had limited circulation.

In Autumn of 1993, work began on a complete rewrite of SnoopDos. A closed user group was set up on the UK conferencing system, Cix, and an initial prototype of the GUI was constructed using Commodore's ToolMaker. This was used to obtain early feedback on the interface from my testers, and acted as the basis for the first real version.

In January 1994, I found myself unexpectedly transferred to St Paul, Minnesota as part of my job. After acquiring a second-hand A3000 (thanks Scott!), I was able to resume work on the project. During these three months, SnoopDos gained most of its functionality, along with a few neat features like the drag & drop format editor.

I returned to Ireland at the end of April, and work continued throughout the summer. Many bugs were squashed, the command language was fleshed out, refinements were made, and documentation was written. In July, the feature set was locked down, and August was spent finetuning. After almost a year, SnoopDos 3.0 was ready for release.

When you consider that the very first version of SnoopDos took less than a week to write, you have to wonder why it took so long to produce this new version. Either I've slowed down a lot, or it's a much more sophisticated program -- you'll have to decide for yourself.

A NOTE ABOUT PIRATE VERSIONS OF SNOOPDOS

You may have noticed the jump in version numbering from SnoopDos 1.7 to 3.0. This is partly to co-incide with the current version of Kickstart and partly to avoid confusion with pirate copies of SnoopDos.

For reasons best known to themselves, various individuals have seen fit to take the source code for SnoopDos (always freely available) and use it to generate new, unauthorised versions which are then released as official upgrades. At least one of these versions has been found to contain a trojan horse. Two particularly widespread hacks were numbered 1.6 and 2.0; if you have either version on your system, you should delete it immediately.

One way for me to reduce the chance of this happening in future is to stop distributing source code for SnoopDos. I have no particular wish to do this; I learnt many of my programming skills from studying other people's code, on Fish disks and elsewhere, and it's nice to be able to contribute something back to the programming community. Instead, I would ask anyone out there considering such an action to put their creative talents to better use by writing their own software.

In the meantime, I recommend that you try and ensure you obtain your copy of SnoopDos from a reputable source such as Aminet or Fred Fish's regular CD-ROM. If you run a BBS, please feel free to check with me first before putting a new version online, if you have reason to suspect it may be an unauthorised copy.

All versions of SnoopDos 3.0 and above are signed using my PGP public key (see Author for information on how to obtain this.) If you have PGP, you can use this digital signature to ensure that your copy has not been tampered with. Even if you don't have PGP, a missing signature is a good sign that something is wrong.

See also: Contents Distribution Credits Author

1.11 credits

SNOOPDOS CREDITS

A project like SnoopDos 3.0 would not be possible without the input of many people. A big thank you to all the users of SnoopDos 1.x who sent in helpful suggestions -- these were a big incentive to write Version 3.0.

Writing this version of SnoopDos provided a good opportunity for me to learn about the new Kickstart 2.0 and 3.0 features. Along the way, I ran into many problems, and I'd like to thank the following programmers for their useful coding advice and suggestions:

Ralph Babel, Christopher Feck, Mike Froggett, Ed Mackey,
Udo K Schuermann and Mike Sinz.

While any project of this size is unlikely to be completely bug free, there would be a great many more bugs without the help of my army of beta testers. Not only did they suffer through every variety of crash, they also made many useful suggestions for improvements:

Dean Ashton	Derek Holdon	Charles O'Reilly
Timothy Aston	Leon Hurst	Jolyon Ralph
Dan Barrett	Paul Kelly (Ireland)	Geoffrey Reeves
Fred Botton	Paul Kelly (England)	Tommy Rolfs
Charlie Chuck	David Malone	Toby Simpson
Rick Costas	Barry McConnell	Paul Wakeford
Jonathan Evans	Markus Moenig	Richard Waspe
Tommy Gibbons	Ian Moran	Ian Wellock
Jim Hawkins	Niall Murphy	Michael Witbrock

Sincere thanks to one and all.

See also: Contents History Distribution Author

1.12 author

HOW TO CONTACT THE AUTHOR

I can be contacted via Internet at:

ecarroll@maths.tcd.ie
ecarroll@cix.compulink.co.uk

If you have any bug reports, suggestions or just general comments about SnoopDos, I would like to hear from you. If you think you could do a better job designing an icon for SnoopDos than I did (which wouldn't be hard!) then I would also like to hear from you.

You can obtain my PGP public key by sending mail to ecarroll@maths.tcd.ie

with the subject line "PGPKEY". If you have an account on Bix or Cix, you can check my online resume instead (user name ecarroll).

I can also be reached by snail mail at this address:

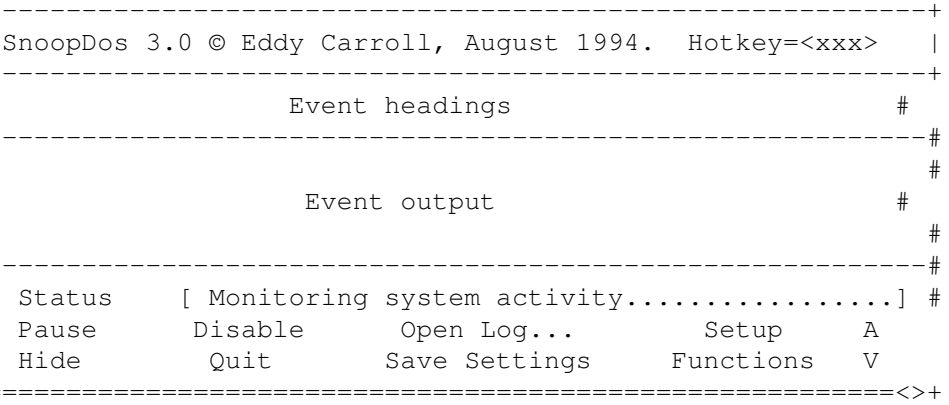
Eddy Carroll
The Old Rectory
Delgany
Co. Wicklow
Ireland

I regret that due to the volume of mail received, I cannot always reply to every letter. This is further compounded by the fact that I travel quite frequently. Please use email if possible.

See also: Contents History Distribution Credits

1.13 mainwindow

The main window looks like this. Click on any item for a more detailed description of its purpose.



The gadgets shown need not be displayed in the window -- you can turn them off via the Windows menu, or by using the HideGadgets and HideStatus commands. Even when hidden, the gadgets can still be accessed via keyboard shortcuts. All gadgets also have menu equivalents for convenience.

In addition to the underlined characters in the gadget labels, a range of other keystrokes are also recognised. See the Keyboard Shortcuts section for more details.

See also: Function window Settings window Format editor
Menu options Show Hide

1.14 eventheadings

EVENT HEADINGS

This line at the top of the main window displays the headings for each of the current output fields.

While the Format editor and Format command allow you to change the current headings, you can also use the mouse to make small adjustments directly in the main window. This is useful if you want to see a bit more of a particular field, such as the Target name field.

If you click on any heading, it is highlighted in white, and two vertical markers show the current boundaries of the output field. In addition, a short vertical line indicates the rightmost position in the current format (this is useful if you are trying to adjust the total width to fit neatly inside the current window).

While the mouse button is held down, you can move the heading left or right to adjust the position and width of the column. As you drag, you will see the other headings adjust accordingly. When you release the mouse button, the rest of the event output will be updated to reflect the new format.

If you click to the blank area at the right of a heading, then you can adjust the width of that heading, while leaving its position unchanged. If the heading title fills the whole width of the column, then you can click on the rightmost character of the title instead to achieve the same effect.

Usually, as you adjust a column, the headings to the right of the selected column remain in the same position. This keeps the overall width of the format the same, and is achieved by making one of the other columns narrower or wider, to compensate for the changes made in the selected column.

Sometimes it's convenient to be able to increase the overall width of a format -- for example, if you've just resized the main window to be wider. You can do this by holding down the SHIFT key while moving a heading. All the headings to the right of the selected heading will then move in synchronisation with it.

At any time while dragging a column, you can press the SPACE key to update the event text to match the new format. This is a little quicker than releasing the column and then clicking on it again, since it allows you to make fine adjustments while still keeping the mouse button pressed.

If you change your mind while dragging a column, you can restore the previous setting by pressing the ESC key or clicking the right mouse button. Once you release the left mouse button, the change becomes permanent. (You may find the Last Saved option on the settings menu useful in this case.)

See also: Main window Event output Format editor Format

1.15 eventoutput

EVENT OUTPUT

A large part of the main window is given over to event output. This is where the information about each function being monitored is displayed.

The exact details displayed for each function depend on the current event Format , but will usually include at least the target name, action, option and result output fields.

Because of the way SnoopDos works, it is possible for several programs to start executing monitored functions simultaneously. When this occurs, you may see several events listed without any text in the Result field. This is perfectly normal. As each call completes, its result field will be filled in accordingly.

Some functions which are expected to take a long time display "----" in the result field instead, to indicate that they are currently executing. These include the Execute , RunCommand and System functions.

You can use the cursor keys to move around the event output. Cursor up and down will move backwards and forwards through the event buffer, while cursor left and right will scroll the window horizontally. You can only scroll horizontally if the current event format is too wide to fit completely inside the window. To move in jumps greater than one character, hold down SHIFT or ALT while pressing the cursor key.

You can highlight any line in the event buffer by clicking on it with the mouse. The line will remain highlighted until you release the mouse button. This provides an easy way to match text in the Options and Result columns with the corresponding text in the Process Name and Target Name columns.

While the highlight is displayed, you can move the mouse up and down to select different lines. If you try and move off the top or bottom of the window, the buffer will scroll in the appropriate direction.

Note that no new output will be displayed while a line is highlighted. If you keep a line highlighted for a long time, you may find that the event information currently displayed has vanished from the buffer, and new events will start to be printed instead -- if this happens, the entire window will be automatically refreshed when you release the button.

If you use a click-to-front window utility, you may find the brief flash of a row being highlighted when you click the SnoopDos window to the front distracting. You can avoid this by configuring SnoopDos to only allow highlighting of rows when a qualifier key like SHIFT, ALT or CTRL is held down. See the RowQualifier command for more details.

See also: Main window Event headings Format editor Format
RowQualifier ShowGadgets ShowStatus TextSpacing

1.16 statusline

MAIN WINDOW / STATUS LINE

The SnoopDos status line can display a variety of messages, depending on what SnoopDos is currently doing. Here's a summary of the information that will appear on the status line:

- when disabled, the time that Disable was selected
- when paused, the time that Pause was selected

- when logging to a disk file, the name of the file
- when logging to a device, the name of the device
- when loading a settings file, the name of the file
- when saving a settings file, the name of the file

If none of these is currently in progress, then SnoopDos will simply say it is monitoring system activity.

The information displayed in the status line is sufficient to allow you to remove the gadgets in the main window if you choose. This provides more room for event text to be displayed.

See also: HideGadgets HideStatus Main window
ShowGadgets ShowStatus

1.17 mainopenlog

MAIN WINDOW / OPEN LOG

The "Open Log..." gadget in the main window is used to start a new log file. The actual gadget label will vary depending on the current LogMode setting. You can change this in the Setup window. The four possible settings are:

Open Log...	Prompt for the new log file using the ASL file requester
Start Log	Open a new log file using the current LogName
Append Log	Add to an existing log file as defined by LogName
Serial Log	Send log file output to a terminal on the serial port

When a log file is open, the gadget label is changed to Close Log .

You can always open a new log file via the ASL file requester by choosing the Open log option on the Project menu, regardless of the current label on the gadget.

See also: Main window AppendLog OpenLog OpenSerialLog

1.18 mainkeys

MAIN WINDOW / KEYBOARD COMMANDS

The following keyboard shortcuts are recognised in the main window:

Cursor Left	Scrolls left one or more columns in the window
Cursor Right	Scrolls right one or more columns in the window
Cursor Up	Scrolls back one or more lines in the event buffer
Cursor Down	Scrolls forward one or more lines in the event buffer
RETURN	Scrolls forward one line in the event buffer
TAB	Performs a Pause (if necessary) and a SingleStep
SPACE	Performs a SingleStep while paused
SHIFT-TAB	Unconditionally cancels Pause or Disable

```
          M  Opens the  Format editor  (like the menu shortcut)
      SPACE  Redraws event text while adjusting  Event headings
        ESC  Cancels any column drag currently in progress

     CTRL-C  Selects the  Quit  gadget
     CTRL-D  Activates the  Disable  gadget
     CTRL-E  De-activates the  Disable  gadget
     CTRL-F  Brings the main window to the front
```

Most gadgets also have a keyboard shortcut, indicated by an underlined letter in the gadget title.

If you press the keyboard shortcut for a button gadget, it will not take effect until you release the key. If you change your mind after pressing it, you can press ESC (with the original key still held down) and the gadget selection will then be ignored.

For many gadgets, holding down SHIFT while you type the shortcut key will produce slightly different behaviour. For example, holding down SHIFT while you type the shortcut key for the Setup gadget in the main window will close the Setup window instead of opening it.

See also: Main window

1.19 settingswindow

The settings window lets you control overall settings which govern how SnoopDos operates. Click on any item for a more detailed description.

```
    +-------------------------------------+
    | SnoopDos Settings...................|
    +-------------------------------------+
    |  Hide method        Hotkey          |
    |  Screen type        Screen name     |
    |  Log mode           Log file        |
    |  File I/O           Window font     |
    |  Buffer size        Buffer font     |
    |                                       |
    |  Buffer format              Edit    |
    |  Logfile format             Copy    |
    |                                       |
    |    Use        Undo      Cancel      |
    +-------------------------------------+
```

See also: Main window Function window Format editor
OpenSetup SetupWindowPos

1.20 settingscopy

SETUP WINDOW / COPY GADGET

This gadget copies the current Format string into the LogFormat . Usually, the format used when sending event information to a log file is the same as that used to display the information in the main window. However, you can choose a separate format if you wish.

This allows you to design your log file format using the format editor and the main window. Once you are happy with it, you can Copy it into the LogFormat gadget, and then restore the original window format by clicking Undo in the format editor.

If you hold down the shift key while selecting the gadget, then instead of copying the current buffer format, it will clear the log format. When the log format is clear, it indicates that the current buffer format should be used when logging to a file.

See also: Setup window Format editor Format LogFormat WindowWidth

1.21 settingsbuttons

SETUP WINDOW / GADGETS

The USE, UNDO and CANCEL gadgets in the Setup window are used to control whether or not the current window settings are made permanent.

When you change a gadget in the Setup window, the change takes effect immediately. For example, if you choose a new HotKey , you will see the main window titlebar updating to reflect the change. Selecting Use will close the Setup window and make all such changes permanent. If you want to close the window while maintaining the current settings, then choose this option.

Selecting Undo will restore all the settings in use when you first opened the Setup window. This allows you to recover from any accidental changes you might have made.

Selecting Cancel will restore the original settings, and then close the Setup window. This will cause any changes that have been made to the settings since the window was opened to be discarded.

