

# SuperDuper

---

An Amiga Disk Copier  
Version 2.02

by Sebastiano Vigna

---



# 1 Introduction

**SuperDuper** is a disk copier/formatter that tries to be to disk handling what *Mostra* is to IFF displaying: a fast, compact, system-friendly tool which combines speed, features, and some bells and whistles to make your life easier.

By *fast* I mean exactly what you're hoping—blazingly fast. A disk is usually copied and verified in less than 100s. Without verify, the time drops to 69s. You can buffer a disk in RAM in less than 36s, and then making a verified copy takes 67s, while a non-verified copy takes less than 36s. Adding another destination drive increases verified copy times by 34s, but hardly changes non-verified copy times (the Amiga can write more than one drive at a time; I just need a few tenths of a second in order to measure the drive speed and step the heads). Thus, if you really trust your drives and your media you can make four copies in 38s. These timings can vary with the system configuration, the multitasking overhead, the disposition of the blocks on the surface of the disk, the state of the `DATE` option (which requires a separate write on the root block track for each disk) and the DMA access of the custom chips (previous users of **SuperDuper** might think this release is slower than the previous one: it is really faster, but **SuperDuper** 1.0 was a little bit optimistic about its copy times—the motor on/off delays were erroneously skipped).

## 1.1 Changes

**SuperDuper** 2.01 has almost no visible changes with respect to **SuperDuper** 2.0, apart from the extension of the `ARexx` macros, which is now `'supdup'` instead of `'sd'` in order to avoid conflicts with other programs.

However, it was discovered that many flakey drives have power supply problems when four of them are connected to an Amiga. Sometimes the head of a drive won't step, and this error cannot be caught even by verifying, since the drive doesn't know where the head is—its position has to be tracked via software.

In order to prevent this annoying phenomenon, which was the only known source of bad copies, the head moving strategy was slightly changed. **SuperDuper** no longer steps multiple heads at the same time. This marginally increases (about 3 tenths of second for each destination) the non-verified copy times, but gives you a 100% reliability even on out-of-specs systems.

If something wierd happens in spite of this patch, it can be tracked at recalibration time. When a mismatch between **SuperDuper**'s internals and the drive signals is detected, a requester `'Error`

`while recalibrating`' is issued. In this case, you can try to slow down the head moves using the `SetTDDelay` utility which is supplied with **SuperDuper**.

The only other noticeable enhancement is the fact that now **SuperDuper** checks the `NOCLICK` flag separately for each drive. If you have some drives which support `NOCLICK` and some which don't, **SuperDuper** will click only the allowed drives. Previously, the information in the public unit of the drive 0 was used for all the drives. The utility `ToggleClick` which is supplied with **SuperDuper** allows to hush selectively any unit. Moreover, a new `noclick` ARexx command allows to force no-clicking selectively even under 1.2/1.3 (see Section 4.3 [Selection Commands], page 14).

It should be remarked that **SuperDuper** is much less tolerant than `trackdisk.device`. Bad drives can work (almost) perfectly with `trackdisk.device`, because of its many, frequent cross checks. For instance, at each disk insertion some track is read by DOS, and if the track number doesn't match with `trackdisk`'s internals, a recalibration is started. **SuperDuper** instead doesn't read anything before copying (for speed reasons); thus, if your drive has a flakey `DSKTRACK0` signal **SuperDuper** could believe it's on track 0 while it isn't.

**SuperDuper 2.02** is a minor maintenance release. Let alone a couple of bug fixes (notably the French keyboard problem), support for Release 2 Public Screens was implemented (see Section 5.2 [The CLI Interface], page 19). Moreover, the `requesters` ARexx command, which has always been present, is now documented (see Section 4.3 [Selection Commands], page 14).

## 1.2 Main Features

- **SuperDuper** copies, formats and checks from/to any combination of Amiga drive(s).
- **SuperDuper** has a switchable 880K RAM buffer that allows for any number of duplications while reading the source disk only once. The combination of destination drives can be changed at each pass. If you have a hard disk, you can create on it an image file that will act as a buffer. This file can be saved and reused many times. Also, all kinds of virtual disks are supported for buffering (`VD0:`, `RAD:`, `FMS:...`). Moreover, a count is kept of the copies generated by a buffered disk.
- **SuperDuper** checksums the RAM buffer. If some badly written program is trashing your memory, you are alerted. Thus, buffered copies are as safe as direct copies.
- **SuperDuper** also checks its internal DMA buffers at each write.
- **SuperDuper** can allocate a buffer of less than 880K. In this case, it will use real-time compression in order to do multiple pass copies with maximum efficiency. Most disks can be wholly buffered on a 1MB machine. You can also make multiple copies with multiple passes. Copy times are (almost) unaffected.

- **SuperDuper** will automatically retry tracks which produce a verify error. The number of retries is programmable. A simple visual clue is given to the position of the error, but on request detailed error information printing is available.
- **SuperDuper** is highly system-friendly—the use of CPU time is negligible, so you can multitask efficiently.
- **SuperDuper** has the option of incrementing the creation date of the copy so AmigaDOS doesn't get confused. If, however, the option is switched on and the disk is not an AmigaDOS disk, **SuperDuper** won't increment the date.
- **SuperDuper** is faster than diskcopy—actually it pushes the drives to their limits. At the time of this writing, **SuperDuper** is the fastest Amiga copier both from a “pure” (physical time) and from a “per-copy” (real time for each copy when a big number of copies of the same disk is produced) point of view.
- **SuperDuper** alerts the user with sound (and optionally voice) about the operations in progress—so you can really be doing something else!
- **SuperDuper** can format both OFS and FFS disks.
- **SuperDuper** displays a list of the last few disks copied. If you do a lot of copying, you'll find this feature more than a little useful.
- **SuperDuper** can manage the Amiga drives without help from the trackdisk.device. Through the supplied utility **SDBootInstall**, you can create a boot disk which will keep the system away from your drives, giving you back more than 30K per unit. This is very useful when doing intensive buffered copying on a 1M machine.
- **SuperDuper** can automatically start any copy or format operation by monitoring the disks' extraction and insertion.
- Almost all elements of the 3-D, Release 2-like graphical user interface have keyboard equivalents. When possible, Release 2 features like TAB gadget cycling and window zooming were supported.
- **SuperDuper**'s window can be opened on any public screen under Release 2.
- The start/end cylinder of a copy is programmable.
- Unique numbered names can be automatically generated while formatting.
- **SuperDuper** works under 1.2, 1.3 and 2.0.
- **SuperDuper** has a time indicator.
- **SuperDuper** has a beautiful name. 8~)
- If this is not enough, an ARexx interface allows any kind of customization. In particular, a startup ARexx script lets you set up a custom configuration. Since **SuperDuper** can turn off its graphical user interface via a command line switch, it is possible to use **SuperDuper** as a CLI command by writing a suitable ARexx macro. A switch allows you to shut down ARexx in order to gain memory. ARexx macros can be launched via a file requester (asl.library, arp.library and req.library are supported).

