Format64

COLLABORATORS

	TITLE : Format64	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 7, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Format64

1.1 Format64 user's guide

```
                  Format 64 v1.03 -- A format replacement for drives Larger than 4 ↩
                     GB.
  ------------------------------------------------------------------
  This work is based on the
                 NewFormat v1.00
                 by Dave Schreiber

   Disclaimer       I'm not responsible.
   Story            Why was this program written.
   NewFormat        Original documentation.

   Usage            How to use.
   Requirements     What you need in order to use this.
   History          History list.

   About NSD        Info about New Style Devices.

  ------------------------------------------------------------------

  NewFormat v1.00 is copyrighted (c) 1992 by Dave Schreiber
  Format64 v1.03 is copyrighted (c) 1998 by Jarkko Vatjus-Anttila

  All rights reserved.

  This program may not be sold for more than a small copying and
  shipping and handling fee, except by written permission of Jarkko
  Vatjus-Anttila.
```

1.2 disclaimer

```
  Disclaimer
  ----------

THERE  IS  NO WARRANTY FOR THE PROGRAMS, TO THE EXTENT PERMITTED BY APPLICABLE
```

LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAMS "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAMS IS WITH YOU. SHOULD THE PROGRAMS PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY REDISTRIBUTE THE PROGRAMS AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAMS (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAMS TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1.3 story

Story

Why was Format64 written??

Well, I just recently bought Seagate Medalist 6.4GB harddisk and properly installed the new scsi.device and FastFileSystem to make sure that I could use all of the 6.4GB disk. I was aware of Amiga's stupid 4GB HD limit.

The funniest thing was that I didn't find any Format program that could utilize the scsi.device to format all of my disk. This requires a proper handling of the 64bit commands but it really seemed to be that no such programs exist. At this point I thought that I had to write my own.

Fortunately Dave Schreiber had made a program called NewFormat which sources were included in the package. It was no job at all to re-engineer the sources to make them to use the 64bit command set and to format a drive beyond 4GB border.

I studied the information given in the Amiga Develoer CD and eventually I managed to make a program with the 64bit support. That's when Format64 was born. Only this time all of the valuable hints in the Developer CD were taken into mind and Format64 was written in a way that it works in the future too.

Thank you Amiga for being the best computer ever.

- Jarkko Vatjus-Anttila

1.4 newformat

This is the old document file of the NewFormat v1.00

(New)Format
August 31, 1992
Written by Dave Schreiber

This program is a replacement for the AmigaDOS Format command. For the Workbench user, it sports a more friendly interface, and allows greater control over formatting options.

First, a disclaimer:

The purpose of this program is to erase the contents of floppy disks, hard drives, and other forms of temporary and permanent computer storage. The author, Dave Schreiber, will not be held responsible for any data lost through the (correct or incorrect) use of this program, nor will he be held responsible for any damages (financial or otherwise) resulting from the use of this program.

These instructions don't go into what formatting a disk means, why you needs to do it, the options available, etc. For this information, I'd suggest reading the manual that came with you Amiga or your Workbench 2.04 (or later) upgrade.

CLI Usage:

From the CLI, this program is almost identical to the AmigaDOS Format command. The only change is that a new switch, NOVERIFY, has been added. When given, this switch prevents NewFormat from verifying that each cylinder has been formatted correctly.

Workbench Usage:

There are one of two ways to use NewFormat from Workbench. The first is to select the drive or drives that you want to format, then double click on the NewFormat icon while holding down the shift key. You can also put NewFormat in the System directory on your Workbench disk or boot partition. This will cause Workbench to use it in place of the default Format command when you choose the "Format Disk..." item from the Icons menu.

Once you start NewFormat, you will notice that the Workbench interface is almost completely different from the interface of the old Format command. You are first presented with a window that lets you enter the new name of the disk and set every formatting option that you can enter from the CLI; the defaults are the same defaults as the old Format command uses. If you decide that you don't want to format the disk, you can choose the Cancel button (which will move you on to the next disk, if any), or the close gadget (which will quit the program). If you want to continue once you've chosen the name and other options, you can select the OK button. This will bring up a requester that asks if you're sure that you want to format the disk. If you answer "Yes" to this, the formatting procedure starts.