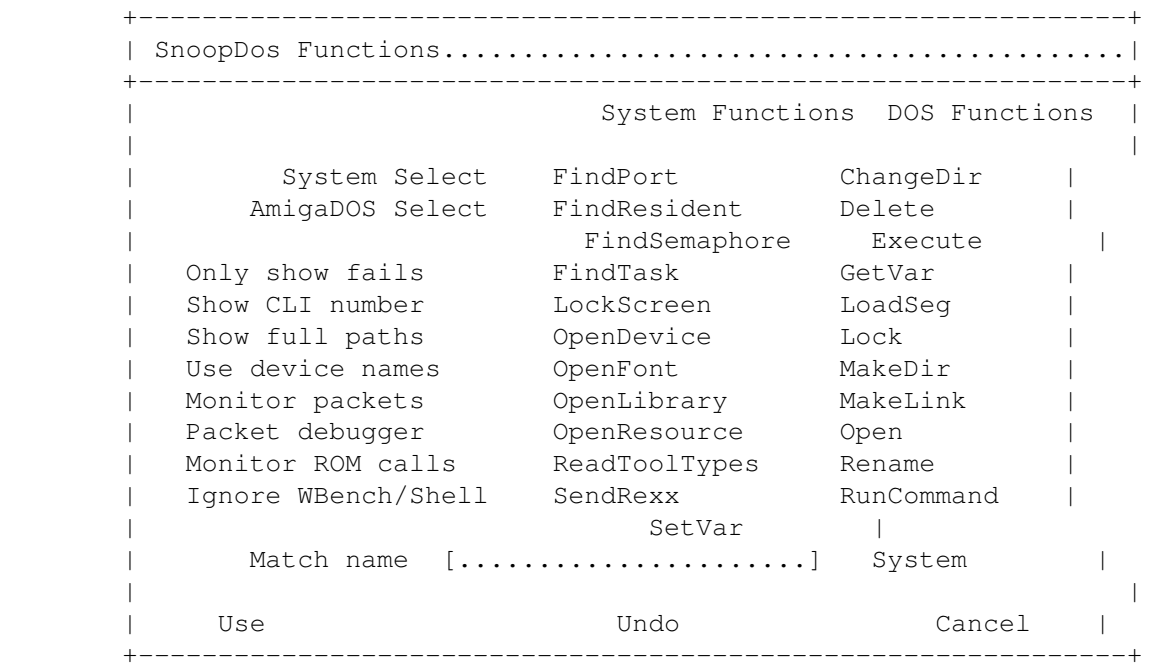
There are two exceptions to the earlier statement that all changes made in the Setup window take effect immediately. The first is the BufferSize gadget, which controls the size of the event buffer. Since changing the buffer size causes the current buffer contents to be lost, the change doesn’t take effect until you select Use. This allows you to change your mind if you alter it by mistake.

The second exception is the LogFormat gadget. If a log file is currently open when you alter it, the change will not take effect until after the log file is closed. This avoids producing a confusing log file with a variety of different formats.

See also: Setup window

1.22 functionwindow

The SnoopDos function window allows you to choose which events are monitored. Click on any item for a more detailed description.



See also: Main window Settings window Format editor
OpenFunction FunctionWindowPos

1.23 selectgadgets

FUNCTION WINDOW / SYSTEM AND DOS SELECTORS

These gadgets allows you to quickly select or deselect all the functions listed in the System or AmigaDOS column of the Function window.

There are three possible settings. SELECTED is the usual setting, and indicates that a mixture of functions are currently being monitored.

ALL and NONE indicate respectively that all or none of the functions in each column are currently being monitored. The function checkbox gadgets will reflect this.

As long as you don't click on any of the function checkboxes, SnoopDos will remember the last SELECTED configuration, and you can then restore that configuration by changing the selection back to SELECTED again. However, if you change any checkbox gadget while in the ALL or NONE state, then the state changes back to SELECTED, and SnoopDos will no longer be able to restore the original settings.

This is more difficult to explain than to understand. Play with all the gadgets and everything should become clear.

You can use the Functions command to enable or disable groups of functions from within a script.

See also: Function window Functions

1.24 functionbuttons

FUNCTION WINDOW / GADGETS

The USE, UNDO and CANCEL gadgets in the Function window are used to control whether or not the current function settings are made permanent.

When you change a function gadget in the window, the change takes effect immediately. Selecting Use will close the window and make all such changes permanent. If you want to close the Function window while maintaining the current function settings, then choose this option.

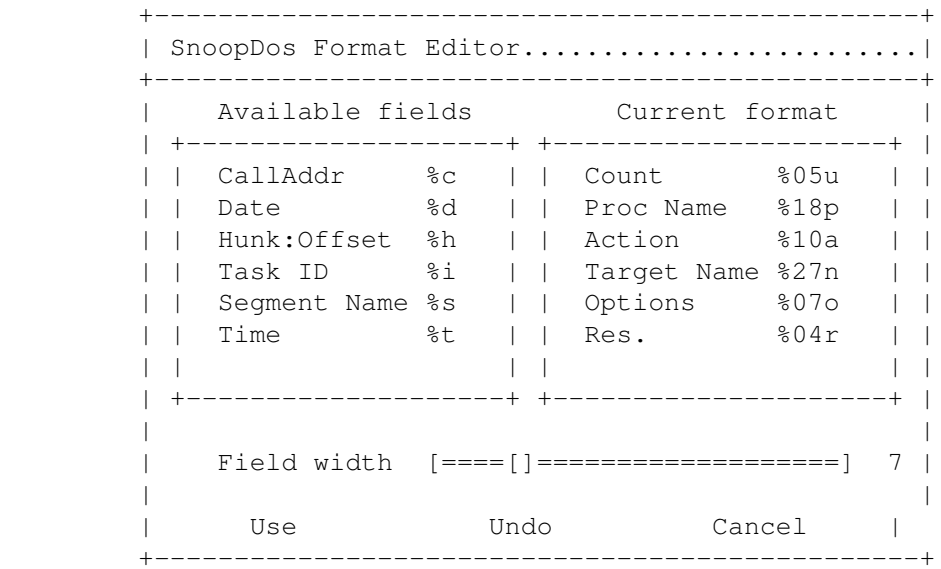
Selecting Undo will restore all the function settings in use when you first opened the window. This allows you to recover from any accidental changes you might have made.

Selecting Cancel will restore the original function settings, and then close the window. This will cause any changes that have been made to the settings since the window was opened to be discarded.

See also: Function window

1.25 formateditor

The format editor lets you choose what information is displayed in the main window for each monitored event. Click on any item for a more detailed description.



The current format is listed in the righthand box while other fields not currently in use are listed in the lefthand box. You can use the mouse to drag fields from their current position to a new position. As you do this, you will see the main window updating to reflect your change.

You can also use the cursor keys to edit the format. Cursor Up and Down select a field to edit in the current format, and Cursor Left and Right adjust the field width.

See also: Main window Settings window Function window
Format OpenFormat FormatWindowPos

1.26 Format width gadget

FORMAT EDITOR / FIELD WIDTH SLIDER

The format width slider gadget allows you to adjust the width of the currently selected format output field. As soon as you release the slider, the main window will be updated to reflect the new width of the selected field. The width of each event in the current format is displayed to the right of the event name (between the '%' and the format identifying letter).

Only fields in the current format can have their widths adjusted. If there is no currently selected field, then the slider is disabled.

See also: Format editor Main window Format

1.27 formatbuttons

FORMAT EDITOR / GADGETS

The USE, UNDO and CANCEL gadgets in the Format editor are used to control whether or not the current format is installed permanently.

When you make changes in the format editor, you can immediately see the effect of those changes in the main window. Selecting Use will close the format editor and make those changes permanent. If you want to close the format editor while maintaining the current settings, this is the option to choose.

Selecting Undo will restore the format in use when you first opened the format editor. This allows you to recover from any accidental changes you might have made.

Selecting Cancel will restore the original format, and then close the format editor. This will cause any changes that have been made to the format since the editor was opened (including any changes made in the main window) to be discarded.

See also: Format editor Format Event heading

1.28 Action output field

%A -- ACTION

The action output field is one of the most important parts of any event. It shows what action a program is trying to carry out. The possible actions correspond roughly to the functions listed in the Function window, but you may also see other actions listed occasionally.

In particular, if you have the PacketDebugger turned on, you will see many actions of the form #XXXXXX (such as #FINDOUTPUT and #COPY_DIR). These are raw AmigaDOS packets, and of interest only to programmers.

Any actions prefixed with an asterisk correspond to operations carried out using direct packet i/o, rather than dos.library function calls. You will only see these if you have enabled the MonitorPackets option. Typically, such calls are only made by programs compiled using GNU C.

See also: Format editor Main window Format

1.29 Call Address output field

%C -- CALL ADDRESS

This output field displays the memory address from which a monitored function was called. On its own, this is fairly meaningless. However, if you are a programmer, you can use this information to locate the piece of code that made the call, and see what it tries to do next.

The address is displayed as a 7 digit hex number since very few, if any, Amigas have memory above $0FFFFFFF. If your Amiga happens to be the exception, then ensure that you make this field at least 8 characters wide to catch any wider addresses.

Also note that the value actually represents the address of the first instruction after the call to the monitored function. If you want to see the instruction that made the call, subtract 6 from this value.

If another program has patched a function after SnoopDos, then SnoopDos will usually be unable to determine the correct return address -- instead, the address of the second program's patch code will be displayed. If this causes a problem, then you should not enable SnoopDos's monitoring of the function in question until after the second program has been loaded.

See also: Format editor Main window Format

1.30 Date output field

%D -- DATE

This output field displays the date an event took place on, in the standard AmigaDOS format DD-MMM-YY. You are unlikely to find this useful unless you

intend running SnoopDos for several days at a time.

See also: Format editor Main window Format Time field

1.31 Hunk:Offset output field

%H -- HUNK:OFFSET

This output field is only of use to programmers, and then only when SegTracker is loaded. It displays the hunk and offset from where the current function was called, in the format HH:OOOOOO. (Both the hunk and the offset are given in hex).

You can use this information in conjunction with the Segment Name output field and the FindHit utility to identify which line in a program's source code was responsible for making the function call. This can be invaluable when you are debugging your code.

If SegTracker is not loaded, or if it cannot identify which module the function was called from, then this field will usually be blank. However, if the function was called from ROM, then the offset from the beginning of the ROM module will be displayed instead.

The same caveat discussed for the Call Address field regarding programs that patch functions after SnoopDos has been installed also applies here.

See also: Format editor Main window Format

1.32 TaskID output field

%I -- TASK ID

This output field displays the task address of the program that called the current function. The address is displayed as a 7 digit hex number. If your Amiga happens to have memory above $0FFFFFFF, you will need to widen this field to allow all 8 digits to be displayed.

This field can be useful in identifying the process that called a function if there are several processes running on your system with the same name. You can use a tool such as ARTM or XOper to find out more information about the process based on its address.

See also: Format editor Main window Format

1.33 TargetName output field

%N -- TARGET NAME

This output field displays the most vital piece of information associated with the monitored function. This will vary from function to function, but

is typically the name of the requested system resource or disk file.

This field is unusual in that it can be displayed aligned on the left or the right -- all the other fields are always displayed left aligned. See the LeftAligned and RightAligned commands for more details.

See also: Format editor Main window Format

1.34 Options output field

%O -- OPTIONS

This output field displays any secondary options associated with the item displayed in the TargetName output field. This will vary from function to function.

For example, with the Open() function, this will show whether the file is being opened for reading or writing. With the RunCommand() function, it will show the stack size of the command being run.

If a particular function has no secondary options, then this field will remain blank.

See also: Format editor Main window Format

1.35 ProcessName output field

%P -- PROCESS NAME

This output field displays the name of the process that called the function being monitored.

Note that while this indicates the program that was responsible for the call, the program displayed may not have made the call directly. For example, it may have called a function in an external library, and that library may be making the call on the program's behalf. If this distinction is important, you can use the Segment Name output field to get more precise information.

Sometimes you may find you have several processes running which all have the same name. In this case, you can use the Task ID output field to distinguish between them. Alternatively, if they were all run from the CLI instead of Workbench, you can use the ShowCLI command to display the CLI number of each process in brackets at the start of the name.

See also: Format editor Main window Format

1.36 Result output field

%R -- RESULT

This output field displays the result of each function call. This will usually be either OK or Fail, depending on the outcome.

Some functions behave slightly differently. In the case of the Execute , RunCommand and System functions, this field will still display Fail if the function failed, but will return the numeric program return code if the function succeeded. Since these functions typically take some time to execute, the result will be shown as "----" while the command is executing.

If you Pause SnoopDos, then each task that calls a monitored function will be put to sleep. When this occurs, the result field displays "WAIT" to indicate the task is no longer active. This will be replaced by the appropriate return code when execution is resumed.

If you have enabled the PacketDebugger , then this field will not only contain OK or Fail, it will also contain the value or values returned by the DOS device being monitored. To see this additional information, you will need to make the field wider than its normal 4 character width.

If you have been using the Packet debugger or the MonitorPackets option, you may occasionally see a result code of "Missed". This simply indicates that SnoopDos couldn't determine the result code of a particular packet operation, most likely because you disabled packet monitoring while the operation was still in progress.

See also: Format editor Main window Format

1.37 Segment Name output field

%S -- SEGMENT NAME

This output field is only of use when SegTracker is loaded. It displays the name of the module that called the current function. It is usually used in conjunction with the Hunk:Offset field.

The Segment Name will often, but not always, be the same as the process name. They can differ if a program calls a module outside its own code, such as a ROM function or a function in an external disk library. If that function then calls any functions monitored by SnoopDos, it will be identified independently of the original process. This is useful if you are trying to find out what part of the system is responsible for a particular operation.

If SegTracker is not loaded, SnoopDos can still identify calls made from within resident ROM modules. In this case, the ROM module name will be displayed.

Since calculating the calling module is a relatively time consuming process, SnoopDos only actually performs the calculation when it needs to display the segment name on the screen or output it to the log file. This means that if you don't have any interest in the segment name, then SnoopDos isn't slowed down unnecessarily.

However, this has a side effect which can trip up the unwary. If the module that made a function call is unloaded from memory before SnoopDos has a chance to print the details about the functions it called, then the module name and hunk:offset can't be calculated; a "Module not found" message will be displayed instead. This is most likely to happen if SnoopDos is hidden (since it won't display anything until the window is re-opened) but it can also happen if a program exits quickly while it is being monitored.

To guard against this, you can take a few simple precautions. Make sure you add the Segment Name field to your output format before you start monitoring.

Raise the priority of SnoopDos using the Task priority menu option so that it's higher than the priority of the processes you will be monitoring. This lets SnoopDos call SegTracker before the module has a chance to unload itself.

Finally, if you plan on running SnoopDos hidden in the background, keep a log file open, even if it's only to NIL:. This will make sure that SnoopDos always resolves segment names as soon as possible. If you have the main window open, then this won't be necessary.

See also: Format editor   Main window   Format

1.38 Time output field

%T -- TIME

This output field provides a simple way to timestamp each function that SnoopDos monitors. It is particularly useful if you will be running SnoopDos unattended for long periods of time.

The time is displayed in the format HH:MM:SS. If you are only interested in the hours and minutes, you can narrow the field width from 8 to 5 characters wide.

If SnoopDos will be running for several days, you may also want to consider using the Date output field.

See also: Format editor   Main window   Format

1.39 Count output field

%U -- COUNT

This output field simply provides a unique sequence number that identifies the position of an event in the buffer. Events are numbered starting from one and each new event increments the count accordingly.

If you are logging events to a file, this field is particularly useful

because it provides an easy way of matching up partially completed events
with the completed version which appears later on in the file. See the
 OpenLog command for more information.

See also: Format editor   Main window   Format

1.40 SegTracker and FindHit

SEGTRACKER AND FINDHIT

SegTracker and FindHit are two useful utilities by Mike Sinz that form
part of the Enforcer package. SegTracker keeps track of the memory
occupied by any module loaded from disk, while FindHit allows a hunk and
offset within a module to be mapped back to a line number in source code.

For best results, SegTracker should be run immediately after SetPatch in
your startup-sequence. This lets it keep track of as many programs as
possible. Running SegTracker after SnoopDos has loaded will still work,
but it will not be able to identify programs which were already running
at that time.

You can usually find the latest version of Enforcer on Aminet in the
/pub/aminet/dev/debug directory.