## 1.3 First Steps

To use **SuperDuper**, you simply double-click on its icon. You will see five rows of gadgets. The first one has at most one gadget selected: it's the source. The second one lets you select the destination drive(s). The other lines contain option and action gadgets.

Every gadget can be activated via mouse or keyboard (using the underlined letter). The line of destination drives can be controlled by pressing **SHIFT** together with the underlined number. You can use **Q** or **ESC** to exit, instead of hitting the close gadget. Three of the string gadgets have underlined letters which activate them. Moreover, if you're running under Release 2, you can use **TAB** or **SHIFT-TAB** to pass from a string gadget to another one.

To make your first copy, if you have two (or more) drives simply select in the first line the gadget for the drive which contains the source floppy, and in the second line the gadget(s) for the drive(s) containing the destination(s) (for the time being do not choose the same drive both as source and as destination). Then hit the **GO** gadget. After a while, the display will flash, a beep will be generated, and the copy will be finished. As each cylinder is copied, the elapsed time indicator is updated. Note that a first beep will be generated when the copy is almost finished, so you have time to prepare yourself.

If you have only one drive, select it both as source and as destination. Then hit the **BUFFER** gadget, thus creating a RAM buffer. Depending on the memory available, it will be a full 880K buffer or a partial buffer. In the latter case, real-time compression will let **SuperDuper** get the best out of it. Now put in the source disk and hit the **READ** gadget: the buffer will be filled with the contents of the disk. If the progress bar reaches its maximum length, then the whole disk has been buffered. Pull out the source disk, put in the destination, and hit the **GO** gadget. The buffer will be written to the disk. If only a part of the source disk was buffered, put it in again, buffer it again (note that now the progress bar starts where it stopped before) and write it again. This process must be repeated until the whole disk has been copied. It is safer to set the write protect tab on the source disk, in order to avoid the unpleasing side-effects of source/destination mismatches.

## 2 Gadgets

### 2.1 The Action Gadgets

Four gadgets control **SuperDuper**'s copy/format/check operations:

STOP	stops any operation. If pressed while the multi-pass real-time compression buffer is selected and no operation is in progress, it will empty the buffer and reset the pass count, thus allowing you to buffer another source even if the previous one wasn't finished (see Section 2.3 [The Buffering System], page 7). If you STOP immediately after starting a copy operation and nothing has been drawn in the progress bar, nothing has been written to the destinations.
READ	can be used only when a buffer is selected; it fills the buffer by reading from the source drive.
GO	initiates a copy operation. If no buffer is selected, the source is copied to the destination(s). If a buffer is selected, the content of the buffer is written on the destination(s). If FORMAT is selected, the destination drive(s) are formatted.
CHECK	is basically a READ without buffering. The source disk is scanned for errors. No buffer is needed to use it. Note that <b>SuperDuper</b> will detect trackdisk.device related errors, but it won't find DOS checksum errors (for this purpose, for instance, you can use <b>FixDisk</b> ).

When **SuperDuper** starts an operation which involves reading a disk, i.e., READ, CHECK and non-buffered GO, it scrolls up the name list and marks the current drive as '<UNKNOWN>'. This happens because it can't know if the disk is a DOS disk before reading track 0. After less than a second, the track will be read, and the name will be changed to '<NDOS>' if the disk is not a DOS disk. Otherwise, as soon as the track 80 is read (the progress bar is in the middle) the name of the disk will be displayed. However, if for any reason the name is incorrect (wrong format, read error, etc.) **SuperDuper** will name the disk '<BAD NAME>'. In this case, it is very likely that the root block is a little bit scrambled, so it's probably a good idea to turn off the DATE option gadget. Beware: if you are using a multi-pass buffer, the name of the disk could be unavailable at the first pass.

If **SuperDuper** finds an error on read (or verify), it will retry reading (writing) the track, each time incrementing the first number of the RTRY:ERR indicator. If after a number of retries specified in the gadget RETRY# the error remains, **SuperDuper** will increment the second number (the error counter), restore the original retry counter and continue. A little rectangle in the progress bar will point out where the error occurred. It will be positioned horizontally, proportionally to the track number, and vertically, proportionally to the unit number (the first line of rectangles shows errors on unit 0 and so on). Note that while retrying **SuperDuper** can't be stopped: don't set the RETRY# gadget to 99 unless you really know that's what you want to do. At the end of the copy, the first number shows how many retries leading to a successful write were done, while the second one points out the number of tracks with an actual error. If you want to get a very detailed error report, you can set **printererrors on** from ARexx. A console window will appear, and every wrong read, write or retry will generate a message explaining what doesn't work. Usually you will get bad checksums, but if a track is really scrambled **SuperDuper** could be unable to get the first sector after a gap, in which case nothing at all is recovered.

The progress bar is drawn in a different color if you're doing an read, a format or a copy operation—so you can be sure you read the new chunk in the buffer, and so you can avoid formatting your floppies when you think you're copying something to them. The gadget corresponding to the action currently executing will remain highlighted in order to remind you what you're doing. Note also that the progress bar and the elapsed time indicator are not updated if something locks the screen (like using menus). The update is delayed until the screen is unlocked (thus **SuperDuper** won't get stuck as will almost all programs which do any rendering to their windows).

The volume of the beeps produced by **SuperDuper** while copying can be set with the `ARexx volume` command.

If you specify start/end cylinders different from 0/79 in the SC and EC gadgets, only the part of the disk specified will be copied. The main use of this option is for retrying some lazy disk (usually on the last tracks) if you're not satisfied with the number of retries issued by **SuperDuper**. Please refer to the section on the buffering system for some subtle interactions between the RAM/HD/VDisk buffer and the start/end cylinder selectors.

While doing buffered copies, at each successful copy (that is, without errors) the `CPY#` indicator will be incremented. Thus you can know precisely how many disks you copied. Moreover, the counter will be incremented only if the operation ended on the last track of the disk and started from the first track of the buffer. This allows you to manually retry spare tracks by changing the CS/EC gadgets without getting spurious increments, and if a multi-pass copy is in progress only the last pass will actually increment the counter.