There are two ways that NewFormat keeps you informed of the status of the format while it is occurring. The first is a box that displayed the same "Formatting cylinder xxx, xxx to go" message as does the old Format command. Below that is a bar gauge that displays graphically how much of the format has been completed. A bar fills the box from left to right; when the box is full, the format is complete. When it is half full, it is half complete, etc.

At any time during the formatting procedure you may select the "Stop" button to abort the format.

That's it! If you have any questions or comments, please feel free to get in touch with me at the address below.

Dave Schreiber
1234 Collins Lane
San Jose, CA 95129
e-mail (until 6/19/93): davids@cats.ucsc.edu

1.5 usage

```
  Usage
  -----
```

Usage is compatible to original Format command. Only the obsolete switches like OFS/FFS are replaced with one single boolean switch. Well, usage is easy, take a look:

```
  DRIVE/K/A,NAME/K/A,FFS/S,INTL=INTERNATIONAL/S,NOICONS/S,QUICK/S,
  NOVERIFY/S

  DRIVE/K/A:  This is the device ID to be formatted. DH0: or DF0:
      or something like that. Must be always defined.

  NAME/K/A: This it the name of the device you want to give
      it after the format sequence. Must be always
      defined.

  FFS/S:    This is a boolean switch to tell Format64 whether
      to format an FFS disk or not. Default: FALSE.

  INTL/S:   This switch tells if the drive should be made
      international. Default: FALSE.

  NOICONS/S:  This tells if the trashcan should be created after
      format sequence. Default: TRUE.

  QUICK/S:  This tells should the sectors be formatted too or
      is the root block the only block that needs the
      operation. Quickformat can be cancelled afterwards
      with proper tools, but complete format cannot.
      Default: FALSE.

  NOVERIFY/S: Verify the format of not. If QUICK is defined, this is
      obsolete. New harddrives for example don't need this
```

```
      switch because very rarely you get a faulty disk.
      Well, better safe than sorry. Default: TRUE.
```

1.6 requirements

```
  Requirements
  ------------

  Format64 requires the following:

  - Amiga
  - KickStart 2.04+
  - mc68000
  - math libraries.
```

As you can see, basically Format64 doesn't require at all. The math libraries are requred because of StormC. I compiled the program for non FPU machines and because StormC startup code wants to open those libs, I cannot do anything to it.

```
NOTE: In order to use 64bit command set i.e in order to format a drive
  beyond 4G border you need to have:

  - The device that controls the drive you want to format must be
    compatible with 64bit command set. (Ex: scsi.device v43.23)
  - The filesystem on that drive must be familiar the the new commands
    too. (Ex: FastFileSystem 43.19)
```

My harddrive is Seagate Medalist 6.4GB and I use scsi.device 43.23 and FastFileSystem 43.19. they work just fine with Format64 and both of then can be found from the Net. Check out the main pages of Amiga. www.amiga.de for instance.

Format64 checks the device before operation and if the drive doesn't support 64bit commands, Format64 doesn't allow you to format beyond that 4GB border. This is just for preventing trouble from happening.

1.7 history

```
  History
  -------

v1.03 20th of April 1998
  - Format loop rewritten. I got strange bug reports about the format
    sequence. If this version does not fix the problems, then your
    drive controller doesn't work. (Well, keep the reports coming,
    though)
  - Format64 used 64bit commands with all >4GB drives although they
    should be used only with the partitions exceeding the 4GB border.
    Now fixed.
  - Removed some printf's to keep new format engine compatible with
```

```
    the GUI.

v1.02 12th of April 1998
  - Oh, brother. Recompiled the whole program.  International switch
    didn't work at all and some other trouble encountered too. Is
    now tested once again to ensure compatability.
  - This guide written.

v1.01 10th of April 1998
  - Replaced some custom routines with better working ANSI C ones.
  - Added INTL=INTERNATIONAL/S switch. This is usable only from
    CLI commandline though.
  - If a verify error occured, the program could end up in an infinite
    loop. Fixed.

v1.00 5th of April 1998
  - Initial release
```