See also: Hunk:Offset   Segment name

1.41 commandindex

SnoopDos supports over 100 commands which can be accessed via command
line arguments, Workbench tooltypes, settings files, or ARexx. See the
 Command syntax section for general information, or click on any
command in the list below for detailed information about that command.

AddLog	HideGadgets	Pause
AppendLog	HideMethod	Quit
AutoOpen	HideStatus	ReadToolTypes
BufferFont	HotKey	Rename
BufferSize	IconPos	RightAligned
ChangeDir	IgnoreShell	RowQualifier
ClearBuffer	Language	RunCommand
CloseFormat	LeftAligned	SaveBuffer
CloseFunction	LoadDefSettings	SaveDefSettings
CloseLog	LoadSeg	SaveSettings
CloseSetup	LoadSettings	SaveWindow
CopyBuffer	Lock	ScreenName
CopyWindow	LockScreen	ScreenType
CreateIcons	LogFormat	ScrollDown
CX_PopKey	LogMode	ScrollUp
CX_Popup	LogName	SendRexx
CX_Priority	MainWindowPos	Settings
Delete	MainWindowSize	SetupWindowPos
Disable	MakeDir	SetVar

```
DisableWhenHidden    MakeLink             Show
Enable               MatchName            ShowCLI
Execute              MonitorPackets       ShowFullPaths
FlushLog             MonitorROMCalls      ShowGadgets
FileIOType           OnlyShowFails        ShowStatus
FindPort             Open                 SingleStep
FindResident         OpenDevice           SimpleRefresh
FindSemaphore        OpenFont             SmartRefresh
FindTask             OpenFormat           StackLimit
Format               OpenFunction         System
FormatWindowPos      OpenLibrary          TaskPri
Functions            OpenLog              TextSpacing
FunctionWindowPos    OpenResource         Unpause
GetVar               OpenSerialLog        UseDeviceNames
GotoLine             OpenSetup            WindowFont
Help                 PacketDebugger       WindowWidth
Hide                 PatchRamLib
```

1.42 commandsyntax

```
SNOOPDOS COMMAND SYNTAX

SnoopDos understands over 100 individual commands. These commands can be
executed from the CLI, as Workbench icon tooltypes, as ARexx commands, or
in SnoopDos settings files. In each case, the syntax is the same.

There are two broad categories of commands: those that take a parameter
and those that don't. You can get a quick summary of commands by typing
SNOOPDOS HELP in a CLI window -- this will list all the supported commands,
and those that require a parameter will be marked with an '*'.

If a command takes a parameter, you can specify the parameter in the
following ways. Let's use the HideMethod command as an example (note
that all commands and parameters are case insensitive):

    HideMethod=Iconify
    HideMethod="Iconify"
    HideMethod Iconify
    HideMethod "Iconify"

Which of these four variants you use will depend on personal preference,
and on whether you are executing the command from ARexx, the CLI or an
icon tooltype. However, if a command parameter contains spaces, it must
be enclosed in quotation marks.

Most of the commands that don't need a parameter are used to control
various program settings, such as what functions to monitor. There are
several ways to select whether you want each setting on or off. For example,
let's look at the Lock command, which controls whether or not SnoopDos
monitors the Lock() function in dos.library. All of the following will
turn on Lock monitoring:

    Lock
    Lock=Yes
    Lock=On
```

```
    Lock=1

To turn off lock monitoring, you can say:

    NoLock
    Lock=No
    Lock=Off
    Lock=0

The '=' is optional in most cases, but is required if you are executing the
command from the CLI. If you are executing the command via ARexx, it's a
good idea to leave out the '=' to prevent it confusing the ARexx command
interpreter.

In general, most commands have both a positive and negative version. For
example, Disable=No, NoDisable, and Enable are all equivalent. Similarly,
you can say HideGadgets=Yes or ShowGadgets=No -- they both have the same
effect.

If SnoopDos doesn't understand a particular command, it will usually inform
you -- exactly how depends on the method used to invoke the command.
Commands executed from the CLI produce an error message in the CLI window,
while those executed from ARexx return a result code in the RC variable.
Unrecognised commands in icon tooltypes and settings files are silently
ignored.

If you want to experiment with SnoopDos commands, you can use the
 LoadSettings  command with a filename of "CON:////SnoopDos/CLOSE" to
create a window where you can type commands interactively.

See also:  Command Index    CLI usage    Workbench usage    ARexx usage
```

1.43 addlog

```
Command: ADDLOG "text"

Example: ADDLOG "Loading BBS..."

This command lets you output a custom message to the current SnoopDos
log file. It can be useful if SnoopDos is monitoring a DOS or ARexx
script which generates large amounts of output, and you want to
indicate when certain parts of the script were executed in relation to
the SnoopDos output.

See also:  Command Index    AppendLog    CloseLog    FlushLog
           LogFormat        OpenLog
```

1.44 appendlog

```
Command: APPENDLOG "filename"

Example: APPENDLOG "ram:SnoopDos.log"
```

This command opens a log file. Any output displayed in the SnoopDos window will also be written to this file. If the file already exists, SnoopDos will append its output to the existing contents of the file; otherwise, a new file is created.

Files opened using AppendLog have an advantage over those opened with OpenLog, in that it is possible to access the contents of the file from another program while SnoopDos is still writing to it. This is useful if you want to quickly review the events that have been logged so far -- you can simply TYPE the file in a CLI window.

If you are doing this, you can ensure that the file contents are up to date by selecting then unselecting the Pause gadget. This will write any information that might still be in the file buffer to the file.

See the OpenLog documentation for more information about how SnoopDos outputs information to a log file.

See also: Command Index AddLog CloseLog FlushLog
LogFormat OpenLog OpenSerialLog

1.45 autoopen

Command: AUTOOPEN
NOAUTOOPEN

This command enables or disables SnoopDos's AutoOpen setting. This setting can also be controlled via an option on the Windows menu.

When AutoOpen is enabled, SnoopDos will automatically open its window whenever something happens which proces output in the window. This is useful if you have configured SnoopDos to only show one or two event types, since it lets you run SnoopDos in the background and have it pop up as soon as those events happen.

This command is automatically written to the settings file when you save settings.

See also: Command Index MatchName

1.46 bufferfont

Command: BUFFERFONT "fontname.size"

Example: BUFFERFONT "courier.13"

This command selects the font used by SnoopDos when displaying events in the main window. You can also change the font by using the font requester accessible through the Settings window.

The font selected must be a non-proportional font, and you must give a size.

This command is automatically written to the settings file when you
save settings.

See also: Command Index   WindowFont

1.47 buffersize

Command: BUFFERSIZE <size>

Example: BUFFERSIZE 32

This command changes the size of the buffer SnoopDos uses to record
information about events. You can also change the buffer size using
the Buffer Size gadget in the Settings window.

The buffer size is given in kilobytes. As a very rough guide, each
event occupies about 100 bytes of memory, so you can store about 10
events per kilobyte. The actual size of each event varies depending on
the length of the file and process names, and several other factors.

Note that changing this setting in the Settings window only takes
effect when you select Use; this is in contrast to most of the other
gadgets which take effect as soon as you change them. This is because
the SnoopDos buffer is cleared when you change its size, and so the
action is left until the last possible moment.

If there is not enough memory available for the buffer size you request,
SnoopDos will allocate as large a buffer as possible using the available
memory. Note that SnoopDos requires contiguous memory for its buffer,
so if you have been running many applications before loading SnoopDos,
you may find less memory is available then expected.

This command is automatically written to the settings file when you
save settings.

See also: Command Index   ClearBuffer

1.48 changedir

Command: CHANGEDIR
         NOCHANGEDIR

This command enables or disables monitoring of the CurrentDir()
dos.library function. It is also available as a gadget in the
 Function window.

Programs call the CurrentDir() function when they want to change their
working directory to somewhere else. Files opened by the program are
then assumed to be in this directory if a pathname isn't explicitly
specified. This can sometimes show you where a program expects to
find certain files.

This command is automatically written to the settings file when you save settings.

See also: Command Index

1.49 clearbuffer

Command: CLEARBUFFER

This command erases the contents of SnoopDos's event buffer. All events which occur from this point on are numbered starting from one again. If you are about to start monitoring a new program, this is a convenient way to ensure that you don't get the new output mixed up with output from earlier programs.

This option is also available as an item on the Buffer menu.

See also: Command Index BufferSize

1.50 closeformat

Command: CLOSEFORMAT

This command closes the Format editor if it happened to be open. The current position of the window on the screen is remembered so that it can re-open in the same position next time.

You can also close the format window by holding down the shift key while selecting the Show format item on the Windows menu, or while double-clicking on the header line in the main window.

Yet another way to close the format window is to hold down the shift key while selecting the "Edit..." gadget in the Setup window.

See also: Command Index OpenFormat FormatWindowPos

1.51 closefunction

Command: CLOSEFUNCTION

This command closes the Function window if it happened to be open. The current position of the window on the screen is remembered so that it can re-open in the same position next time.

You can also close the function window by holding down the shift key while selecting the Show functions item on the Windows menu, or while clicking on the Function gadget in the main window.

See also: Command Index OpenFunction FunctionWindowPos

1.52 closelog

Command: CLOSELOG

This command closes any currently open log file. It is available as an option on the Project menu, and also as a gadget in the main window (it replaces the Open Log button while a log file is open).

See also: Command Index AddLog AppendLog FlushLog
LogFormat OpenLog

1.53 closesetup

Command: CLOSESETUP

This command closes the Setup window if it happened to be open. The current position of the window on the screen is remembered so that it can re-open in the same position next time.

You can also close the setup window by holding down the shift key while selecting the Show setup. item on the Windows menu, or while clicking on the Setup gadget in the main window.

See also: Command Index OpenSetup SetupWindowPos

1.54 copybuffer

Command: COPYBUFFER

This command copies the entire contents of the SnoopDos event buffer to the system clipboard, allowing you to easily paste it into another application such as a word processor or text editor. It can also be accessed as an item on the Buffer menu.

The data will be formatted according to the current buffer format. Since this can potentially generate a large amount of text, you may prefer to use the CopyWindow command instead, which only copies that data currently displayed in the main window.

See also: Command Index CopyWindow SaveBuffer SaveWindow

1.55 copywindow

Command: COPYWINDOW

This command copies the contents of the main window to the system clipboard, allowing you to easily paste it into another application such as a text editor or word processor. It is also available as an item on tbe Buffer menu.

If the current buffer format is too wide to fit completely in the main window, only that portion which is currently displayed will be copied. If this doesn't include enough information, you can use the CopyBuffer command instead and then delete any extra lines that you don't need.

See also: Command Index CopyBuffer SaveBuffer SaveWindow

1.56 createicons

Command: CREATEICONS
NOCREATEICONS

This command controls whether or not SnoopDos should create an icon when it saves a settings file. You can find a corresponding menu option to control this on the Settings menu.

Note that no icon is saved for default settings files, only for settings files that are saved using the SaveSettings command. If an icon is created for a settings file, then the settings in that file can be loaded at any time by double-clicking the icon.

When SnoopDos writes out icons, it tries to use the same image as that used by the SnoopDos program icon. If it couldn't locate a program icon, it uses a simple built-in icon instead.

This command is automatically written to the settings file when you save settings.

See also: Command Index SaveDefSettings SaveSettings

1.57 cx_popkey

Command: CX_POPKEY "key description"

Example: CX_POPKEY "ctrl alt d"

This command allows you to select what key sequence is used to open the SnoopDos main window when it is hidden. This corresponds to the Hotkey gadget in the Settings window. It is a synonym for the HotKey command.

Any standard commodities key description can be given. You can always tell which key sequence is currently in use by looking at the titlebar of the main window. If you enter an incorrect description, then the titlebar will display "<invalid>".

When you have defined a hotkey, SnoopDos can be called up at any time by pressing that key sequence. If the main window was already open, it will be brought to the front and activated. If it was open, but on a different screen, it will be re-opened on the current screen (as selected using the ScreenType command).

See also: Command Index CX_Popup CX_Priority HotKey

1.58 cx_popup

Command: CX_POPUP=YES
CX_POPUP=NO

This command controls whether or not SnoopDos should open its window when you run it. If you give the command while SnoopDos is already running, then the window will be opened or closed accordingly.

As a special case, if you run SnoopDos from an icon which contains the tooltype CX_POPUP=NO, it will only have an effect the first time SnoopDos is run. If you run it again while SnoopDos is hidden the background, then the main window will be automatically opened, overriding this tooltype.

This ensures that a hidden SnoopDos window can always be re-opened by double-clicking on an icon, even if the icon contains a tooltype of CX_POPUP=NO.

This command is identical to the Show and Hide commands. It is included for compatibility with other commodities.

See also: Command Index CX_PopKey CX_Priority

1.59 cx_priority

Command: CX_PRIORITY=n

Example: CX_PRIORITY=5

This command is used to control the priority of the hotkey used to call up the SnoopDos window when SnoopDos is running as a commodity.

Most people will never need to change this from its default value of zero. However, occasionally you may find that another commodity is intercepting several keystrokes, one of which is the SnoopDos key. By increasing the value of CX_Priority, you can ensure that SnoopDos gets precedence.

Alternatively, by setting CX_Priority to a negative value, you could ensure that the other application gets priority.

See also: Command Index CX_PopKey CX_Popup

1.60 delete

Command: DELETE
NODELETE

This command enables or disables monitoring of the DeleteFile()
dos.library function. It is also available as a gadget in the
 Function window.

The DeleteFile() function is used to delete a file from disk. Programs
often delete temporary files when they no longer need them. If a file
seems to constantly vanish, and you can't figure out why, monitoring
DeleteFile() may show you which program is responsible.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index

1.61 disable

Command: DISABLE

This command provides a simple way to completely disable SnoopDos
while still leaving it ready for action in the future. It corresponds
to the Disable gadget in the main window and the Disable option on
the Project menu. In addition, CTRL-D is recognised as a keyboard
shortcut for Disable, for SnoopDos 1.x compatibility. This works even
when sent via an external log device such as CON: or AUX:.

When SnoopDos is disabled, it removes all its system patches and enters a
dormant state where it has no effect on system performance. If a log file
is open, then any outstanding buffered output is written to the file, along
with a message indicating what time SnoopDos was disabled at. (If flushing
the output file is all you want to do, the FlushLog command is an
easier way to achieve this.)

See also: Command Index   DisableWhenHidden   Enable   Pause

1.62 disablewhenhidden

Command: DISABLEWHENHIDDEN
         NODISABLEWHENHIDDEN

This command controls whether or not SnoopDos continues to monitor system
activity when it is running in the background as a commodity. There is a
corresponding menu option in the Windows menu.

If DisableWhenHidden is set, then SnoopDos automatically disables monitoring
when it is hidden (using the Hide command). When the SnoopDos window is
re-opened, the previous monitoring state is restored, unless monitoring was
re-enabled using the Enable command while SnoopDos was hidden -- in that
case, no change is made.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index Disable Enable Pause

1.63 enable

Command: ENABLE

This command is the counterpart to the Disable command. It undoes the effect of an earlier Disable and allows SnoopDos to resume monitoring functions. This corresponds to the Disable gadget in the main window, and the Disable menu option in the Project menu.

For compatibility with SnoopDos 1.x, CTRL-E will also select Enable. This works even when sent via an external log device such as AUX: or CON:. Yet another keyboard shortcut is SHIFT-TAB, which will also perform an unconditional Unpause .

If a log file is open, the time at which monitoring was enabled is written to the file.

See also: Command Index Disable DisableWhenHidden Pause

1.64 execute

Command: EXECUTE
NOEXECUTE

This command enables or disables monitoring of the Execute() dos.library function. It is also available as a gadget in the Function window.