## 2.2 The Copy Options

Five gadgets control the copy/format options.

VERIFY	turns verify on and off (you can also format without verifying). However, turning off verify is not recommended.
DATE	toggles on or off the change of the date of an AmigaDOS disk. This change is necessary so AmigaDOS can distinguish otherwise identical disks; if two truly identical disks are inserted in the drives, AmigaDOS gets confused and crashes. However, if for some reason you want a “physical” copy, you would turn off this option. DATE will be ignored for a non-AmigaDOS disk.
FORMAT	enables formatting. When you hit GO, all destination drives will be formatted. To copy again, you must deselect FORMAT by clicking it again. If VERIFY is selected, the format process is verified. Note that when you hit READ, FORMAT is automatically

deselected. This happens in order to avoid the unpleasant error of thinking you're writing a buffered disk, while actually formatting it.

The gadget prefixed by LABEL: allows you to choose a name that **SuperDuper** will use while formatting. The name must be chosen before clicking GO—it is disabled (ghosted) during the formatting.

- INCNAME makes easy to format a bunch of disks with different, unique names. If this gadget is selected while formatting, **SuperDuper** will scan the Label string gadget searching for a numeric pattern (i.e., one or more digits) and will increment the pattern value for each disk formatted. In case more than one pattern is present, the last one is used. For instance, if you format four disks with label 'Foobar.000', the disks will be named 'Foobar.001', 'Foobar.002',... and at the end of the copy the label gadget will contain 'Foobar.004', thus being ready for the next formatting. The more digits, the more unique names. Since you can start from any number, and after 99...9 the numeration wraps around, if you need to start with 00...0 you can put in something like 'Foobar.999': The first disk will be labeled with 'Foobar.000'
- FFS enables the formatting of FFS disks; for copying it is ignored.

## 2.3 The Buffering System

Three gadgets control the full-featured buffering system of **SuperDuper**. Buffering is useful when you have to do a lot of copies: you read a disk only once, and then you can make as many copies as you want without rereading it. It also has other uses: if you have to create distribution disks (for instance for a commercial package) you can create them using high speed virtual floppies, such as Commodore's RAD: or Matt Dillon/Jim Cooper's FMS: disk. **SuperDuper** can then read from those virtual disks and make many copies on floppies at high speed.

Since data integrity is a primary issue, **SuperDuper** checksums the RAM buffer. The possibility of writing a munged track is very low. Strict control is also kept on the validity of the buffer—you can't write random data on your disks inadvertently.

- BUFFER allocates a RAM buffer. **SuperDuper** will try to get an 880K buffer: if you don't have enough memory, a warning will be issued, showing the number of buffers allocated (each buffer is 11K) and warning you that the real-time compression system is activated. Beware of the fact that many programs tend to crash under low-memory conditions, so if you have 1MB or less you should close everything you can before hitting BUFFER, and you should possibly also use KILLSYS (see Section 2.4 [The User Interface Gadgets], page 10).

The memory allocated will be used as a buffer to make multiple pass copies. If **SuperDuper** can find 880K, the process is very simple and uses very little of the CPU, but if (for instance on a 1MB machine) it's impossible to buffer a whole disk this way, **SuperDuper** will use a real-time compression algorithm. As the disk is read in the buffer, it is compressed in a special format. The gain in size is usually 35% for empty tracks, 20-30% for text, 15-25% for programs and 5-10% for IFF ILBM images. Tracks which can't be compressed are simply stored. The only disks which can't really be compressed are disks filled with compressed files, like .lzh or .zoo files, but for the others the size gain is enough to buffer a whole disk on a 1MB machine. 8~)

Of course, the compression overhead eats a lot of CPU power. The algorithm has been devised in such a way that compression and decompression are absolutely real-time, i.e., you will notice no slowdown. However, beware of the fact that while doing compression **SuperDuper** always fully uses the CPU. Even moving the mouse can slow down the operation in progress. Anyway, if you have all of your memory allocated for the buffer, it is definitely not a good idea to do anything besides waiting for the copy to finish.

A little side-effect of the allocation of all of the available RAM is that some requester could be turned into an alert, or could even disappear without waiting for the user to acknowledge it.

**HDBUF** creates an 880K file in the current directory of **SuperDuper**, and uses this file as a buffer, exactly like **BUFFER** does with RAM. Of course you must use it only if you have a hard disk, and you started **SuperDuper** from it. The file contains the 1760 blocks which form a disk in their natural order. The **READ** operation will be a little slower, but if you have a good hard disk you should be able to make copies as fast as with a RAM buffer. The file is named '**SD\_Buffer**', and it's accessed only during the copy operations. This means that you can read or write it using the CLI commands, or the Workbench (but you will have to supply an icon). You can easily write an **ARexx** macro which retrieves/stores binary images of a disk from/to '**SD\_Buffer**' (see Chapter 4 [ARexx], page 13). Then **SuperDuper** will use the new contents when writing to floppies.

If you put a file named '**SD\_Buffer**' in **SuperDuper**'s directory *before* clicking **HDBUF**, then **SuperDuper** will assume this is a buffer file and will use it. You can even write directly to floppies without reading anything. Note that usually the buffer file is deleted when the **HDBUF** gadget is deselected, but if you supply a buffer file before activating the gadget your file will be left untouched.

**VDBUF** is probably **SuperDuper**'s most esoteric feature. By typing a device name in the string gadget named **VDNAME**, you can select any device (**SuperDuper** needs the Exec device name, e.g., '**ramdrive.device**' for the RAD: AmigaDOS device). The unit number is taken from the gadget with the label **VDUNIT#**. The device you specified will be used as a buffer for your disks. **SuperDuper** expects the device to behave like the

trackdisk.device, namely it must be able to write data at specific offsets. The main devices you can use, with their respective names, are:

RAD: the recoverable RAM drive. Configure it in your mountlist as a floppy, and you can use it as a buffer (Exec name: `'ramdrive.device'`).

FMS: Matt Dillon/Jim Cooper's virtual floppy-on-hard disk (Exec name: `'fmsdisk.device'`).  
VD0:, etc. other recoverable, sector-oriented RAM drives.

The device you specify is checked on opening to see if it has enough space to contain a full disk. The check is done simply on the number of sectors available—if there are enough sectors, and they are arranged differently than on a floppy, you will be able to use the device as a buffer, but don't expect AmigaDOS to get anything meaningful from it.

**Warning:** many of these devices are buggy and return *no error* on unsuccessful opening or failed size test. Some of them in this case will trash your memory. Be sure that the device is configured properly—try an AmigaDOS command on it first.