1.8 nsd

```
  New Style Devices
  -----------------

  I decided to include the NSD information from the Developer CD in this
  guide. I think nobody will sue me because if it.

   NewStyleDevices
   Command Handling
   TrackDisk64

  Also check out the NSD.h include file for NSD devices:

   NSD.h
```

1.9 nsd1

```
What is this about?
===================

Up  to  and  including OS 3.1 (V40), custom device commands usually started at
CMD_NONSTD.   Each developer added commands for custom device features freely.
This  lead  to messy and incompatible devices and strange compatibility tricks
to identify device capabilities.

As  of January 1st, 1996, Amiga Technologies reserves two ranges in the set of
command  values  for future enhancements.  Only commands as specified by Amiga
Technologies  may  be  used  here.   It  is  illegal for a device developer to
randomly "add" custom commands or command features in the reserved are!

There are 65536 command values possible with io_Command:

    $0000 - $3fff       old style and 3rd party commands
```

```
    $4000 - $7fff       RESERVED AREA!
    $8000 - $bfff       old style and 3rd party commands
    $c000 - $ffff       RESERVED AREA!
```

To say it again: Commands in the reserved areas may only be assigned and specified by Amiga Technologies. Any "custom" implementation of these commands or other commands in these reserved areas is illegal and violates programming standards!

A device driver is REQUIRED to return IOERR_NOCMD on any command it does not understand. This requirement is very old. It has been documented publically in 1991 already.

Commands in the reserved ranges are called "new style". A device that supports new style commands correctly as described in this document is called a "new style device".

What will new style commands do for you?

With old style devices it is impossible to find out about the type of device you are accessing. You don't know if it is serial.device like or a trackdisk style device. New style commands will let you query the device for its capabilities. As capabilities are added to a device, device users can use more features transparently without compatibility hacks.

There are two types of new style commands. "General" commands are the same for all new style devices. Depending on the device type there are device specific commands, too.

General commands will be defined in the range $4000-$7fff, device specific commands are using the range $c000-$ffff. It is illegal to use a device specific command before using the general commands to confirm the device's capabilities.

With a new style device, you can be sure that no 3rd party command will interact with the standard meaning of all the device's commands as documented for V40 of the OS.

Basic requirements

Let's make a quick list about the most basic requirements for any new style device. Keep it in mind when reading on.

- IOERR_NOCMD support
- no redefinition of standard V40 or reserved commands
- no 3rd party commands in reserved areas
- NSCMD_DEVICEQUERY is supported
- lib_IdString must contain the name, version, and creation
  date of the device and any other readable information to make
  unique identification of a device for 3rd party
  command use possible.

Some devices may have additional requirements. These will be listed below.

General commands
================

At the moment there is just a single new style general command:

```
#define NSCMD_DEVICEQUERY   0x4000
```

It is required for all new style devices.

You set up io_Data and io_Length pointing to a struct NSDeviceQueryResult below. You must clear SizeAvailable and set up DevQueryFormat before sending the command to the device. A new style device will set up the data in the buffer appropriately. If the command executed successfully the returned io_Actual value must be greater than zero and equal to the SizeAvailable value that has been set by the device. The device may only be considered a new style device if these conditions are met. And only in this case the results of the query may be considered valid.

```
struct NSDeviceQueryResult
{
    /*
    ** Standard information
    */
    ULONG   DevQueryFormat;         /* this is type 0              */
    ULONG   SizeAvailable;          /* bytes available             */

    /*
    ** Common information (READ ONLY!)
    */
    UWORD   DeviceType;             /* what the device does        */
    UWORD   DeviceSubType;          /* depends on the main type    */
    UWORD   *SupportedCommands;     /* 0 terminated list of cmd's  */

    /* May be extended in the future! Check SizeAvailable! */
};
```

The device will fill in the struct NSDeviceQueryResult with read only data for the caller. All data must remain constant and valid as long as the device is open. After CloseDevice(), all bets are off.