Older applications call Execute() to run an external command from inside the main program. For example, a program might try to run Commodore's Colors utility to provide a simple way of editing the screen colours. Most modern software will call RunCommand or System instead.

The Target name output field shows the command line being executed by the application. Usually, only a single command is being executed; in this case, the Options output field will show "Single". If the application has provided additional commands to execute afterwards, then "Batch" will be displayed instead.

Batch mode is most commonly used to create a new CLI or Shell; the additional commands are then read from a console window rather than passed directly by the application.

The Result output field will show whether or not the command line was loaded successfully. Note that this doesn't indicate that the command itself completed successfully, only that AmigaDOS was able to locate the command on disk and load it into memory.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index RunCommand System

1.65 fileiotype

Command: FILEIOTYPE=AUTOMATIC | IMMEDIATE | BUFFERED

Example: FILEIOTYPE=IMMEDIATE

This command determines what method of buffering SnoopDos uses when capturing information to a log file. It is also available as a gadget in the Settings window

SnoopDos distinguishes between logging to a disk file, and logging to a device such as PRT:, AUX:, or CON:. If you select AUTOMATIC then SnoopDos will automatically use buffered output for disk files and disable buffering when logging to devices.

If you select IMMEDIATE, then SnoopDos will never use buffered output. This ensures that the information written to the log file (whether it's on disk or on a physical device such as a printer) will always be completely up to date. However, this can slow things down a little since one file operation needs to be carried out for every single event that SnoopDos monitors.

If you select BUFFERED, then SnoopDos will always buffer output, even if logging to a device. The buffering works by waiting until up to 8K of event data has been collected, and then outputting all the data in one operation. This is quicker than IMMEDIATE, but has the disadvantage that the log output may not always be up to date. In the case of a printer, for example, you would see no output for a while, and then suddenly 100 lines would all be printed at once.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index FlushLog

1.66 findport

Command: FINDPORT
NOFINDPORT

This command enables or disables monitoring of the FindPort() exec.library function. It is also available as a gadget in the Function window.

Programs call FindPort to rendezvous with other applications on the system. It is most commonly called to locate another application's ARexx port, but can also be used to allow several products from the same company to work together.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index SendRexx

1.67 findresident

Command: FINDRESIDENT
NOFINDRESIDENT

This command enables or disables monitoring of the FindResident()
exec.library function. It is also available as a gadget in the
Function window.

The Kickstart ROM on all Amigas contains a number of resident modules --
libraries, devices, resources, and so on. The FindResident() function
searches the list of all ROM modules for a particular name.

Most programs have no need to call FindResident() directly, but you will
sometimes see ROM libraries calling it if your current format string
includes the Segment name field. In normal use, you probably won't
find it a useful function to monitor.

SnoopDos automatically writes this command to the settings file when
you save settings.

Note for OS experts: The background ramlib process calls FindResident
frequently when searching for libraries and devices. SnoopDos deliberately
ignores these calls to keep the amount of information displayed down to a
reasonable level.

See also: Command Index PatchRamLib

1.68 findsemaphore

Command: FINDSEMAPHORE
NOFINDSEMAPHORE

This command enables or disables monitoring of the FindSemaphore()
exec.library function. It is also available as a gadget in the
Function window.

This is similar to FindPort but not as commonly used. It allows access
to a piece of data to be arbitrated so that only one task at a time can
access it.

It is often used to allow several copies of a single program to run at
once, with all copies sharing access to a single set of data. The main
reason you might want to monitor it is if you are writing a program
yourself that makes use of it.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index

1.69 findtask

Command: FINDTASK
NOFINDTASK

This command enables or disables monitoring of the FindTask() exec.library function. It is also available as a gadget in the Function window.

The FindTask() function allows a program to locate another task on the system by name. Having located the task, the program can then proceed to signal that task or communicate with it in some other way, or even try and kill it off. Generally, this function is used by utilities to locate system tasks, rather than by applications.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index

1.70 flushlog

Command: FLUSHLOG

When SnoopDos is logging output to a disk file, it usually buffers up output in memory and only writes it to disk when the buffer is full (about every 100 lines or so). This is much more efficient than writing every line of output to disk separately, but it has the disadvantage that if the system crashes, you may lose the most recent lines of output.

This command instructs SnoopDos to flush any outstanding output in the buffer to the log file, thus bringing it up to date. It has no effect if no log file is open.

A quick way to do this interactively is to Pause and then Unpause SnoopDos using the Pause gadget.

Note that you can use the FileIOType command to control whether or not SnoopDos uses buffering when writing to the log file.

See also: Command Index AddLog AppendLog CloseLog
LogFormat OpenLog

1.71 format

Command: FORMAT "formatstring"

Example: FORMAT "%u %p %a %n %o %r"

This command defines how SnoopDos should format the output detailing each event it monitors. It is also available as a gadget in the Settings window. You can edit the format string using the Format editor, or by manipulating columns in the main window using the mouse.

The format string is similar to that used by printf in the C programming language. It contains a list of format fields, with each field corresponding to one piece of information about a monitored event. The fields are as follows:

Action	%a	The name of the function being called
CallAddr	%c	The address the function was called from
Date	%d	The date the function was called
Hunk:Offset	%h	The module hunk and offset the call was made from
Task ID	%i	The address of the task making the call
Target Name	%n	The name of the item the function call is accessing
Options	%o	Any additional options required by the function
Proc Name	%p	The name of the process making the call
Res.	%r	The result returned by the function
Segment Name	%s	The name of the module making the call
Time	%t	The time the function was called at
Count	%u	The sequence number of this event

Each field may appear only once in a format string, but need not appear at all. Several of the fields are really only of use to programmers and can be ignored by most people.

The % characters listed are used in the format string to represent each field. Between the % and the field identifier, a number may appear; this defines the number of characters available to display that field. If no number is given, then a sensible default is used instead.

Here are some example formats:

```
 "%21p %5a %39n %4o %r"           -- Classic SnoopDos 1.7 format
 "%16p %r %6a %6o %80n"           -- Maximizes space for target name
 "%u %7i %p %h %s %7a %39n %o %r" -- For programmers with big screens
```

Normally, SnoopDos will use the currently defined format string when logging information to disk. However, you can change this by supplying a special LogFormat string instead.

If your format string includes some unrecognised or duplicate fields, SnoopDos will silently remove them and simply use what remains.

See also: Command Index Format editor LogFormat

1.72 formatwindowpos

Command: FORMATWINDOWPOS=<left>,<top>

Example: FORMATWINDOWPOS=0,20

This command is used to change the position of the format editor window on the current screen. If the format window is open, then it is moved

immediately, else it will appear in the new position the next time it
is opened. <left> and <top> are the pixel co-ordinates of the upper
left corner of the window.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index   Format editor

1.73 functions

Command: FUNCTIONS=ALL | NONE | ALLDOS | NODOS | ALLSYSTEM | NOSYSTEM

Example: FUNCTIONS=ALLDOS

This command provides a convenient way for an ARexx script to quickly
enable or disable monitoring of a group of functions.

The ALL and NONE option apply to the entire group of functions that
SnoopDos can monitor. ALLDOS and NODOS affect only those functions
listed under "DOS Functions" in the function window. ALLSYSTEM and
NOSYSTEM affect only those functions listed under "System Functions"
in the function window.

See also: Command Index   Function window

1.74 functionwindowpos

Command: FUNCTIONWINDOWPOS=<left>,<top>

Example: FUNCTIONWINDOWPOS=0,20

This command is used to change the position of the function window on
the current screen. If the function window is open, then it is moved
immediately, else it will appear in the new position the next time it
is opened. <left> and <top> are the pixel co-ordinates of the upper
left corner of the window.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index   Function window

1.75 getvar

Command: GETVAR
         NOGETVAR

This command enables or disables monitoring of the GetVar() and FindVar()
dos.library functions. It is also available as a gadget in the Function

window.

AmigaDOS has three types of environment variables: Local, Global and Alias. These functions allow access to all three types. Local variables are associated directly with a single CLI or Shell, and can't be accessed by any other part of the system. Global variables reside in the ENV: directory and can be accessed by all programs. Alias variables are manipulated using the AmigaDOS Alias command, and aren't usually accessed by application programs.

The Options output field displays which of the three types of variable a program is asking for. If the variable type is followed by an * (for example, "Local*") then this indicates that the program is expecting a binary variable which may contain non-printable characters and which was probably created by the program itself, rather than the AmigaDOS SetEnv command.

GetVar() and FindVar() are very similar in operation. GetVar() is typically used by application programs that are only interested in the current value of a variable, while FindVar() is used by Commodore commands that want to directly alter a variable's current value.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index SetVar

1.76 gotoline

Command: GOTOLINE <n>

Example: GOTOLINE 100

This command is used to reposition the buffer display in the main window at a particular line. The line numbers correspond to those displayed in the Count output field.

A quick way to move to the top or bottom of the buffer is to give the command GOTOLINE 0 or GOTOLINE 99999.

See also: Command Index ScrollDown ScrollUp

1.77 help

Command: HELP <topic>

Example: HELP MainWindow

You can call up help from within SnoopDos itself via the Help menu option on the Project menu, by pressing the Help key while any menu option is highlighted, or by pressing the Help key inside any SnoopDos window.

The HELP command is mainly of use from the CLI. If you give it with no parameters, then it will display a list of all the currently recognised SnoopDos commands. If you give a topic, then help on that topic will be loaded into AmigaGuide for you to read.

If in doubt, HELP MAIN will always bring you to the main Contents page, from where you can find any information you require.

See also: Command Index

1.78 hide

Command: HIDE

This command closes all the SnoopDos windows, but leaves SnoopDos monitoring system activity in the background. It is available as a gadget in the Main window, and as a menu option on the Project menu. You can also hide SnoopDos by clicking on the Close gadget in the main window. If the current HideMethod is set to None, then this option is not available.

When SnoopDos is hidden, the main window can be re-opened by pressing the currently defined HotKey . Depending on the hide method, you may also be able to open the SnoopDos AppIcon, or select the Show SnoopDos menu item on the Workbench Tools menu.

See also: Command Index CX_Popup HideMethod Show

1.79 hidegadgets

Command: HIDEGADGETS

This command disables the display of the panel of button gadgets in the Main window. It is also available as a menu option on the Windows menu.

Removing the gadgets allows more information about events to be displayed in the SnoopDos window. Even though the gadgets are no longer visible, you can still use the gadget keyboard shortcuts to activate the various functions.

To maximise the amount of room available for displaying event output, you might also like to turn off the Status line gadget.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index HideStatus ShowGadgets
ShowStatus TextSpacing

1.80 hidemethod

Command: HIDEMETHOD=NONE | INVISIBLE | ICONIFY | TOOLSMENU

Example: HIDEMETHOD=ICONIFY

This command controls the mechanism SnoopDos uses to run in the background when you select the Hide command. It is also available as a gadget in the Settings window.

Setting the hide method to NONE stops SnoopDos from running in the background at all. The Hide menu and gadget options will be disabled, and when you click on the main window's Close gadget, SnoopDos will exit completely, rather than just hiding itself.

The INVISIBLE hide method completely removes all traces of SnoopDos from the screen. The only way to re-activate it is to use the currently defined commodity HotKey , use the Show Interface option in the commodities exchange, or send a Show command to SnoopDos (double-clicking on the SnoopDos program icon will do this).

The ICONIFY hide method is similar to INVISIBLE, but creates an AppIcon on the Workbench desktop whenever SnoopDos is hidden. SnoopDos can then be re-activated by double-clicking on the AppIcon. The image used for the AppIcon will be the same as the SnoopDos program icon if possible; if no icon was found, a simple built-in image is used instead. The icon position will be automatically selected by Workbench; if you prefer, you can select a fixed position using the IconPos command.

The TOOLSMENU hide method creates an entry on the Workbench Tools menu entitled "Show SnoopDos". Selecting this menu item will automatically re-open the SnoopDos window.

Note that the SnoopDos hot key can be used to re-activate the main window at all times, even if SnoopDos is not currently hidden.

See also: Command Index Hide HotKey Show

1.81 hidestatus

Command: HIDESTATUS

This command tells SnoopDos to disable the display of the Status line gadget in the Main window. It is also available as a menu option on the Windows menu.

The status line is not strictly necessary, but it provides a useful "at-a-glance" summary of whether or not a log file is currently open, and whether SnoopDos is currently paused or disabled .

Some people prefer to sacrifice the status line to allow more lines of event output to be displayed in the main window. If you are doing this, you may like to also hide the button gadgets, which will allow the maximum number of lines of text to be displayed.

SnoopDos automatically writes this command to the settings file when

you save settings.

See also: Command Index HideGadgets ShowGadgets
ShowStatus TextSpacing

1.82 hotkey

Command: HOTKEY "key description"

Example: HOTKEY "ctrl alt d"

This command allows you to select what key sequence is used to open the SnoopDos main window when it is hidden. This corresponds to the Hotkey gadget in the Settings window. It is a synonym for the CX_PopKey command.

Any standard commodities key description can be given. You can always tell which key sequence is currently in use by looking at the titlebar of the Main window. If you enter an incorrect description, then the titlebar will display "<invalid>".

When you have defined a hotkey, SnoopDos can be called up at any time by pressing that key sequence. If the main window was already open, it will be brought to the front and activated. If it was open, but on a different screen, it will be re-opened on the current screen (as selected using the ScreenType command).

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index CX_PopKey CX_Popup CX_Priority

1.83 iconpos

Command: ICONPOS=<left>,<top>

Example: ICONPOS=500,300

This command defines the position of the AppIcon created by SnoopDos when it is running in an Iconified state. Normally, the AppIcon will be automatically positioned in an appropriate position on the screen, but you may sometimes want to override this with your own preferred co-ordinates. Setting the position to -1,-1 allows Workbench to choose the best position once again.

Note that there is no way to set this option from within SnoopDos; you must change it by editing the SnoopDos default settings file directly, or by adding it as a tooltype to the SnoopDos icon.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index HideMethod

1.84 ignoreshell

Command: IGNORESHELL
NOIGNORESHELL

This command enables or disables SnoopDos's Ignore Workbench/Shell setting. This setting can also be controlled via a gadget in the Function window.

When this is enabled, SnoopDos will ignore all system calls made by Workbench, or by any shell processes or background CLI's that don't currently have a command loaded. This can reduce the amount of output substantially.

If you have the MonitorROMCalls option enabled, then most Workbench and Shell activity will be automatically ignored anyway. In that case, you mays prefer to leave this option disabled, since it will speed up SnoopDos's monitoring a little.

Note: if you enable MonitorROMCalls and disable IgnoreShell, then you can see which tooltypes Workbench looks for by default whenever it starts a program from an icon.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index MatchName MonitorROMCalls

1.85 language

Command: LANGUAGE "languagename"

Example: LANGUAGE "english"

This command is used to define the language used by SnoopDos to display all menus, windows, and requesters. SnoopDos has built-in support for English, but can also support other languages via external language catalog files.

At this time, SnoopDos only supports English. However, I expect to release localised versions over the next few months. If you track down the SnoopDos source code archive on AmiNet, you will find it contains a catalog description file that will enable you to create your own language catalog. If you would like to create a catalog for your native language, please email me first in case someone has already performed the translation.

Unlike most other commands, the LANGUAGE command only makes sense when specified as either an icon ToolType or a CLI command line option. This is because SnoopDos needs to read the language file very early on during its startup initialisation, in order to allow it to display error

messages in the correct language.

Normally, you should never need to use this command -- SnoopDos will automatically try and use one of the languages defined in the system Locale preferences, only falling back to English if an appropriate catalog file can't be located.