Of course, many people will find incredible ways to use this feature (for techies: if you want try something weird, consider that **SuperDuper** reads 512 bytes at offset 900608 on opening to test for size, and then reads 1760 chunks of 512 bytes, one for each sector, for every copy. The sectors are read sequentially as they are distributed on the disk, so if the device ignores the offset indication, you can feed it with 880K of a continuous bytes stream. Buffering is another story though—the offset indication is important because **SuperDuper** places the blocks on the device “in the right place” as soon as it encounters them).

A BUFFER is considered non-valid as soon as allocated, because it will contain random info. To make it valid, you must read in a floppy. VDBUF and HDBUF instead assume the buffer is always valid, because it could be externally fed. This mechanism allows you to prepare, for instance, a distribution disk at high speed in RAD: or in your hard disk using FMS:, and then to copy it to floppies directly.

In the same vein, **SuperDuper** will act slightly differently when determining if a buffer contains a DOS disk (if not, the incrementing of the date is inhibited even if selected). At read time, the information is recorded, but if at write time the pass starts from track 0, **SuperDuper** will re-fetch the DOS mark from the buffer and check it again. This way if for instance you externally feed a ramdrive.device with a diskcopy command **SuperDuper** will be aware of it and will increment the date if requested to do so.

Some care must be taken in order to obtain what you really want when mixing the buffering features and the selection of the start/end cylinder. **SuperDuper** implements a reasonable mean of flexibility and reliability for these kinds of operations.

When using VDBUF or HDBUF, the read/write operations start and end exactly where you specify with the start/end cylinder gadgets. Since **SuperDuper** has no control over what you do to the virtual disk while it's not accessing it, it has to assume you made it right.

When using a RAM buffer, **SuperDuper** can clearly make some assumptions on its validity. In particular, just after allocation or a stopped READ it assumes the buffer is not valid.

If you have a valid buffer and you change the start/end cylinders, there are two cases: either the buffer range and the start/end range do not intersect, in which case an error message is issued if you try to write the buffer, or there is a non-empty intersection, in which case the intersection will be written, i.e., the starting track will be the greatest of the start of the buffer and the start cylinder, while the ending track will be the least of the end of the buffer and the end cylinder. Example: if you read something with SC=20, EC=30, then you set SC=10, EC=25 and hit GO, the range 20–25 will be written.

There are however two subtle differences between the behaviour of a complete (880K) RAM buffer and a partial one. First of all, the track range chosen for reading in a complete RAM buffer is always the full start/end cylinder range, while if reading in a partial buffer **SuperDuper** will start from the last track of the previous buffer (of course, if the last track is past the end cylinder, it will start from the start cylinder). Moreover, if a long range of tracks is skipped (for instance, you read in a buffer range of 0-79 and you write 70-79) a few (less than 10) seconds will pass while **SuperDuper** unpacks the data you don't want to write—they have to be decompressed anyway.

If all this scares you, don't fear: the buffer/range interaction will simply work just as you intuitively expect. I hope at least 8^).

## 2.4 The User Interface Gadgets

- |      |  |
|------|--|
| TALK | activates <b>SuperDuper</b> 's ability to give its status by voice. Currently only English is supported.   |
| AUTO | activates automatic operation starting. <b>SuperDuper</b> will monitor disk insertion and ejection. When all destination(s) have been ejected and re-inserted, a GO operation is started. If FORMAT is selected, the destination(s) are formatted. Else, if a buffer is selected, it is written to the destination(s). If neither formatting nor buffering is requested, <b>SuperDuper</b> will monitor the source, too, and will start a disk-to-disk(s) copy as soon as the source and all destination(s) have been ejected and re-inserted. |
- Warning:** especially on one-drive-only systems, AUTO can be extremely dangerous. You'd better write-protect your source disks.

KILLSYS

RESTORE closes the Workbench and voice, flushes the memory and opens a very small screen with only two colors. Moreover, the window is of `SIMPLE_REFRESH` type rather than `SMART_REFRESH`. This way, the maximum amount of memory for your system is at your disposal (unfortunately, under 1.3 the window can be refreshed incorrectly because of an Intuition bug). If the Workbench can't be closed for some reason, a warning is issued (usually some application has a window opened on the Workbench screen). When you want to get back, hit the gadget again (this time it will be named `RESTORE`). This feature is very powerful if coupled with `SDBootInstall` and with the CLI option `'LOWMEM'`.

**Warning:** If you grab the `disk.resource` (by selecting a source and/or a destination) just after a disk was inserted, it's likely the Workbench will be locked, waiting for you to unlock the drive in order to load the icon of the disk. If in this moment you hit `KILLSYS`, you will lock the entire system, since `SuperDuper` will be waiting for the Workbench to close, while the Workbench will be waiting for you to release the disk.

## 2.5 The String Gadgets

The string gadgets have been more or less discussed in the previous sections. They are gathered here for sake of clarity.

SC

EC select the start and the end cylinders, respectively, for any operation.

LABEL lets you choose a name for the disks formatted by `SuperDuper`. See Section 2.2 [The Copy Options], page 6, for the effect of the `INCNAME` gadget.

VDNAME

VDUNIT# select the name and the unit number of the Exec device that `SuperDuper` will use as a virtual disk if the `VDBUF` gadget is selected.

RETRY# selects the number of read/verify retries on each track.

## 3 Special requesters

When `SuperDuper` needs to inform the user about something, usually a requester with a message appears (if the `TALK` option is on the message is also read out loud). While most of the requesters are self-explanatory, some of them need a more detailed description.

`'Can't get disk.resource'`

The `disk.resource` is the Exec way of controlling the access to the low-level disk hardware. **SuperDuper** can't access the resource, probably because someone is already using it. If you suspect a particular program, close it and try again to select a disk gadget.

`'Please free disk.resource'`

(See also previous requester). If the `disk.resource` can't be grabbed, Exec won't give back the message passed by **SuperDuper** until the resource is free. Thus, until that moment **SuperDuper** can't exit.

`'Checksum error: buffer munged.'`

Someone wrote over **SuperDuper's** RAM buffer. The buffer is no longer valid, and the current copy is probably munged, too. You should probably reboot, because if something writes on someone else's memory it's likely it will do it again.

`'A track buffer has been munged.'`

Someone wrote on one of **SuperDuper's** track buffers. The same comments of the previous requester apply.

`'ARexx server not active'`

In order to use ARexx macros, the ARexx server has to be activated. Type `RexxMast` at a CLI prompt (if it's not in your path, you should locate it easily).