SizeAvailable

> Tells you how much of the structure contains valid data. It is measured in bytes. Do not try to use fields in the strutcure that are not fully included in the SizeAvailable range. SizeAvailable must include SupportedCommands for any successful query. So the minimum valid value is 16.

DeviceType

> This tells you about the type of device you are accessing. The type names often match existing devices in AmigaOS. If a device returns such a type, the device must be able to at least handle all the documented features for V40 of the operating system that the respective original device has.

If an old style or new style command can not be supported in a compatible manner, the device is required to return IOERR_NOCMD for safety reasons. Modification of the specification is not allowed. Add a 3rd party command if needed.

It is illegal to "reuse" command numbers of documented standard commands that cannot be supported for other purposes. E.g. a device driver that does not support TD_GETGEOMETRY and uses it for something else can never be a new style device and may not support new style commands.

It is also illegal to use the results of an NSCMD_DEVICEQUERY without testing DeviceType first.

```
#define NSDEVTYPE_UNKNOWN       0   /* No suitable category, anything */
#define NSDEVTYPE_GAMEPORT      1   /* like gameport.device */
#define NSDEVTYPE_TIMER         2   /* like timer.device */
#define NSDEVTYPE_KEYBOARD      3   /* like keyboard.device */
#define NSDEVTYPE_INPUT         4   /* like input.device */
#define NSDEVTYPE_TRACKDISK     5   /* like trackdisk.device */
#define NSDEVTYPE_CONSOLE       6   /* like console.device */
#define NSDEVTYPE_SANA2         7   /* A >=SANA2R2 networking device */
#define NSDEVTYPE_AUDIOARD      8   /* like audio.device */
#define NSDEVTYPE_CLIPBOARD     9   /* like clipboard.device */
#define NSDEVTYPE_PRINTER       10  /* like printer.device */
#define NSDEVTYPE_SERIAL        11  /* like serial.device */
#define NSDEVTYPE_PARALLEL      12  /* like parallel.device */
```

More types will be defined by Amiga Technologies as necessary and requested. Each device type represents a certain set of minimum capabilities that the device must support. If the type does not match a device that does exist in OS V40, the exact requirements are defined below.

DeviceSubType

There might be special incarnations of a device with special capabilities beyond the standard set. At the moment none are defined and a device is required to return 0 here. As extensions that are marked by this field have to be upwards compatible, this field need not be tested by users who don’t need any special capabilities beyond the standard set. Otherwise it must be tested first.

SupportedCommands

This points to a 0 terminated list of io_Command values that are supported by the device. This list must contain all legal command values, old style, new style, special commands, that are understood by the device and do not cause an IOERR_NOCMD right away.

You might notice the similarity to the SANA2-R2 (S2R2) S2_DEVICEQUERY command. It is intentional.

If you are developing software that needs to use 3rd party commands, you must try an *exact* device identification via the lib_IdString in the device base and reject all devices that you don't know for a new style device.

A code fragment that could help you set up a new device query follows:

```
    struct IOStdReq *io;
    struct NSDeviceQueryResult nsdqr;
    LONG error;
    BOOL newstyle = FALSE;
    BOOL does64bit = FALSE;
    UWORD *cmdcheck;

    ...
    nsdqr.SizeAvailable  = 0;
    nsdqr.DevQueryFormat = 0;

    io->io_Command = NSCMD_DEVICEQUERY;
    io->io_Length  = sizeof(nsdqr);
    io->io_Data    = (APTR)&nsdqr;
    error = DoIO((struct IORequest *)io);
    if((!error) &&
       (io->io_Actual >= 16) &&
       (nsdqr.SizeAvailable == io->io_Actual) &&
       (nsdqr.DeviceType == NSDEVTYPE_TRACKDISK))
    {
        /* Ok, this must be a new style trackdisk device */
        newstyle = TRUE;

        /* Is it safe to use 64 bits with this driver? We can reject
         * bad mounts pretty easily via this check!
         */
        for(cmdcheck = nsdqr.SupportedCommands;
            *cmdcheck;
            cmdcheck++)
        {
            if(*cmdcheck == NSCMD_TD_READ64)
            {
                /* This trackdisk style device supports the complete
                 * 64 bit command set without returning IOERR_NOCMD!
                 */
                does64bit = TRUE;
            } /* if */
        } /* for */
    } /* if */
```

Device specific requirements

Depending on the type returned by NSCMD_DEVICEQUERY, a device may support device specific commands in the $c000-$ffff range.