However, if you are developing a new catalog file for SnoopDos, then this command can be useful since it allows you to tell SnoopDos to use your new catalog file during testing.

SnoopDos will look for the language you specify in two separate places. The first is "PROGDIR:Catalogs/<language>/SnoopDos.catalog" and the second is "LOCALE:Catalogs/<language>/SnoopDos.catalog". You can override these defaults by including a full path name to the directory containing the SnoopDos.catalog file.

For example, setting LANGUAGE="ram:" will look for the catalog file "ram:SnoopDos.catalog". If you do this, you will need to temporarily alter the name given in the ##language line of your translation (.CT) file to read "##language ram:" as well, or locale.library will not recognise the generated catalog as valid. Also remember to AVAIL FLUSH after quitting SnoopDos, since otherwise locale.library will keep a copy of the first catalog file loaded into memory and ignore the new version on disk.

See also: Command Index

1.86 leftaligned

Command: LEFTALIGNED

This command controls how the Target name output field is displayed in the main window. It is also available as a sub option to the Target Name menu option on the Windows menu.

Left aligned is the normal state of the Target Name field; in this mode, it behaves just like all the other output event columns. See the discussion under RightAligned for more information.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index RightAligned ShowFullPaths

1.87 loaddefsettings

Command: LOADDEFSETTINGS

This command reads in the default settings file, as defined by the Settings command. The Last Saved option on the Settings menu will usually do the same thing.

If no file has been explicitly defined, then the file that was loaded
during startup will be reloaded. If no such file could be found, then
it will attempt to load ENVARC:SnoopDos.prefs.

See also: Command Index   LoadSettings   SaveDefSettings
           SaveSettings   Settings

1.88 loadseg

Command: LOADSEG
         NOLOADSEG

This command enables or disables monitoring of the LoadSeg() and
NewLoadSeg() dos.library functions. It is also available as a gadget
in the Function window.

These functions are called to load an executable module from disk into
memory, where it can be executed. You are mostly likely to see these
functions being called by the ramlib process, which is responsible
for loading in libraries and devices from disk.

You may also see these being used to load in fonts and keymaps. Sometimes,
an application will call LoadSeg() directly to load a sub-component of the
program which has not yet been needed. If you see LoadSeg() being called
with no apparent name, this indicates that an application is loading an
overlay -- a part of the application which has remained on disk until
it was needed, to help conserve memory.

Functionally, both LoadSeg() and NewLoadSeg() are identical. The main
difference is that NewLoadSeg() was not introduced until Kickstart 2.04,
and may in the future gain some additional options that are not available
with LoadSeg().

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index

1.89 loadsettings

Command: LOADSETTINGS "settingsfile"
         LOADSETTINGS "con:////SnoopDos Command Window/CLOSE"

Example: LOADSETTINGS "ram:SnoopDos.prefs"

This command reads in a SnoopDos settings file. It is also available
as a menu item on the Settings menu.

A settings file is any ascii text file which contains valid SnoopDos
commands, one per line. The file must contain the following text on
the first line to be recognised as a settings file:

```
    <SnoopDos Settings>

Anything between a semicolon and the end of a line will be treated
as a comment and ignored.

Settings files are usually used to store a particular combination of
SnoopDos settings that define what functions will be monitored etc.
However, they can also contain other commands which perform actions
like opening or closing windows or log files.

If the filename you specify is an interactive file (i.e. a CON: window)
then LoadSettings behaves slightly differently. It prints a command prompt
in the window, and allows you to type in commands directly at the keyboard.
This lets you try out any SnoopDos command interactively and see the
effect it has. Note however that as long as the console window is open,
other SnoopDos operations will be suspended.

See also:  Command Index    LoadDefSettings    SaveDefSettings
           SaveSettings     Settings
```

1.90 lock

```
Command: LOCK
         NOLOCK

This command enables or disables monitoring of the Lock() dos.library
function. It is also available as a gadget in the  Function  window.

A program usually calls Lock() to check whether or not a file actually
exists. It can also use Lock() to locate a directory so that it can access
the files within that directory.

A file or directory can be locked for Read access, where several programs
are free to access it at the same time, or Write access, where the program
requires exclusive control with no interruptions. The  Options  output
field displays which access mode a program asked for.

You may occasionally see an access mode of "Read???" -- this indicates that
a program has forgotten to specify which access mode was required. Most
AmigaDOS disk devices will assume that Read was intended, but some may be
confused, causing the program to work incorrectly on those devices.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also:  Command Index    ChangeDir    Open
```

1.91 lockscreen

```
Command: LOCKSCREEN
         NOLOCKSCREEN
```

This command enables or disables monitoring of the LockPubScreen() intuition.library function. It is also available as a gadget in the Function window.

Many modern programs support Intuition public screens -- screens similar to the Workbench screen, but possibly with a different resolution and number of colours. Monitoring this function allows you to see if a program expects to find a public screen with a particular name.

If it finds such a screen, it may choose to open there rather than on the Workbench. You can use one of the many public screen manager utilities to create a screen with a particular name. In fact, SnoopDos itself has the ability to open on a named public screen -- for more information, see the ScreenType command.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index

1.92 logformat

Command: LOGFORMAT "formatstring"
LOGFORMAT NONE

Example: LOGFORMAT "%t %p %a %n %o %r"

This command is similar to the Format command, but instead of controlling the format of the events displayed in the Main window, it controls the format of events written to the log file. It is also available as a gadget in the Settings window.

Normally, you won't need to use this. If no log format has been defined, then SnoopDos will use the current main window format instead. However, occasionally you may prefer to use a custom format. For example, you may have a large Workbench screen which can display 130 columns of text in the main window, but you would like your log files to fit on an 80 column display.

The format string specified is identical to that used for the Format command. Usually, the quickest way to design a new log format is to set the main window to 79 columns using the Change window width menu option and then use the Format editor to choose what items you want displayed.

Once you are happy with the format in the main window, you can quickly copy it to the log format by using the Copy gadget in the Settings window.

The special keyword NONE is used to tell SnoopDos to stop using a custom log format and just use whatever format is installed in the main window instead. This is equivalent to just giving an empty format string, but is often more convenient to use from within an ARexx script.

Note that when SnoopDos is writing to a log file, it automatically leaves the first character on each line blank (see OpenLog for more details).

As a result, a log format that's 79 characters wide produces lines of 80 characters in the log file.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index Format Format editor OpenLog

1.93 logmode

Command: LOGMODE=PROMPT | APPEND | OVERWRITE | SERIALPORT

Example: LOGMODE=PROMPT

This command determines the precise operation of the Open Log gadget in the main window. It is also available as a gadget in the Settings window.

When the log mode is set to PROMPT (the most common choice), then the main window gadget will be titled "Open Log...", and selecting it will produce an ASL file requester allowing a log file to be chosen. If the chosen file already exists, SnoopDos will display a requester allowing you to overwrite or append to it.

When a file requester is displayed, you are not limited to entering only filenames -- you can also enter device names such as PRT:, CON:////, or AUX:.

When the log mode is set to APPEND, the main window gadget will be titled "Append Log" and selecting it will automatically open the log file defined using the LogName command in append mode. This is convenient if you like all logging information to be captured into a single file, and prefer not to have to deal with the ASL file requester.

The OVERWRITE log mode is similar to APPEND, but the main window gadget now reads "Start Log". When that gadget is selected, the filename defined with the LogName command is overwritten with the new log information. This is most appropriate if you frequently log to a device, such as the printer (PRT:) or perhaps a console window -- you can set the log name accordingly and then easily turn on or off logging to that device with a single click of the mouse.

If you are logging to a console window, note that the old SnoopDos 1.x control keys still operate. CTRL-D and CTRL-E can be used to disable and enable logging, and in addition, CTRL-F will bring the main SnoopDos window back to the foreground again. CTRL-C will quit SnoopDos entirely, so be careful not to press it by mistake.

The SERIAL PORT log mode will send the log output directly to a serial debugging terminal connected to the internal serial port. This is mainly intended for programmers. It is particularly useful if you use Carolyn Scheppner's useful Sushi utility to redirect debugging output to a window or file, since it allows SnoopDos output to be interleaved with output from other tools like Enforcer and Mungwall. When this option is chosen, the main window gadget is titled "Serial Log".

Note that even when the log mode is APPEND, OVERWRITE or SERIAL PORT,
you can still open a log file via the ASL file requester by choosing the
 Open Log menu item on the Project menu.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index   AppendLog   OpenLog   OpenSerialLog

1.94 logname

Command: LOGNAME "logfile"

Example: LOGNAME "PRT:"

This command defines the default log filename to be used when the current
 LogMode is set to either APPEND or OVERWRITE. It is also available as a
gadget in the Settings window.

The APPEND and OVERWRITE log modes allow the ASL file requester to be
bypassed when opening a log file; this is convenient if you frequently log
output to a single file or device.

If you are using a log mode of PROMPT, where an ASL file requester is
displayed to let you choose a log file, then the first default filename
supplied to the ASL requester will match the currently defined LogName.
This is useful if you usually log to the same file, but prefer to be able
to easily change to a different filename if you want to.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index   LogMode

1.95 mainwindowpos

Command: MAINWINDOWPOS=<left>,<top>

Example: MAINWINDOWPOS=0,20

This command is used to change the position of the main window on
the current screen. If the function window is open, then it is moved
immediately, else it will appear in the new position the next time it
is opened. <left> and <top> are the pixel co-ordinates of the upper
left corner of the window.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index   Main window

1.96 mainwindowsize

```
Command: MAINWINDOWSIZE=<width>,<height>

Example: MAINWINDOWSIZE=640,150
```

This command is used to change the current dimensions of the main window. If the main window is open, then it is resized immediately, else it will appear with the new size the next time it is opened. <width> and <height> are the new dimensions of the window in pixels.

In addition to using this command, or manually resizing the window with the mouse, you can also use the WindowWidth command to define the width of the window in text columns rather than pixels.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index Main window

1.97 makedir

```
Command: MAKEDIR
         NOMAKEDIR
```

This command enables or disables monitoring of the CreateDir() dos.library function. It is also available as a gadget in the Function window. (The command name has been chosen to match the CLI command of the same name, which is more familiar to most users than CreateDir).

Programs call this function to create a new directory on disk. This is most commonly done during installation, where an application creates a special directory to hold data or project files.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index

1.98 makelink

```
Command: MAKELINK
         NOMAKELINK
```

This command enables or disables monitoring of the MakeLink() dos.library function. It is also available as a gadget in the Function window.

This function is called to create a file link on disk, which provides a new name for an existing file, while retaining the original name as well. Unlike the Copy command, this approach does not require a second copy of the file to be stored, which saves disk space.

There are two types of Link: Hard and Soft. Hard links can only reside on the same partition as the original file. Once created, they become indistinguishable from the original; if the original file is deleted, then the hard link automatically takes its place.

Soft links can be created on any disk or partition, not just that which contained the original file. However, if the original file is deleted, then the soft link becomes useless.

The Target name output field will show the link information in the form "Newlink --> original_file". The Options output field will show "Soft" or "Hard" according to the link type.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index

1.99 matchname

Command: MATCHNAME "pattern"

Example: MATCHNAME "~(Yak|TurboText)"

This command lets you define what programs SnoopDos should or should not monitor. It is also available as a gadget in the Function window.

The pattern you give is a standard case-insensitive AmigaDOS pattern. This will be compared with the name of each process that calls any of the functions currently being monitored; only those processes with names that match the pattern will be displayed.

Typically, you will use the MatchName to ensure that certain programs are automatically ignored by SnoopDos. For example, the TurboText text editor has the annoying feature of calling CurrentDir() every time you type a key. Similarly, the Yak commodity calls OpenDevice() every time you type a key when you have key-click enabled. You can use the MatchName pattern shown in the example above to prevent the event buffer from being quickly filled with useless events.

In general, you can make SnoopDos ignore several different programs by using a pattern of the form "~(prog1|prog2|prog3|prog4|...)", repeating for as many program names as required.

Another less common use for MatchName is to monitor only those events generated by one particular program. For example, a MatchName of "Multiview" would ignore any function calls by programs other than Multiview. A MatchName of "Multiview|AmigaGuide" would monitor calls by either Multiview or AmigaGuide. If a program you're trying to monitor may be run from a Shell rather than Workbench, you may wish to prefix its name with "#?" in the pattern, to ensure that any preceding pathname is ignored. For example: "#?Multiview|#?AmigaGuide".

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index

1.100 monitorpackets

Command: MONITORPACKETS
NOMONITORPACKETS

This command enables or disables SnoopDos's MonitorPackets setting. This setting can also be controlled via a gadget in the Function window.

Some programs, particularly those written using GNU C and linked with ixemul.library, bypass the normal dos.library functions and communicate directly with DOS devices using packet I/O.

Normally, SnoopDos can't detect such activity. However, if you enable this option, then SnoopDos will monitor the PutMsg() exec.library function. This allows packets to be detected, and displayed in the SnoopDos window just like normal function calls.

When a packet is spotted that corresponds to one of the currently monitored DOS functions, the Action output field will display the packet type (e.g. Open, Lock, MakeDir, etc.) but it will be prefixed by '*' to indicate that direct packet i/o was involved. Thus, in the examples given, you will see *Open, *Lock, *MakeDir, etc.

SnoopDos can monitor packet versions of the following functions:

Delete Lock MakeDir MakeLink Open Rename

Enabling MonitorPackets can potentially slow down your system, since PutMsg() is one of the most frequently called system functions. Thus, it should only be enabled if you encounter a program that appears to be opening files but which doesn't show up under normal monitoring.

Packet operations carried out by AmigaDOS on behalf of an application will not usually be reported by SnoopDos -- this is because there is no need; the corresponding AmigaDOS function will be reported instead. However, if you enable MonitorROMCalls at the same time as MonitorPackets, then you may find some events being listed twice: once as a DOS function, and once as a packet operation.

Sometimes SnoopDos is unable to determine the result code of a packet operation. This can happen if you stop monitoring packets while that operation is incomplete, or occasionally when there is heavy packet activity (SnoopDos can keep track of about 5 outstanding packets at any one time). When this happens, the Result field will display "Missed" to indicate it has given up.

If you need more information than that provided by MonitorPackets, you can use the PacketDebugger option. This lets you monitor every single packet sent by an application to a DOS device, though since it generates vast quantities of raw output, it is really only intended for device driver programmers.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index MonitorROMCalls PacketDebugger

1.101 monitorromcalls

Command: MONITORROMCALLS
NOMONITORROMCALLS

This command enables or disables SnoopDos's MonitorROMCalls setting. This
setting can also be controlled via a gadget in the Function window.

When a program calls a system function, that function may in turn trigger
internal calls to other system functions. For example, calling OpenLibrary()
may result in calls being made to FindResident() and Open() as well.
Usually, these additional calls just add to the clutter and aren't needed,
so by default SnoopDos doesn't display any calls made by the operating
system (i.e. from within the ROM).

However, sometimes it can be useful to see these, especially if all else
fails. Turning on this option tells SnoopDos to go ahead and print function
calls made from within the ROM, along with all the calls made by user
applications.

Be warned though: this can often double the amount of output produced! If
you enable MonitorROMCalls, then you will probably want to ensure that
IgnoreShell is also enabled, to make SnoopDos ignore output generated by
Workbench or the Shell.

One thing to be aware of is that SnoopDos can only check if a function call
has been from ROM by looking at the return address on the stack. If another
program has patched a function call after SnoopDos, then SnoopDos will see
all calls to the function as coming from that program rather than from the
original caller, and so won't ignore them. One such example is the
OpenLibrary() call, which is patched by the CodeProbe debugger.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index MonitorPackets PacketDebugger

1.102 onlyshowfails

Command: ONLYSHOWFAILS
NOONLYSHOWFAILS

This command enables or disables SnoopDos's OnlyShowFails setting. This
setting can also be controlled via a gadget in the Function window.