`'Error while recalibrating unit x.'`

**SuperDuper** found an error while recalibrating a drive head. The head was moved to track 0, but the drive signal `'DSKTRACK0'` wasn't activated. This means that either your drive has lazy signals, in which case there's nothing to worry about, or that some head step wasn't actually performed (possibly because of power supply reasons) in which case the last copy could be bad, even if `VERIFY` is on. Better `CHECK` it. Try also to increase the step and calibrate delays of the drive with `SetTDDelay`.

`'Better write-protect your sources.'`

This message is issued every time you select the AUTO gadget on a machine with a single drive (see Section 2.4 [The User Interface Gadgets], page 10).

## 4 ARexx

ARexx is the system macro language of the Amiga. It was originally developed by Bill Hawes (to whom every Amiga owner owes much more than he probably realizes) and was then included in the Release 2 of the operating system.

ARexx is a beautiful interpreted language, with unique features such as syntax/semantics collapsing (for instance, you can ask the value of a variable given its name as a string) and, overall, the ability to interface itself with external applications. A single ARexx script can control several different programs and make them interact.

The ARexx interface consists of a port, which is used for communications, and a set of commands that ARexx can issue to the application. For **SuperDuper**, the port name is **SUPERDUPER**, and the command set is described below. ARexx scripts written for **SuperDuper** should have extension ‘**supdup**’, like ‘**foobar.supdup**’. This is in order to distinguish ARexx scripts written for different applications.

ARexx provides at little or no implementation cost a powerful macro language which substantially increases the performance and the versatility of an application. Maybe some feature you would like to have is not in **SuperDuper** at this time, but it’s very likely you’ll be able to put it in via a suitable ARexx script.

### 4.1 General Issues

Besides being able to execute commands issued by an ARexx macro, **SuperDuper** is also able to start an ARexx macro. This is indeed the purpose of the AREXX gadget (the last one in the last row). The gadget is activated if 1) the ‘**rexsyslib.library**’ is somewhere in your LIBS: directory and 2) you have a file requester. **SuperDuper** is able to recognize and use the ASL file requester (under Release 2), the ARP file requester or the req.library file requester (the first available in this order will be used). You can start any number of macros at the same time (beware of wild interactions though).

**SuperDuper** commands generally correspond to gadgets, and are similarly named: for instance, the command **check** will check the source drive, while **vdunit 4** will set the virtual disk buffer unit

number to 4. Commands are case insensitive, and only the first two or three letters are significant. So you can write `ch` instead of `check` but you have to write `rea` for `read`, or you could make confusion with `restore` or `retry`.

ARexx needs a console by which it communicates with the user. If you started **SuperDuper** from the CLI, the your original CLI will be used. Otherwise, a console window will be opened. Under 1.3, this window appears at the start of any ARexx macro and gets closed when there is no macro running. Under Release 2 it's always open, but it's an **AUTO** console window, so you can close it if you wish: it will be reopened as soon as something is printed into it.

## 4.2 Action Commands

The commands `go`, `read`, `check` and `stop` act just like their gadget counterparts, starting a copy (buffering, formatting) process or stopping it. The first three return at the end of the operation. However, for instance, if another task sends a `stop` command while a copy is in progress, the copy is interrupted and the `go` command returns. You can then check the `rc` variable to see what happened (see *The ARexx User's Reference Manual*).

The pair `killsys` and `restore` work like the corresponding gadget. The operations which are nonsense have no effect (i.e., if you send `killsys` and the system has already been killed, nothing happens).

Finally, `rx string` executes the ARexx macro named *string*.

## 4.3 Selection Commands

I list here for sake of completeness the whole group of selection commands. They could be easily deduced anyway from the gadget names, apart from `noclick`, `volume`, `printererrors` and `rx` which are available only through the ARexx interface. Here *string* is a string of characters and *n* is a nonnegative number. When `on/off` is specified as an argument, you have two ways of invoking the command: *command on* will switch the thing on, and *command off* will switch it off. Note that the ARexx interface of **SuperDuper** is rather lazy about syntax—strings too long will be silently truncated, and passing a non-numerical argument where *n* is required will usually produce a value of 0.

**source *n***    Selects drive *n* as source;

**source off**  
Turns off source drive;

**dest *n*** Selects destinations using *n* as a bit mask. For instance, 0 selects no drive, 1 selects drive 0, 5 selects drives 0 and 2, 15 selects all destinations;

**buffer on/off**  
Controls the RAM buffer;

**hdbuf on/off**  
Controls the hard disk image file buffer;

**vdbuf on/off**  
Controls the virtual disk buffer;

**verify on/off**  
Turns on/off verify;

**date on/off**  
Turns on/off date adaptation;

**incname on/off**  
Turns on/off name increment while formatting;

**ffs on/off**  
Selects Fast File System or Old File System while formatting;

**talk on/off**  
Toggles talk mode;

**auto on/off**  
Toggles auto mode;

**label *string***  
Sets the disk label

**retry *n*** Sets the number of retries;

**vdunit *n*** Sets the virtual disk unit number;

**vdname *string***  
Sets the virtual disk unit device name;

**scyl *n*** Sets the start cylinder;

**ecyl *n*** Sets the end cylinder.

**quit** Quits the program.

The following commands are only available through the ARexx interface:

**noclick *n*** Forces SuperDuper to not click the drives specified by *n* as a bit mask (the same format of **dest**).

**printererrors on/off**

Opens/closes **SuperDuper**'s detailed error report window;

**requesters on/off**

Inhibits the error requesters from popping up (mainly useful when **SuperDuper** is in its no-GUI mode; see Section 5.2 [The CLI Interface], page 19).

**volume n** Sets the volume of the beeps ( $0 \leq n \leq 64$ );

## 4.4 Return Codes

Commands issued by ARexx to an application should return useful values in order to tell what really happened. Generally, a command which fails returns an error level, while a successful command returns an error level of zero and, upon request of the caller via the **OPTIONS RESULTS** command, a result string which can be parsed in order to get useful information.