It is valid and strongly suggested behaviour for a device to reject all new style commands except NSCMD_DEVICEQUERY with IOERR_NOCMD unless the caller

executed a valid and successful NSCMD_DEVICEQUERY at least once before.

There is also a minimum list of requirements for some types of devices.

If a device type is not listed below, no device specific commands are defined for it at this time. The device might still support special custom 3rd party commands outside the reserved range, though. If a device type is listed below, the device driver must conform to the mentioned specifications. A new style device specific command may possibly match an old style extended command exactly to avoid future identification troubles or code misunderstanding.

Device specific commands and behaviour

NSDEVTYPE_TRACKDISK

May support all V40 trackdisk.device commands and possibly HD_SCSICMD as scsidisk.device is obviousy a trackdisk like driver. If it doesn't, IOERR_NOCMD must be returned for the respective command. 3rd party commands may be added in slots that don't conflict with the V40 command set or the reserved areas.

A new style trackdisk like device *must* also return this new identifier for TD_GETDRIVETYPE. You should use TD_GETGEOMETRY on a new style driver to obtain geometry hints if needed.

```
#define DRIVE_NEWSTYLE  (0x4E535459L)   /* 'NSTY' */
```

At the moment, only four new style commands in the device specific range may be implemented.

```
#define NSCMD_TD_READ64     0xc000
#define NSCMD_TD_WRITE64    0xc001
#define NSCMD_TD_SEEK64     0xc002
#define NSCMD_TD_FORMAT64   0xc003
```

These commands behave almost like the trackdisk commands CMD_READ, CMD_WRITE, TD_FORMAT, respectively, but support 64 bit handling for large storage devices. The upper 32 bits, bit 32 to 63, need to be put into io_Actual before the commands are executed. io_Actual can be named io_HighOffset in that case.

If you choose to implement the 64 bit commands, you must implement all of them, even if some of them would possibly return dummy results. A partial implementation with some 64 bit commands returning IOERR_NOCMD is not acceptable.

A detailed description of the 64 bit commands can be found in the document "trackdisk64". Implementors are required to read this document.

```
Recommended use
===============

Anyone who wants to take advantage of the new style device interface should do
this:

    - OpenDevice() as usual
    - Try a NSCMD_DEVICEQUERY
    - If successful according to the rules mentioned above,
      use the commands as listed in "SupportedCommands".
    - Otherwise it must be an old style device. Hope and pray that
      you got what you wanted.

*** EOT ***

Heinz Wrobel
<heinz@amiga.de>
```

1.10 nsd2

```
Command Sets for Devices
========================

There  has always been the problem that people made incompatible extensions to
the  device  command  set.  Doing  a  low  level  format  on a TD_GETGEOMETRY
trackdisk command is probably the worst example for this that anyone can think
of.

Amiga  Technologies  intends  to solve this problem now by specifing new rules
that EVERY device has to adhere if it wants to be compatible to any OS release
beyond OS 3.1 (V40).

WARNING!
========

If  you  don't follow these rules when writing or updating a device driver, we
will  have  to list your device publically as violating the Amiga Technologies
programming  standards!  And  it will make the system crash which most likely
isn't satisfactory to your customers!


Rules for Device development
============================

Rule #1: Support IOERR_NOCMD
----------------------------

In  1991,  Commodore  specified  publically and in writing that any Exec style
device  must  return  IOERR_NOCMD on any unknown command value [RKM Libraries,
3rd Edition, page 924].  Note that this specification was for compatibility to
OS  2.0  and  that  five years have gone by in the meantime.  There was enough
time to adapt.

    Returning IOERR_NOCMD on any unknown command value is a *MUST*!
    If you don't do this, you break things!
```

Remember, this is an old rule. We can no longer tolerate violations of this rule!