Depending on the functions you have enabled, SnoopDos can provide vast
quantities of output. Enabling OnlyShowFails provides an easy way to

reduce the amount of output you see, while still retaining information
that is likely to be of interest.

This is achieved, as the name implies, by only showing those functions
which failed -- any function call which is successful is not listed.

You need to be a little wary when using this option, since it is possible
to be misled if you are not careful. Often, a program may search for files
or other information in several places. If the first two attempts fail, but
the third succeeds, you will only see information about the two failed
attempts and may incorrectly surmise that the program is in trouble.

Thus, while OnlyShowFails is useful for identifying potential problems
early on (and saving valuable space in the event buffer), it's best to
turn it off when you are diagnosing an actual program failure.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index

1.103 open

Command: OPEN
NOOPEN

This command enables or disables monitoring of the Open() dos.library
function. It is also available as a gadget in the Function window.

If you only choose to monitor one function, the Open() function will
usually provide the most useful information, since missing configuration
files are one of the most common reasons an application will not run.

Note however that an application may do other checks to determine if a file
actually exists, before trying to open it. The most common method is to use
the Lock function to try and locate the file.

Another alternative, and one which SnoopDos can't easily monitor, is to
scan a directory looking at all files in that directory. If you suspect
this is happening, you can use SnoopDos's PacketDebugger option to watch
out for this, but be prepared to sift through a lot of superfluous output.)

Files may be opened in one of three modes: Read, Write or Modify. Read
implies that the file will only be read, not modified. Write implies that
a new file should be created, overwriting any file that already exists.
Modify is used to write to an already existing file (or to create a new
file if none exists). The Options output field indicates the mode used
to open the file.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index Lock

1.104 opendevice

```
Command: OPENDEVICE
         NOOPENDEVICE
```

This command enables or disables monitoring of the OpenDevice() exec.library function. It is also available as a gadget in the Function window.

Almost every non-trivial program needs to use devices, and this is the function called to provide access to each device.

Monitoring of OpenDevice() can result in quite a lot of output, most of which isn't very useful. If you're only interested in the devices that a program needs to open but can't, they will show up under the output from LoadSeg since the Amiga automatically tries to load any device it can't find in memory from the DEVS: directory.

When this function is being monitored, the Options output field will show the unit number of the device the program is trying to open.

Note to OS experts: All OpenDevice() calls made by CON processes are ignored by SnoopDos. This is because CON frequently opens timer.device, sometimes many times a second, and reporting all these events would quickly fill up the event buffer with useless output.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index OpenLibrary

1.105 openfont

```
Command: OPENFONT
         NOOPENFONT
```

This command enables or disables monitoring of the OpenFont() graphics.library function. It is also available as a gadget in the Function window.

OpenFont() is called to locate a particular font for display on the screen. The Options output field shows the size of the font that's being requested.

Usually, there is no need to monitor this function -- if a program can't locate a font in memory, AmigaDOS will automatically try and load it from the FONTS: directory on disk, and it will be displayed under LoadSeg . If you want to see every font used by a program, however, then this is the function to use.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index

1.106 openformat

Command: OPENFORMAT

This command opens the Format editor window. You can also open the format editor by selecting the Show format menu item on the Windows menu, or by double-clicking on the header line in the Main window.

Yet another way to open the format editor is using the "Edit..." gadget in the Setup window.

See also: Command Index CloseFormat FormatWindowPos

1.107 openfunction

Command: OPENFUNCTION

This command opens the Function window. You can also open the function window by selecting the Show functions item on the Windows menu, or by selecting the "Functions..." gadget in the main window.

If your Amiga only has a 68000, you may find that the function window takes a long time to open the first time you access it. This is due to the large number of gadgets it contains. SnoopDos only needs to create the gadgets the first time the window is opened; on the second and subsequent openings, you should find it much quicker.

See also: Command Index CloseFunction FunctionWindowPos

1.108 openlibrary

Command: OPENLIBRARY
NOOPENLIBRARY

This command enables or disables monitoring of the OpenLibrary() exec.library function. It is also available as a gadget in the Function window.

It's impossible to write a useful program on the Amiga that doesn't call OpenLibrary() at least once. It provides access to the Amiga's multitude of shared libraries.

In general, a program will fail to run if it can't find a particular library. However, normally such failed attempts will show up as LoadSeg events, because if AmigaDOS can't find a library in memory, it will try and load it from the LIBS: directory instead.

A program won't necessarily stop working if it can't find a particular library; it may just disable some of its features. For example, SnoopDos itself can run without the Commodities or ASL libraries being present, but if they're not around, you won't be able to Hide the main window or call up file and font requesters.

The Options output field gives the version number of the library being opened. This is useful information, because if a program can't open a library, you can then see the minimum version of the library you must install.

Note that SnoopDos automatically ignores all attempts to open dos.library. This is necessary to avoid a few subtle problems that can otherwise occur. Since dos.library must exist to even load the program in the first place, you won't miss out on anything important by not seeing these events.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index OpenDevice

1.109 openlog

Command: OPENLOG "filename"

Example: OPENLOG "ram:snoopdos.log"

This command opens a new log file. Any new output displayed in the main SnoopDos window will be written to this file. If the file already exists, it will be overwritten. You can also open a new log file from within SnoopDos by choosing the Open log menu option on the Project menu, or by using the Open Log button in the Main window.

When SnoopDos displays events in the main window, it can easily handle several different events taking place simultaneously. For example, one program might call RunCommand() to start a command executing, and that command may itself open several files. SnoopDos can display information about those open actions even though the previous RunCommand() call has not yet completed. When the RunCommand() does complete, SnoopDos goes back and fills in the result code accordingly.

When outputting events to a log file, it's not so easy to go back and fill in the result field after several additional events have occurred. For example, the log file may actually be a printer or some other device that doesn't allow backtracking.

To avoid this problem, SnoopDos behaves slightly differently when printing events to a log file. It will usually wait until an event is fully completed before it tries to output it to the file. However, if a new event arrives before the current event is complete, SnoopDos will output as much of the current event as possible. To show that the event is incomplete, it outputs a '/' character in the first column of the line.

Later on, when that event finally completes, SnoopDos will output the entire line a second time, but this time with the result filled in. To indicate that this is the completion of an earlier event, the first column will contain a '\', to match the '/' used earlier.

If you anticipate this happening a lot, it's a good idea to include the Count output field in your log format. This will include a sequence

number on each line, and you can then easily match each incomplete event
with the final completed version since both lines of output will have
the same number.

See also: Command Index AddLog AppendLog CloseLog
FlushLog LogFormat OpenSerialLog

1.110 openresource

Command: OPENRESOURCE
NOOPENRESOURCE

This command enables or disables monitoring of the OpenResource()
exec.library function. It is also available as a gadget in the
Function window.

Resources are typically used to provide very low-level access to standard
Amiga hardware, such as the gameport, CIA chips, battery-backed clock,
floppy disk, etc. You can monitor OpenResource() to see if a program is
trying to access the hardware directly in a system-legal manner.

Of course, programs that break the rules and don't allocate hardware
properly before using it won't show up (but your system will probably
have crashed by then anyway).

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index

1.111 openseriallog

Command: OPENSERIALLOG

This command opens a new log file. Unlike normal log files, the output from
this will be directed to a debugging terminal connected to the internal
serial port. You can configure the "Open Log" button in the Main window
to perform this command by setting the LogMode to SERIALPORT.

This is mainly intended for programmers. It is particularly useful if you
use Carolyn Scheppner's useful Sushi utility to redirect debugging output
to a window or file, since it allows SnoopDos output to be interleaved with
output from other tools like Enforcer and Mungwall.

See the OpenLog documentation for more information about how SnoopDos
outputs information to a log file.

See also: Command Index AddLog AppendLog CloseLog
FlushLog LogFormat OpenLog

1.112 opensetup

Command: OPENSETUP

This command opens the Setup window. You can also open the setup window by selecting the Show setup menu item on the Windows menu, or by selecting the "Setup..." gadget in the main window.

See also: Command Index CloseSetup SetupWindowPos

1.113 packetdebugger

Command: PACKETDEBUGGER
NOPACKETDEBUGGER

This command enables or disables SnoopDos's Packet debugger. This setting can also be controlled via a gadget in the Function window.

The packet debugger provides a way to snoop on packets sent to any mounted filesystem device. This is mainly of use to device driver writers, but can also occasionally be useful if you want to monitor DOS operations not directly supported by SnoopDos, such as Examine()/ExNext().

When the packet debugger is first enabled, SnoopDos makes a list of currently mounted devices. Only packets sent to those devices are monitored.

If a new device is mounted while the debugger is running, SnoopDos will usually automatically add it to this list. However, if the device has been mounted using a third-party mount command, you may need to disable and re-enable the packet debugger to get SnoopDos to recognise packets destined for the new device.

The first time you enable the packet debugger, you may find that it doesn't seem to have much effect. This is because almost all packets to DOS devices are generated by dos.library, and dos.library is in ROM. By default, SnoopDos ignores any packet activity generated by ROM calls. Enable the MonitorROMCalls setting to change this.

SnoopDos can identify all of Commodore's officially defined packets, as of Kickstart 3.0, and can also recognise a few third party packets, as listed in Ralph Babel's excellent book, The Amiga Guru Guide. The name of each packet is displayed in the Action output field. Each name is shown in capitals and is prefixed with '#', to help distinguish the output from normal monitored functions.

If SnoopDos doesn't recognise a packet, then the packet type will be shown in the form "#(nnn)" where nnn is the packet type code in decimal.

Most packets take one or more parameters (dp_Arg1, dp_Arg2, etc). These parameters are listed in the Target name field as 32-bit hex numbers. The interpretation of each number depends on the packet; see Ralph's book for a comprehensive explanation.

The Options field shows the name of the device to which the packet is being sent.

Finally, the Result field shows the return value from the packet. Most packets return a single 32-bit value for success, or a fail code and a secondary error code for failure. SnoopDos will display "OK" or "Fail" for each packet, and will also show the actual value returned.

For successful packets, this value matches dp_Res1; for failed packets, it matches dp_Res2. (SnoopDos knows that some packets return -1 for failure, while others return 0). Some packets return two values -- in this case, SnoopDos will display both values.

Since the default width of the Result field is only four characters, you may need to widen it to view this additional information. This is easily done by simply dragging the right edge of the Result event heading in the main window with the mouse, or by using the Format editor .

Sometimes SnoopDos is unable to determine the result code of a packet operation. This can happen if you disable the packet debugger while a packet operation is still in progress, or occasionally if there is heavy packet activity (SnoopDos can remember up to 5 outstanding packets at a time). When this happens, the Result field will displayed "Missed" to indicate SnoopDos has given up waiting for a result.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index MonitorPackets MonitorROMCalls

1.114 patchramlib

Command: PATCHRAMLIB
NOPATCHRAMLIB

This command controls whether or not SnoopDos should apply a patch to the ramlib background process (ramlib is responsible for loading libraries and devices from disk, whenever they are needed by a program). This command is slightly unusual in that it must be given on the CLI command line or as an icon tooltype the first time SnoopDos is started, or it will have no effect. If it is not present, the default action is PATCHRAMLIB.

Normally, you should never need to use this command. However, if you like to know exactly what's happening to your Amiga, you may want to read on.

The reason for this command is because the ramlib process contains a bug in all versions of Kickstart from 2.0 through 3.1. This bug does not show up during normal use, but when SnoopDos is running, the bug can cause a crash during heavy system activity.

For those interested, the bug is that ramlib uses SIGB_SINGLE as the signal bit for its message port, which means that system calls made by ramlib cannot safely use this signal for any purpose -- after all, you never know when a new message might arrive at ramlib's message port, causing a SIGB_SINGLE signal. The most critical use of SIGB_SINGLE in the operating

system is to synchronise sempahore operations when two tasks are trying
to access a single piece of data simultaneously.

The reason this doesn't cause a problem in normal use is because ramlib
calls very few system functions that depend on SIGB_SINGLE, and those
that do (mainly LoadSeg, which locks the device list) usually don't do
anything too terrible if it should fail.

SnoopDos makes heavy use of semaphores in its patches. This means that any
calls from ramlib that it monitors are potentially at risk. Even then,
problems only occur when several programs all ask ramlib to load a library
or device at the same time. However, when a problem occurs, it causes an
instant crash as SnoopDos metaphorically gets its knickers in a twist.

To avoid this problem, SnoopDos patches the ramlib process to use a
different signal bit (SIGBREAKB_CTRL_E in fact). This patching is performed
in a safe way, so that if a future SetPatch ever corrects the bug, it won't
cause any conflicts. If you have the SnoopDos source code from Aminet, you
can review the patch code in patches.c:InitRamLibPatch().

Despite the efforts to make the patch safe and upwards compatible, it's
possible that it may cause problems with some future revision of the
operating system. It's also possible that you may not like the idea of
a third party utility altering an internal system structure like this.

Thus, NOPATCHRAMLIB provides a way to stop SnoopDos from making this change.
A side effect of using NOPATCHRAMLIB is that SnoopDos will automatically
ignore all calls made by the ramlib process. This obscures some useful
information about what libraries and devices are being loaded loaded, but
for the most part, you can simply turn on monitoring of OpenLibrary and
 OpenDevice and get the same information.

Note that any calls made to the FindResident function by ramlib
are automatically ignored by SnoopDos; this helps keep the amount of
information displayed in the window to manageable levels.

See also: Command Index   StackLimit

1.115 pause

Command: PAUSE

This command is used to temporarily pause any programs calling functions
monitored by SnoopDos, so that you have a chance to see exactly what each
program is trying to do. It is also available as an item on the Project
menu, and as a gadget in the Main window. You can use the TAB key to
unconditionally select Pause (this is sometimes more convenient than
pressing the Pause gadget's keyboard shortcut, since that toggles Pause
on and off).

Pause is particularly useful if you have a program that crashes during
startup, because it allows you to step through the program's initial
actions as quickly or as slowly as you like. It is also useful if a program
is carrying out many actions, too fast for you to read -- you can pause the
output temporarily to give you a chance to catch up. While a program is

paused, it goes to sleep.

At each step, SnoopDos will display the function the program is about to execute, before the call is actually made. To indicate that the program is currently paused, the result field will read "WAIT".

When a task is in a WAIT state, you can tell it to execute the current function but immediately pause again the next time it calls a monitored function. This lets you step through a program's function calls one step at a time. The easiest way to do this is to simply press Space on the keyboard, or click the down arrow scroll gadget using the mouse. See the SingleStep command for more details.

Note that SnoopDos pauses individual tasks, rather than the main SnoopDos program itself. Thus, if three separate programs try to call a monitored function, you will see all three tasks listed in the main window, with three WAITs in the result column. If you select single step, then all three programs are allowed to execute the current call and advance to the next one.

If you execute certain file operations while SnoopDos is paused, such as opening a new log file or selecting a new font from the font requester, then SnoopDos will temporarily unpause itself for the duration of the operation. Any tasks which were WAITing when the action began will remain in the WAIT state, but any new calls made during the file activity will be executed as normal (they will still appear in the event buffer of course).

This is to prevent deadlocks occurring -- sometimes accessing a file may involve additional operations by an external program.

When SnoopDos is paused, system activity can quite literally come to a halt. To prevent you getting into a situation where you cannot select unpause, SnoopDos will automatically unpause itself if you select the Hide command.