**SuperDuper** returns an error code of 10 if the syntax of the command was wrong. This will cause ARexx to complain with an error message. An error code of 1 is returned if the syntax was right but the command couldn't be executed, but there is no real failure (for instance, if you send **go** while a copy is already in progress or if you try to select a ghosted gadget). An error of 30 is returned in extreme cases, for instance when you hit the close gadget and there are still some commands pending. No strings are ever returned, since we have only a few cases to differentiate. Return codes with special meanings are returned by the following commands:

**source**

**dest**

- |   |                                      |
|---|--------------------------------------|
| 2 | The selected drive is not connected. |
| 5 | The disk.resource is not available.  |

**buffer**

**hdbuf**

**vdbuf**

- |   |                                 |
|---|---------------------------------|
| 5 | The buffer cannot be allocated. |
|---|---------------------------------|

**buffer**

- |   |  |
|---|--|
| 2 | A full buffer cannot be allocated. Compression is on, and there is the possibility of multi-pass copies. |
|---|--|

**go**

**read**

**check**

2	This pass is not the last one.
3	Something is wrong with the chosen source, destination and buffer options. For instance, you're trying to copy from df0: to df0: without a buffer.
4	The buffer is not valid.
5	A unit is empty.
6	A unit is write-protected.
7	The start/end cylinders chosen are meaningless. This can happen if the numbers are out of range, or (for a RAM-buffered GO) if there is no inter- section with the current buffer.
8	There were errors.
9	There were errors. Moreover, this pass is not the last one.
20	Someone munged the RAM buffer or the track buffer.
talk	
5	The voice system cannot be activated.
killsys	
restore	
20	The current window has been closed, but it was impossible to open the new one. The program exits in this case.

## 4.5 What Can I Do with ARexx?

Basically you can expand **SuperDuper**'s capabilities and/or make it interact with other programs. A couple of examples of the first case could be a '**CheckAll.supdup**' macro which checks all drives in sequence. The "native" **SuperDuper** can only check one drive at a time, but if you have two or more drives you can check many drives using a macro like

```
/* CheckAll */
do i = 0 to 4
  source i
  if rc==0 then check
end
```

After checking you should of course look at the return codes in the **rc** variable and decide upon appropriate actions.

Suppose now you have four drives and you want to make a copy of two different floppies. You can put the sources in drives 0 and 1, the destinations in drives 2 and 3, and then

```
/* DoubleCopy */
source 0
dest 4
go
source 1
dest 8
go
```

(of course I'm assuming **SuperDuper** is in its default configuration). This will produce the two copies in a completely unattended way.

## 5 The CLI Interface

### 5.1 Line Command Options

When you start **SuperDuper** from the CLI, you have the chance to specify an option. The possible options are printed in the standard Amiga template format if you type **SD ?**. In this case, the following line

```
PUBSCREEN/K,NOGUI/S,LOWMEM/S
```

will be displayed. Its meaning is that **NOGUI** and **LOWMEM** are switches that you can activate, while **PUBSCREEN** must be followed by the name of an existing public screen. For instance, the command line **SD NOGUI** will invoke **SuperDuper** in its **NOGUI** mode. The two flags **NOGUI** and **LOWMEM** are mutually exclusive—if both are specified, only the first one counts.

#### PUBSCREEN

**SuperDuper** will open its window on the specified public screen. This parameter is non functional under 1.3.

#### NOGUI

**SuperDuper** won't open its graphical user interface but you can then control it through the **ARexx** interface. This makes possible to write an **ARexx** macro allowing you to use **SuperDuper** from the shell much as the **diskcopy** command. Moreover, the startup file '**Startup.supdup**' is not executed, so that in your **ARexx** macro which calls **SuperDuper** directly you can expect to get the standard configuration.

#### LOWMEM

This switch shuts down the **ARexx** port and the sound system. **SuperDuper** won't open either the **ARexx** port/**rexsyslib.library** pair, or the **audio.device**. This mode is

provided for user with 1M or less who want to have as much free memory as possible (read also the section about `SDBootInstall`).

These options are only available from the CLI.

## 5.2 The Startup File

At startup time, **SuperDuper** checks if **ARexx** is available, and in this case it tries to start an **ARexx** macro named `'Startup.supdup'`. This file should contain your usual settings: for instance, it's a very good place where to put a `volume` command. The startup file is a regular **ARexx** macro, just like any other one started by the **AREXX** gadget or by the `rx` command. However, a couple of conventions were implemented in order to get a better behaviour on systems without **ARexx**. In particular, the absence of the **ARexx** server or the **ARexx** error message `'Program not found'` will *not* be displayed if caused by the startup file. Notice that the last message can also be caused by the first line of `'Startup.supdup'` not being a comment (every **ARexx** macro must start with a comment).

# 6 Performance

## 6.1 SuperDuper and Your System

**SuperDuper** has been written keeping in mind that a good program doesn't have to eliminate everything from the system in order to work. The Amiga has a very efficient multitasking kernel which allows for resource arbitration.

When **SuperDuper** is started, it won't allocate anything from your system. As soon as a source/destination gadget is clicked, it will inhibit all of the drives (so don't select a gadget while reading or writing to floppies) and then will grab the `disk.resource`. Until the resource is released, *no one else* can access the Amiga drives. This is necessary in order to avoid unpredictable collisions with the system or other programs. Inhibiting the drives is not enough, since some other file system (like **CrossDOS**) could access them.

If you need to temporarily access your drives, you must simply deselect all **SuperDuper** source/destination gadgets: the disk system will be restarted (it will be re-grabbed on a gadget selection of course). The heads will be moved to their original position, so that you no longer need to eject the drives under 1.3.

The CPU use of **SuperDuper** is almost unnoticeable. You can do anything else, and you shouldn't notice any slowdown. In particular, if no source/destination is selected **SuperDuper** is completely asleep.

This however is not true if you use compression. In this case, not only will the system be slowed down (a priority 0 task will almost always be active), but *any* operation (including moving the mouse pointer) will slow down **SuperDuper**.

If you use the utility **ToggleClick** distributed with **SuperDuper** (or any other utility which legally kills drive clicks under Release 2) **SuperDuper** won't click empty drives (drive clicking is necessary for monitoring disk insertion; using **ToggleClick** is good but you must be sure your drives won't try to move past track 0 if asked to do so). Anyway, you always have the chance to selectively force **NOCLICK** via the corresponding **ARexx** command.

You should avoid running **SuperDuper** while a 16 color hi-res screen (or a 4-color ECS productivity mode screen) is displayed. The video DMA access will interfere with the disk/CPU/Blitter access to the point that copy times will rise to incredible values—reading and compressing a disk in the buffer can take more than 100s.

## 6.2 SuperDuper and You

“Well,” you could say, “**SuperDuper** is a great copier—but how can I trust it for making my copies? This guy diddles with hardware—maybe I should use the system **DiskCopy** command.”