Rule #2: Don’t modify Reserved Command Semantics

Up to and including OS 3.1 (V40), custom device commands usually started at CMD_NONSTD. Each developer added commands for custom device features at will. This lead to messy and incompatible devices and strange compatibility tricks to identify device capabilities.

As of January 1st, 1996, Amiga Technologies reserves two ranges in the set of command values for future enhancements. Only commands as specified by Amiga Technologies may be used here. It is illegal for a device developer to randomly "add" custom commands or command features within these reserved areas!

There are 65536 command values posisble with io_Command:

$0000 - $3fff	old style and 3rd party commands
$4000 - $7fff	RESERVED AREA!
$8000 - $bfff	old style and 3rd party commands
$c000 - $ffff	RESERVED AREA!

To say it again: Commands in the reserved areas may only be assigned and specified by Amiga Technologies. Any "custom" implementation of these commands or other commands in these reserved areas is illegal and violates programming standards!

As of this writing, not that many new style commands have been defined. The interface to new style commands is described in a separate document "NewStyleCommands".

Rule #3: Check for new style commands before using them

Have you read "NewStyleCommands" yet? Yes? Then you know that you may only use new style commands after checking the device type and capabilities via the new style NSCMD_DEVICEQUERY.

Heinz Wrobel
<heinz@amiga.de>

1.11 nsd3

Support for large storage devices

Up to and including V40 (OS 3.1), there was only support for disk like storage devices up to a size of 4GB. This text specifies a command set for 64 bit based access to trackdisk/scsidisk like drivers. It is based on the "new style device" standard which is described in the document "NewStyleCommands".

Note that these commands may not be implemented unless the implementation fully conforms to the "new style device" standard. Note the recommendations and restrictions for NSDEVTYPE_TRACKDISK well.

Command Set

The command names are similar to their respective 32 bit counterparts.

```
    #define NSCMD_TD_READ64     0xc000
    #define NSCMD_TD_WRITE64    0xc001
    #define NSCMD_TD_SEEK64     0xc002
    #define NSCMD_TD_FORMAT64   0xc003
```

Transfer Offset and Size

The transfer size used for io_Length is still an unsigned 32 bit value. This is not considered to be a problem. To specify a 64 bit transfer offset, io_Actual is used for the upper 32 bit of the transfer offset when setting up a struct IOStdReq. io_Actual is aliased io_HighOffset for this purpose. The io_Offset field still represents the lower 32 bit of the 64 bit offset. It is necessary to reinitialize all the fields on every I/O operation as described in the official OS documentation.

Implementation behaviour

Any 64 bit command may be used in the full 64 bit range, including the current 32 bit range of course. The behaviour for both 32 bit and 64 bit commands is undefined if the respective range is exceeded via a offset/length combination. Obviously this should be avoided.

Command Identification

Identification of the availability of these new 64 bit commands must be tested according to the "new style device" standard.

Recommended use

It is recommended that availability is tested once only, e.g. on startup of a filesystem. At that time it should be decided first wether the 32 bit command or the 64 bit commands need to be used as the test for 64 bit commands is obviously not necessary if 32 bit is enough to do the job. This decision should remain firm, i.e. both command sets should not be mixed. If the access range lies completely within the 32 bit range, only the 32 bit commands should be used. If not, the 64 bit commands must be used. This can be achieved very easily by using variables for the command numbers. Note that there does not need to be a test if io_HighOffset should be set up. io_HighOffset can be set up for both sets of commands. For 32 bit commands,

it will be ignored.

Notes

This specification does not cover ETD commands. For ETD commands there is no 64 bit specification available. It is not expected at this time that there is a need.

Acknowledgements

This document is largely based on previous work by Ralph Schmidt, Ralph Babel, Randell Jesup and others. Many thanks to them for working out necessary details.

Heinz Wrobel
<heinz@amiga.de>