See also: Command Index SingleStep Unpause

1.116 quit

Command: QUIT

This command tells SnoopDos to remove all its patches and unload itself from memory. Usually, SnoopDos can quit immediately -- even though it patches many system functions, it uses a technique that allows those patches to be safely removed.

However, sometimes when you try to quit, another program will still be executing one of the functions patched by SnoopDos. A typical example is when a program tries to open a file on a disk not currently in any drive, causing a "Please insert volume..." requester to be displayed.

Until that requester is acknowledged, SnoopDos won't be able to unload itself from memory. In a situation like this, SnoopDos will check every two seconds to see if it is safe to quit; if not, it goes back to sleep again. After 10 seconds, it displays a warning requester displaying the

name of the function that is currently being executed -- this may give
you a clue as to what is going on.

It is safe to rerun SnoopDos after quitting, even if the first copy is
still not finished cleaning up. However, you can quickly use up all your
available memory if you do this too many times.

See also: Command Index   Hide

1.117 readtooltypes

Command: READTOOLTYPES
         NOREADTOOLTYPES

This command enables or disables monitoring of two separate functions in
icon.library, FindToolType() and MatchToolValue(). It is also available
as a gadget in the Function window.

Both these functions are used by programs that can be started from an icon.
FindToolType() is used to search the icon for a particular tooltype, and
MatchToolValue() is used to check if a recognised tooltype is set to a
particular value.

This can be very useful for identifying what tooltypes a program looks
for in its icon, and what values those tooltypes can take, as programmers
often forget to fully document them. Not all programs use these functions,
but most do.

For MatchToolValue(), the Target name output field shows the various
possible values for the tooltype, with each option separated by a '|'
character. The Options output field shows the actual contents of the
tooltype being checked.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index

1.118 rename

Command: RENAME
         NORENAME

This command enables or disables monitoring of the Rename() dos.library
function. It is also available as a gadget in the Function window.

Rename() is called by an application to change the name of a disk file.
Both the old and new file must be on the same partition, but need not be
in the same directory.

Rename is slightly unusual in that it produces two lines of output in the
SnoopDos window; the first line shows the original name of the file, while

the second line gives the new name.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index

1.119 rightaligned

Command: RIGHTALIGNED

This command controls how the Target name output field is displayed in
the main window. It is also available as a sub option to the Target Name
menu option on the Windows menu.

Normally, there is enough room to fully display the target name associated
with each event, so this command isn't needed. However, if you have a
narrow target name column, or if you have enabled the ShowFullPaths
option, resulting in a lot of long target names, then you may find that
the names are too wide to fit in the allocated space.

When RightAligned is enabled, the rightmost portion of each name will be
displayed, instead of the left portion as with all other output fields.
This is often more useful, since if there is not enough room to display
the entire filename, the rightmost portion includes the actual filename
which would otherwise be truncated or discarded.

When a target name is too long to display completely, the symbol '«' will
appear in the first column of output. This indicates that there are more
characters to the left of the displayed information.

Many people find it useful to leave the target name display set to
 LeftAligned most of the time and only switch to right aligned if a
particularly long target name appears. Since both modes have associated
menu hotkeys (Amiga-[and Amiga-]), the switch can be made very quickly
using the keyboard.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index   LeftAligned   ShowFullPaths

1.120 rowqualifier

Command: ROWQUALIFIER=IGNORE | NONE | SHIFT | ALT | CTRL | ALL

Example: ROWQUALIFIER=ALL

This command selects what qualifier key (if any) must be pressed before
SnoopDos will allow you to highlight a row in the main window by clicking
on it with the mouse. The Row selection key menu option lets you
change this from within the program.

SnoopDos allows you to highlight lines in the main window by clicking on them. This provides a convenient way to quickly identify all the components on a single line of output by providing a horizontal guide for your eye to follow.

However, if you use a "click to front" commodity such as that supplied with Workbench, you may notice that clicking the SnoopDos window to the front causes a row highlight to be flashed briefly. This can be distracting.

Modifying the row qualifier key allows you to avoid this situation. If your system is configured to bring windows to the front when you perform a shift-double-click, then you can set the row qualifier to NONE -- this indicates that rows will only be highlighted when no qualifier key is pressed.

Alternatively, if your system is configured to bring windows to the front when you do a double-click without holding shift (or any other qualifier), you can set the row qualifier to SHIFT, ALT or CTRL. This indicates that the named key must be held down in order to highlight a row -- clicking on the row with no key pressed will no longer be sufficient.

The ALL option allows rows to be highlighted as long as at least one of the SHIFT, ALT or CTRL keys is pressed (it doesn't matter which one). This is usually more convenient than choosing any single qualifier.

Finally, the default setting is IGNORE. When this is selected, SnoopDos doesn't care whether or not a qualifier key is pressed. If you don't use a click-to-front utility, then this is probably the best option to choose.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command index Event output

1.121 runcommand

Command: RUNCOMMAND
NORUNCOMMAND

This command enables or disables monitoring of the RunCommand() dos.library function. It is also available as a gadget in the Function window.

Applications call RunCommand() to execute a single disk command with a particular set of parameters. The event output produced for this function is a little unusual, because the name shown in the Process name output field is usually the name of the command being executed, and not the process making the call. The Target name field lists only the command line parameters for the command, and not the command name itself.

The Options field shows the stack size provided for the command; if this is too small, some commands may not run properly. Finally, the Result field is displayed as "----" while the command is executing, and is replaced with the return code from the command after execution completes (usually 0 for a successful execution). A result of "Fail" indicates that the command couldn't be executed at all.

Since the executed command will probably produce some SnoopDos output of its own, you may need to scroll back in the SnoopDos buffer to retrieve the final return code from the RunCommand line if you need to find it for some reason.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index Execute System

1.122 savebuffer

Command: SAVEBUFFER "filename"

Example: SAVEBUFFER "ram:SnoopDos.txt"

This command saves the entire contents of the SnoopDos event buffer to the named file. This is useful if you want to preserve the output from SnoopDos for later analysis, but forgot to open a log file first.

This function is also available as a menu item on the Buffer menu.

See also: Command Index CopyBuffer CopyWindow SaveWindow

1.123 savedefsettings

Command: SAVEDEFSETTINGS

This command saves the current SnoopDos settings to the default settings file, as defined by the Settings command. It is also available as a gadget in the Main window, and as a menu option on the Settings menu.

If no file has been explicitly defined, then the settings will be written to the file that was loaded during startup. If no such file could be found, then they will be written to ENVARC:SnoopDos.prefs instead.

The next time you load SnoopDos, these settings will automatically be used. The settings saved include the current values of all the gadgets in the Setup and Function windows, and the current value of most boolean menu items, as well as the position and size of each window.

See also: Command Index LoadDefSettings LoadSettings
SaveSettings Settings

1.124 savesettings

Command: SAVESETTINGS "settingsfile"

Example: SAVESETTINGS "ram:SnoopDos.prefs"

This command writes the current values of all gadgets and menu items to the named file. This file can then be reloaded at a later time to reconfigure SnoopDos with those settings.

The settings file generated is plain text and can be edited by hand if required.

See also: Command Index LoadDefSettings LoadSettings SaveDefSettings Settings

1.125 savewindow

Command: SAVEWINDOW "filename"

Example: SAVEWINDOW "ram:SnoopDos.txt"

This command saves the contents of the main window to the named file. It is also available as a menu option on the Buffer menu.

If the current buffer format is too wide to fit completely in the main window, only that portion which is currently displayed will be saved. If this doesn't include enough information, you can use the SaveBuffer command instead and then delete any extra lines that you don't need.

See also: Command Index CopyBuffer CopyWindow SaveBuffer

1.126 screenname

Command: SCREENNAME "Screen Name"

Example: SCREENNAME "Workbench"

This command works in conjunction with the NAMED option of the ScreenType command to allow SnoopDos to open its windows on a particular public screen. It is also available as a gadget in the Settings window.

If the screen name can't be found, then SnoopDos will use the default screen instead. If the current screen type is set to FRONT or DEFAULT, then the screen name defined here will be ignored.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index HotKey ScreenType

1.127 screentype

Command: SCREENTYPE=DEFAULT | FRONT | NAMED

Example: SCREENTYPE=FRONT

This command determines what screen SnoopDos will use for its windows.
It is also available as a gadget in the Settings window.

If the screen type is DEFAULT, then SnoopDos will open on the default
public screen. This is usually the Workbench screen.

If the screen type is FRONT, then SnoopDos will try and open on the
frontmost public screen. Many modern application programs automatically
make their screens public, and you can use third party utilities to
create other public screens. This mode of operation is convenient,
because it allows you to call up SnoopDos on your current screen at
any time by just pressing the currently defined Hotkey .

If the screen type is NAMED, then SnoopDos will try and open on a public
screen with a name that matches the currently defined ScreenName . Note
that SnoopDos won't create a screen of this name itself -- the screen
must already exist. If no such screen is found, then SnoopDos will open
on the default screen instead.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index HotKey ScreenName

1.128 scrolldown

Command: SCROLLDOWN <n>

Example: SCROLLDOWN 10

This command moves the SnoopDos event buffer down by <n> lines. It is
equivalent to using the down arrow gadget in the Main window, or
pressing the down arrow cursor key.

See also: Command Index GotoLine ScrollUp

1.129 scrollup

Command: SCROLLUP <n>

Example: SCROLLUP 10

This command moves the SnoopDos event buffer up by <n> lines. It is
equivalent to using the up arrow scroll gadget in the Main window
or pressing the up arrow cursor key.

See also: Command Index GotoLine ScrollDown

1.130 sendrexx

```
Command: SENDREXX
         NOSENDREXX
```

This command enables or disables monitoring of ARexx messages sent by ARexx scripts or applications. It is also available as a gadget in the Function window.

This can be useful if you want to see how a script is performing its magic, or if you're trying to figure out why an application isn't responding as you expect.

The Target Name output field shows the command being sent to the application, and the Options output field shows the application's port name.

When this function is enabled, SnoopDos monitors every single message sent by the system, in order to identify which ones are ARexx messages. Thus, you may find that your system runs a little slower.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index FindPort MonitorPackets PacketDebugger

1.131 settings

```
Command: SETTINGS="name"

Example: SETTINGS="ENVARC:SnoopDos.prefs"
```

This command defines the name of the default settings file that SnoopDos tries to load at startup. Normally, this is something that you would include in the SnoopDos icon's tooltypes, or as a CLI command line option.

If you don't define an explicit settings file, SnoopDos will look in the PROGDIR:, ENVARC: and S: directories at startup for a file called SnoopDos.prefs. If it can't find such a file in any of those three directories, it will use the name ENVARC:SnoopDos.prefs for future saves.

Note that this command doesn't actually cause the settings file to be loaded; it only shows where the settings file might be found. When you use the Save Settings gadget in the Main window, this is the filename that will be used.

You can also use this command from within a settings file. For example, suppose you wanted to carry out a certain set of actions each time SnoopDos started -- open a new log file, open the format editor, etc.

Normally, if you included those commands in a settings file, they would be overwritten the first time you saved settings. However, by using the SETTINGS command to change the name of the settings file from within the settings file itself, you can avoid this. For example:

```
    <SnoopDos Settings>
    SETTINGS "ENVARC:SnoopDos.realprefs"
    LOADDEFSETTINGS
    OPENLOG "ram:SnoopDos.log"
    OPENFORMAT
```

If this file was saved as ENVARC:SnoopDos.prefs, then SnoopDos would start out by loading the real default settings from "ENVARC:SnoopDos.realprefs", then open a log file, and finally open the format editor. Any future saving of default settings would also be directed to ENVARC:SnoopDos.realprefs, and so the command file would not be overwritten.

See also: Command Index LoadDefSettings LoadSettings SaveDefSettings SaveSettings

1.132 setupwindowpos

Command: SETUPWINDOWPOS=<left>,<top>

Example: SETUPWINDOWPOS=0,20

This command is used to change the position of the settings window on the current screen. If the settings window is open, then it is moved immediately, else it will appear in the new position the next time it is opened. <left> and <top> are the pixel co-ordinates of the upper left corner of the window.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index Settings window

1.133 setvar

Command: SETVAR
NOSETVAR

This command enables or disables monitoring of the SetVar() and DeleteVar() dos.library functions. It is also available as a gadget in the Function window.

SetVar() is the logical counterpart to the GetVar() function; it allows a program to create or update an environment variable with a new value. As with GetVar(), Global, Local and Alias variables are recognised, and a trailing * after the type indicates that the program expects the variable to contain binary information, rather than plain text. See the GetVar command for more information.

DeleteVar() is used to delete environment variables from memory. The same three variable types described above are recognised.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index

1.134 show

Command: SHOW

This command opens the SnoopDos main window on the currently selected
screen (as defined by the ScreenType command). If the window was already
open, then it is brought to the front. If the window was open, but on
different screen, then it is re-opened on the current screen.

This command is automatically executed whenever you press the currently
defined HotKey . It is also automatically executed if you run SnoopDos
from the CLI with no command line options, or if you double-click on a
SnoopDos icon while SnoopDos is already running.

See also: Command Index   CX_Popup   Hide   HotKey   Main window

1.135 showcli

Command: SHOWCLI
         NOSHOWCLI

This command enables or disables SnoopDos's ShowCLI setting. It can also
be controlled via a gadget in the Function window.

The ShowCLI setting determines how the Process name field displays the
name of the CLI command that called a monitored function. When enabled,
the command name is prefixed with the number of the CLI in which it is
running (e.g. "[4] Multiview"). If disabled, then only the name of the
command is displayed. Background processes and programs started from
Workbench are not affected.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index

1.136 showfullpaths

Command: SHOWFULLPATHS
         NOSHOWFULLPATHS

This command enables or disables SnoopDos's ShowFullPaths setting. This
setting can also be controlled via a gadget in the Function window.

Whenever a program passes a filename to an AmigaDOS function, the filename

is interpreted as being relative to the program's current directory, unless an absolute pathname is specified with the filename.

If the program being monitored changes directory often, it can become difficult to keep track of which directory it is in at any given time. Selecting this option makes SnoopDos always prefix filenames with a full path before displaying them in the window.

This can produce rather long filenames which occupy a lot of space in the SnoopDos window, so you should only use this option if you have a specific need for it, rather than leaving it on all the time.

If you have many long filenames displayed, you can use the menu shortcut keys for LeftAligned and RightAligned , Amiga-[and Amiga-], to quickly switch between viewing the start and end of the filename.

Note that if a program tries to access a file on a volume that is not currently mounted, SnoopDos will be unable to resolve the full directory path to that file. In this case, only the volume name will be shown, followed by ellipses and the filename, to indicate that the directory information is unavailable. For example, "Work:.../filename".

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index UseDeviceNames

1.137 showgadgets

Command: SHOWGADGETS

This command enables the display of the panel of button gadgets in the Main window. It is also available as a menu option on the Windows menu.

Having the gadgets easily available is convenient, but some people prefer to have an extra line or two of event output instead. Even when the gadgets are disabled, the keyboard shortcuts for each gadget still operate. The gadgets all have equivalent menu options which can be selected as well.

To maximise the amount of room available for displaying event output, you may also like to turn off the Status line gadget.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index HideGadgets HideStatus
ShowStatus TextSpacing

1.138 showstatus

Command: SHOWSTATUS

This command enables the display of the status line gadget in the Main window. It is also available as an option on the Windows menu.

The status line is not strictly necessary, but it provides a useful "at-a-glance" summary of whether or not a log file is currently open, and whether SnoopDos is currently Paused or Disabled .