This is not a good idea. First of all, **SuperDuper** is *incredibly* picky about verifying. You will get more verify error messages than with the standard copy commands (for techies: **SuperDuper** verifies also the MFM timing bits, not only the data bits; this means a 200% efficiency improvement in catching verify errors and generally bad media).

Moreover, both the 1.3 and the 2.0 trackdisk.device have unpleasant side-effects on frequently read/written tracks. These side-effects are cleared when you do a copy of the disk with **SuperDuper** (for techies: trackdisk.device doesn't check for MFM bits being read in correctly, and doesn't re-MFM the track before writing it; it just re-MFMs the changed sector. If a MFM timing bit is read wrong, it will stay wrong forever, possibly causing read errors; but **SuperDuper** re-MFMs every track it copies, thus restoring every MFM timing bit to its correct value).

Finally, if you don't like coffee-breaks during your copies, you'd better use the fastest copier available—namely **SuperDuper**. Note that if you have four drives and you use top-quality disks, so

you can skip verify, the buffer system allows you to get a per-copy time of 9 1/2 seconds, which is definitely not bad.

### 6.3 SDBootInstall

When your system boots up (at power on or after a reset), the operating system searches for available drives, and creates some trackdisk.device tasks accordingly. These tasks take a lot of memory for their buffers (>30K), but **SuperDuper** doesn't use them at all, because it has its internal routines.

If you have to do intensive copy work, and you have 1MB of memory or less, you could find it useful to boot up your system in a special configuration that will shut down almost all trackdisk.device tasks, thus freeing a lot of memory.

To accomplish this, do as follows:

1. Make a copy of your usual Workbench 1.2 (or greater) disk (from now on we work on the copy).
2. Delete some programs to make room—**preferences**, **diskcopy** and **format** are good candidates. Moreover, delete the file **'Disk.info'**.
3. Copy **SuperDuper** to the disk root directory (by dragging its icon on the disk icon or using the CLI).
4. Edit the startup-sequence of the disk (it's in the **'s'** directory). Delete it entirely, and substitute it with

```
SetPatch >NIL:
Run >NIL: <NIL: SD LOWMEM
EndCLI >NIL:
```

If you're under 1.2, don't put in the first line (you don't have a **SetPatch** command).

5. Now put the disk in df0:, and run the utility **SDBootInstall**. A special bootblock will be installed on the floppy. When booting from it, the operating system (and you) will be able to access only drive 0—the other ones will be for **SuperDuper**'s use only. To get back to normality, a reboot is necessary. You will gain 30/40K per drive using this method (for techies: it is perfectly legal—the bootblock simply **AllocUnit()**s the drives with ID>0).

## 6.4 A Word on Copy Protection

**SuperDuper** won't copy protected disks (or if it will it's just a coincidence). I do not believe in copy protection. Scrambled tracks will produce random data on the destination. If the read error goes beyond a simple checksum error don't expect anything meaningful to be written on the destination disks.

However, **SuperDuper** will faithfully reproduce data block checksum errors ('Disk foobar has a read/write error') or DOS checksum errors ('Key 880 checksum error') on the source disk in disk-to-disk copies (header checksum errors are fixed when renumbering the sectors). Thus, if you got the typical 'Key <n> checksum error' you can make a copy of the disk before fixing it. **SuperDuper** won't do any surgery: use a good tool (such as **DiskSalv** or **FixDisk**) for that. Avoid **DiskDoctor**. On the other hand, during buffered copies data block checksums will be silently fixed by recalculating the right checksum.

## 7 Acknowledgments

The first person I must thank a thousand times is Dirk Reisig. It was by means of his suggestions that I sped up **SuperDuper** to the current, amazing level. I wrote him a letter which he answered gently with a long explanation of the optimizations performed by **PCopy**. The first time I read the letter it seemed greek to me, but little by little I learned all the mysteries of MFM encoding and disk direct hardware driving. Moreover, I learned from the source code of **TrackSalve** the usage of the blitter for MFM encoding and many other subtle things. In other words, without the help of Dirk you would have never seen anything after **DFC5** (for version 2.0, a new optimization was introduced; it was suggested by Dan Babcock).

The second guy behind the birth of **SuperDuper** is Tom Rokicki. He pushed me to write a substitute for **TurboBackup**, and overall suggested the main thing—that on the Amiga it is possible to write many disks at the same time. Without this trick, you could never do four non-verified copies in 38s. Tom also tested all pre-whatever-greek-letter versions, always giving useful comments... and risking the life of his drives 8^). Moreover, I had time to work on **SuperDuper** because the Amiga $\text{\TeX}$  system is so incredibly efficient I got a lot of spare time while writing math papers...

Last but not least, Randell Jesup at Commodore drove me through the labyrinth of non-specified-specs, hardware quirks, strange behaviors, and system esoteric features. Without his help **SuperDuper** could probably work... but I wouldn't trust it for *my* copies 8^).

The name **SuperDuper** popped up during a rather intensive BIX discussion. Many other names were proposed, but in the end I chose this one—it has symmetry, correctly defines the product and has a simple shortening (SD). Thus, a thousand thanks to Kent Kalnasy and Dan Barrans for suggesting this name.

Many features were not my ideas. An incredible number of BIX users came up with excellent suggestions, many of which were actually implemented. Thanks to them you have support for buffering on any device (I never use RAD: nor FMS:, so I didn't think it could be useful).

But, as always, the biggest *thanks* goes to the beta-testers of **SuperDuper**: Dennis Atkin, Michele Battilana, Vittorio Calzolari, Jim Cooper, Doug Erdely, Charlie Fair, Blaine Gardner, Robert Jenks, John Jones, Kent Kalnasy, Robert Kesterson, Paul King, Randy Menzer, Linda Munson, Davide Repetto, Tom Rokicki, Sergio Ruocco, Carlo Santagostino, Reinhard Spisser, Jeff Todd, Carlo Todeschini, Michael Scott Velez and Marco Zandonadi. Beta-testing a copier is different from anything else—if it doesn't work you won't get a marginally corrupted picture on your display: rather, the Fish Disks it took an hour to copy could be unusable. A special kind of patience is needed under these conditions 8^).

## 8 Disclaimer and Author Info

**SuperDuper** is Copyright © 1991,1992 Sebastiano Vigna and it's freely distributable as long as all of its files are included in their original form without additions, deletions, or modifications of any kind, and only a nominal fee is charged for its distribution. This software is provided **AS IS** without warranty of any kind, either expressed or implied. By using **SuperDuper**, you agree to accept the entire risk as to the quality and performance of the program; don't come to me if you destroy your entire Fish Disk library with it! Of course, it was tested rather extensively before it was released...

Comments, complaints, desiderata are welcome.

Sebastiano Vigna  
Via California 22  
I-20144 Milano MI

BIX: svigna  
INTERNET: vigna@imiucca.csi.unimi.it  
          vigna@ghost.sm.dsi.unimi.it  
UUCP:cbmehq!cbmita!sebamiga!seba@cbmvax.cbm.commodore.com  
      ...{uunet|pyramid|rutgers}!cbmvax!cbmehq!cbmita!sebamiga!seba  
FIDO: 2:332/607.28

# Concept Index

<	Errors .....	4
'<BAD_NAME>' .....	Errors while recalibrating .....	1
'<NDOS>' .....		
'<UNKNOWN>' .....		
<b>A</b>	<b>F</b>	
Acknowledgments .....	Fair Charlie .....	22
Address .....	Features .....	2
ARexx .....	First Steps .....	4
Atkin Dennis .....	Flakey drives .....	1
	FMS: .....	7
	French keyboard .....	1
<b>B</b>	<b>G</b>	
Babcock Dan .....	Gardner Blaine .....	22
Bar color .....		
Barrans Dan .....	<b>I</b>	
Battilana Michele .....	Introduction .....	1
Buffer File .....		
Buffering .....	<b>J</b>	
	Jenks Robert .....	22
<b>C</b>	Jesup Randell .....	22
Calzolari Vittorio .....	Jones John .....	22
Changes .....		
CLI Options .....	<b>K</b>	
Cooper Jim .....	Kalnasy Kent .....	22
Copy protection .....	Kesterson Robert .....	22
Copying .....	Keyboard Usage .....	4
CPU usage .....	King Paul .....	22
	<b>L</b>	
<b>D</b>	Low memory .....	18
Disclaimer .....		
Distribution .....	<b>M</b>	
DMA contention .....	Menzer Randy .....	22
Drive inhibition .....	Munson Linda .....	22
	<b>N</b>	
<b>E</b>	No clicks .....	1, 14
E_mail .....	NoGUI .....	18
Erdely Doug .....		
Error Report .....		
Error reproduction .....		

**P**

Performance .....	19
Public Screen .....	18

**R**

RAD: .....	7
Reisig Dirk .....	22
Repetto Davide .....	22
Requesters .....	11
Retries .....	4
Return codes .....	16
Rokicki Tom .....	22
Ruocco Sergio .....	22

**S**

Santagostino Carlo .....	22
Simple Refresh .....	10
Smart Refresh .....	10
Spisser Reinhard .....	22

Startup File .....	18
--------------------	----

**T**

Timing bits .....	20
Timings .....	1
Todd Jeff .....	22
Todeschini Carlo .....	22

**V**

VD0: .....	7
Velez Michael Scott .....	22
Voice .....	10
Volume Control .....	14

**Y**

You .....	20
-----------	----

**Z**

Zandonadi Marco .....	22
-----------------------	----

**Gadget Index****A**

AUTO .....	10
------------	----

**B**

BUFFER .....	7
--------------	---

**C**

CHECK .....	4
-------------	---

**D**

DATE .....	6
------------	---

**E**

EC .....	4, 7, 11
----------	----------

**F**

FFS .....	6
-----------	---

FORMAT .....	6
--------------	---

**G**

GO .....	4
----------	---

**H**

HDBUF .....	7
-------------	---

**I**

INCNAME .....	6
---------------	---

**K**

KILLSYS .....	7, 10
---------------	-------

**L**

LABEL: .....	6
--------------	---

**R**

READ..... 4  
 RESTORE..... 10  
 RETRY#..... 4  
 RTRY:ERR..... 4

**S**

SC..... 4, 7, 11  
 STOP..... 4

**T**

TALK..... 10, 11

**V**

VDBUF..... 7  
 VDNAM..... 7, 11  
 VDUNIT#..... 7, 11  
 VERIFY..... 6

**ARexx Command Index****A**

auto..... 14

**B**

buffer..... 14

**C**

check..... 14

**D**

date..... 14  
 dest..... 14

**E**

ecyl..... 14

**F**

ffs..... 14

**G**

go..... 14

**H**

hdbuf..... 14

**I**

incname..... 14

**K**

killsys..... 14

**L**

label..... 14

**N**

noclick..... 14

**P**

prnterrors..... 14

**Q**

quit..... 14

**R**

rc..... 16  
 read..... 14  
 requesters..... 14  
 restore..... 14  
 retry..... 14  
 rx..... 14

**S**

scyl..... 14  
 source..... 14  
 stop..... 14

**T**

talk..... 14

**V**

vdbuf ..... 14

vdname ..... 14

vdunit ..... 14

verify ..... 14

volume ..... 14

## Program Index

**A**

AmigaTeX..... 22

**C**

CheckAll.supdup ..... 17

**D**

DiskCopy..... 20

DoubleCopy.supdup..... 17

**S**

SDBootInstall ..... 21

SetTDDelay..... 1

Startup.supdup ..... 18

**T**

ToggleClick..... 1, 19

TurboBackup..... 22

# Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>1</b>
1.1	Changes .....	1
1.2	Main Features .....	2
1.3	First Steps.....	4
<b>2</b>	<b>Gadgets .....</b>	<b>4</b>
2.1	The Action Gadgets.....	4
2.2	The Copy Options .....	6
2.3	The Buffering System .....	7
2.4	The User Interface Gadgets .....	10
2.5	The String Gadgets .....	11
<b>3</b>	<b>Special requesters .....</b>	<b>11</b>
<b>4</b>	<b>ARexx .....</b>	<b>13</b>
4.1	General Issues .....	13
4.2	Action Commands .....	14
4.3	Selection Commands .....	14
4.4	Return Codes.....	16
4.5	What Can I Do with ARexx? .....	17
<b>5</b>	<b>The CLI Interface.....</b>	<b>18</b>
5.1	Line Command Options.....	18
5.2	The Startup File.....	19
<b>6</b>	<b>Performance .....</b>	<b>19</b>
6.1	SuperDuper and Your System.....	19
6.2	SuperDuper and You.....	20
6.3	SDBootInstall .....	21
6.4	A Word on Copy Protection.....	22
<b>7</b>	<b>Acknowledgments .....</b>	<b>22</b>
<b>8</b>	<b>Disclaimer and Author Info .....</b>	<b>23</b>

<b>Concept Index .....</b>	<b>24</b>
<b>Gadget Index.....</b>	<b>25</b>
<b>ARexx Command Index.....</b>	<b>26</b>
<b>Program Index.....</b>	<b>27</b>