Some people prefer to sacrifice the status line to allow more lines of event output to be displayed in the main window. If you are doing this, you may like to also disable the Button gadgets, which will allow the maximum number of lines of text to be displayed.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index HideGadgets HideStatus
ShowGadgets TextSpacing

1.139 simplerefresh

Command: SIMPLEREFRESH

This command sets the refresh method used for all SnoopDos windows to Simple refresh. It is also available as a sub option to the "Window Type" menu option on the Windows menu.

Simple refresh windows help conserve memory. When a window is partially obscured, the obscured portion is discarded, and is then redrawn by SnoopDos when the window is later uncovered. Redrawing the window can take some time on slower Amigas, where Smart refresh windows may be more appropriate.

However, some third party graphics cards actually perform better with simple refresh windows than with smart refresh. The best advice is to try both methods and choose whichever you like best.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index SmartRefresh

1.140 singlestep

Command: SINGLESTEP

This command is used when SnoopDos is paused. It allows each of the currently waiting tasks to execute the current function and advance to the next one. See the Pause command for more information.

SingleStep is also available as a menu option on the Projects menu. As an alternative, you can click the down arrow scroll gadget in the main window when the scroll bar is positioned at the end. You can also press

Space or Return on the keyboard for convenience. The TAB key will also single step, but has the additional advantage of putting SnoopDos into Pause mode if it wasn't already paused.

If you have a buffered log file open, then SnoopDos will flush the buffer to disk each time you single step. This allows the most information to be retained in the event of a crash. However, it also means that your disk may become corrupt if a crash should occur in the middle of a disk write.

Thus, if you suspect that a particular program is likely to crash, you should probably experiment without a log file first, or log to a safe device like PRT: or RAD:.

See also: Command Index Pause Unpause

1.141 smartrefresh

Command: SMARTREFRESH

This command sets the refresh method used for all SnoopDos windows to Smart refresh. It is also available as a sub option to the "Window Type" menu option on the Windows menu.

Smart refresh windows are popular, because they allow a partially obscured window to be instantly redrawn when it is uncovered, with no obvious delay.

However, they take up more memory than Simple refresh windows. They can also be slower on some third party graphics cards, such as the Picasso, especially when running with 256 colours.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index SimpleRefresh

1.142 stacklimit

Command: STACKLIMIT=<minstacksize>

Example: STACKLIMIT=1000

This command defines how much free stack space a task or process must have before SnoopDos will monitor one of its function calls. This is needed because some tasks installed by devices have very small stacks and the additional stack space used by SnoopDos would cause a crash if it tried to monitor the function. (CrossDOS is one such example.)

<minstacksize> is the required amount of free stack space in bytes. Note that this does not necessarily correspond to the total stack size for the task, but to the free stack at the time the function call is made.

For example, a task with a 20K stack could use up 19.5K of space before entering a SnoopDos patch (and so would be ignored if StackLimit was 500 or above) whereas a task with a 2K stack might only use 0.5K, leaving 1.5K free for SnoopDos to use.

SnoopDos defaults to a stack limit of 1000 bytes. This should be more than sufficient for most purposes, but if need be, this can be reduced to about 600 bytes. You should only need to do this if you are trying to monitor a particular task known to have a very small stack.

To determine the current stack size, SnoopDos uses the tc_SPLower and tc_SPUpper values in the task structure. If a task or process has allocated its own stack, then these may not be correct -- if this is the case, SnoopDos will go ahead and monitor the function call anyway, on the assumption that the application will have allocated a stack big enough to handle SnoopDos's requirements.

Note that there is no way to set this option from within SnoopDos; you must change it by editing the SnoopDos default settings file directly, or by adding it as a tooltype to the SnoopDos icon.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index PatchRamLib

1.143 system

Command: SYSTEM
NOSYSTEM

This command enables or disables monitoring of the System() dos.library function. It is also available as a gadget in the Function window.

System() is one of the most powerful functions in AmigaDOS. It allows an arbitary command line to be executed, exactly as if it was typed at a Shell prompt.

As with RunCommand the Options field shows the stack size allocated for the command. The Result field shows the return code from the command (usually 0), or "Fail" if the command could not be executed.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index Execute RunCommand

1.144 taskpri

Command: TASKPRI <n>

Example: TASKPRI 5

This command is used to change the priority of the main SnoopDos process.
A selection of common priorities is available via the similarly named
menu item on the Project menu.

Normally, SnoopDos defaults to running at priority 0. However, you may
prefer to run at a higher or lower priority.

If SnoopDos runs at a priority above 0, then it can always keep up to
date when monitoring events. This is particularly true if you are using
SegTracker to determine the calling module for each event, since
otherwise, a module can sometimes unload itself before SnoopDos has a
chance to ask SegTracker to locate it. However, running at a higher
priority can result in an increase in the number of task switches, which
can slow down your machine a little.

If SnoopDos runs at a priority lower than 0, then it will have very little
impact on system performance, even when its window is open. However, the
output will only be updated when no other programs are waiting to run, so
it may appear in sudden bursts, rather than evenly. Even at a low priority,
SnoopDos will not miss any events.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index Pause

1.145 textspacing

Command: TEXTSPACING <n>

Example: TEXTSPACING 1

This command sets the number of extra pixels of space between adjacent
lines of output in the Main window. You can also change the text
spacing using the menu option on the Windows menu.

If you are using a small font, increasing the spacing can make the output
easier to read. The valid values for <n> are 0, 1 or 2.

When the spacing is changed, the buffer contents will be immediately
redrawn using the new spacing. However, the position of the window gadgets
will not be adjusted until the next time the window is resized. Thus, you
may find that if you resize the window slighly, you can fit an extra line
of text as the gadgets reposition themselves to accomodate the change.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index ShowGadgets ShowStatus

1.146 unpause

Command: UNPAUSE

This command undoes the effect of the Pause command. It can be accessed by deselecting the Pause menu option in the Project menu or the Pause gadget in the main window. You can also press SHIFT-TAB to unconditionally cancel any Pause or Disable currently in effect.

Deselecting Pause allows any tasks that were previously in WAIT state to resume operations. To allow such tasks to execute a single function call and then go back into WAIT mode again, see the SingleStep command.

See also: Command Index Pause SingleStep

1.147 usedevicenames

Command: USEDEVICENAMES
NOUSEDEVICENAMES

This command enables or disables SnoopDos's UseDeviceNames setting. This setting can also be controlled via a gadget in the Function window.

UseDeviceNames works together with the ShowFullPaths setting. If UseDeviceNames is enabled, then any filenames that are expanded to full paths will be displayed using the disk device name. If disabled, then the disk volume name is used instead.

For example, when UseDeviceNames is enabled, an expanded name might appear as "HD0:Files/Data". The same name shown with UseDeviceNames disabled would then appear as "System3.0:Files/Data".

Using the device name is often preferred, since device names are typically only three characters in length, and so take up less space. They are also normally unique, which can sometimes help to prevent confusion.

SnoopDos automatically writes this command to the settings file when you save settings.

See also: Command Index ShowFullPaths

1.148 windowfont

Command: WINDOWFONT "fontname.size"

Example: WINDOWFONT "helvetica.11"

This command selects the font used by SnoopDos for all window gadgets. You can also change the font by using the font requester accessible through the Settings window.

Any font can be selected, and you must specify a size. If the font is too large for a particular window, then SnoopDos automatically switches to a

smaller font for that window only.

SnoopDos automatically writes this command to the settings file when
you save settings.

See also: Command Index   BufferFont

1.149 windowwidth

Command: WINDOWWIDTH <size>

Example: WINDOWWIDTH 80

This command adjusts the width of the SnoopDos main window so that exactly
<size> number of characters can be displayed horizontally. If the main
window isn't open, then this command has no effect. If obtaining the
requested width would make the window too small or too large, SnoopDos
will get as close as possible to the width you asked for.

Zero is a special value which will make the width just wide enough to
fully display the current format string.

The main reason you might want to use this command is so that you can more
easily create a custom format suitable for outputting to a file or printer.
This is particularly true if you run Workbench at a high resolution like
1024 x 768, where the SnoopDos window might be much wider than the usual
80 characters.

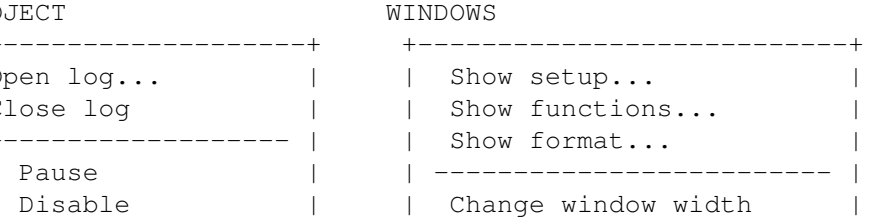
Keep in mind that when SnoopDos is printing to a log file, it adds one
additional character at the start of each line. Usually that character is
a space, but it can sometimes be '\' or '/' -- see the OpenLog command
for more details. In any case, this extra character needs to be taken into
account when you set the window width. For example, a 79 character window
width will result in 80-character lines in the log file.

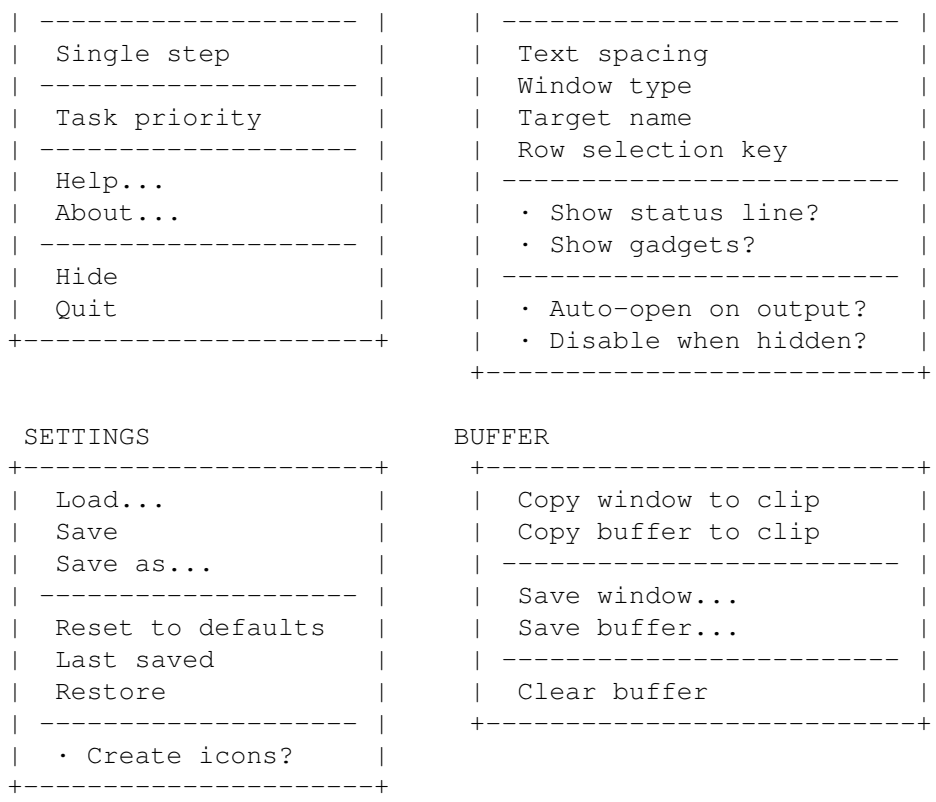
For convenience, the SnoopDos Windows menu offers several common window
widths via the "Change window width" submenu.

See also: Command Index   Format editor

1.150 menu_index

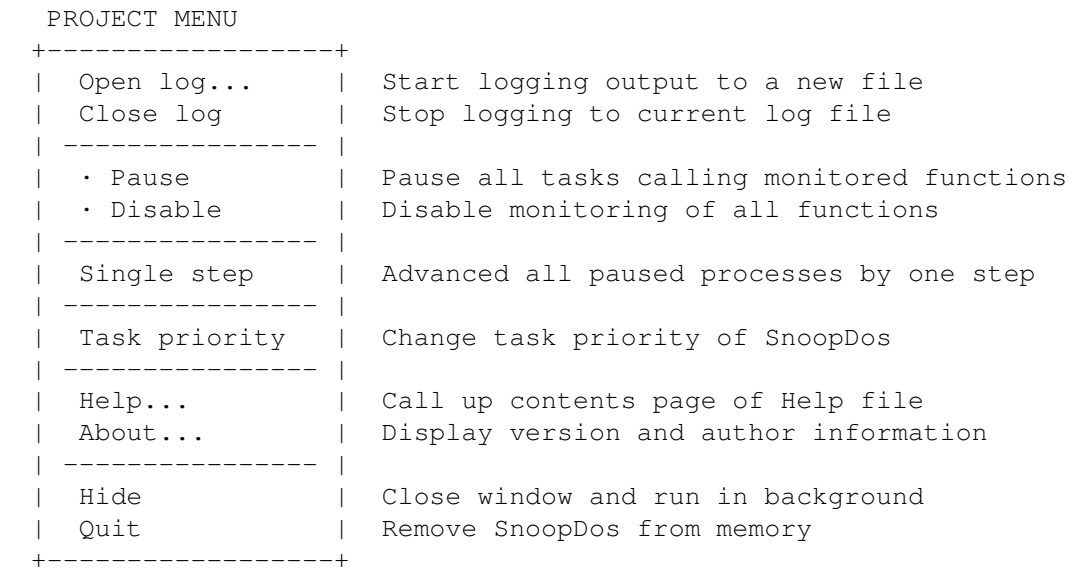
SnoopDos provides the following menus in the Main window.
Click on any item for additional information.





1.151 SnoopDos Project menu

The SnoopDos project menu contains the following program options.
Click on any option for additional information.



See also: Main window Windows menu Settings menu Buffer menu

1.152 Project menu/Open log...

PROJECT MENU / OPEN LOG...

This menu option lets you open a new log file. An ASL file requester will be displayed to let you choose the name of the log file. If you choose a file which already exists, SnoopDos will give you the option of overwriting the existing file, or appending new output to the end of the file.

If you want to log to a device rather than a disk file, type the device name in the Directory gadget of the ASL requester (e.g. PRT:). In this case, the filename should be left empty, unless you are logging to a CON: window, when the filename should contain the last window option (which is usually the window name or CLOSE).

When a log file is open, SnoopDos copies all output displayed in the main window to that file, allowing a permanent record to be kept. Usually, the format of the log file is the same as that displayed in the main window, but you can change this using the LogFormat option.

Note that unlike the Open log button in the main window, this option will always display a file requester. When a log file is open, this menu option is disabled. Selecting the Close log menu option enables it again.

See also: Project menu Main window OpenLog

1.153 Project menu/Close log

PROJECT MENU / CLOSE LOG

This option is the counterpart of the Open log menu option. It closes any currently open log file. If no log file is open, then this option is disabled.

See also: Project menu Main window CloseLog

1.154 Project menu/Pause

PROJECT MENU / PAUSE

Selecting this option causes any tasks that call a function monitored by SnoopDos to pause before executing the function call. This allows you to see the function parameters before the call is actually carried out. You can then use the Single step menu option to execute the call and advance to the next one.

Deselecting this menu option will allow the tasks to run as normal again. For a more comprehensive description, see the Pause command.

See also: Project menu Main window Pause

1.155 Project menu/Disable

```
PROJECT MENU / DISABLE

Selecting this option disables monitoring of all functions. It corresponds
to the  Disable  command.

This provides a convenient way to turn off SnoopDos without having to
quit, or individually deselect each function. To re-enable monitoring,
simply deselect the menu option.

See also:  Project menu    Main window    Disable
```

1.156 Project menu/Single step

```
PROJECT MENU / SINGLE STEP

This option works in c
```