Ik

Alexis WILKE

COLLABORATORS

	TITLE :		
	lk		

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Alexis WILKE	March 28, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

lk

1.1   lk V1.04

```
                              lk© Index

                  lk© was written by Alexis WILKE (c) 1993-1994
                            All rights reserved.

  Please, select an item in the following list:
  Help for Help
  Frequently Asked Questions ... and answers!
  About (Copyright Notice)
  Disclamer
  What is lk© ?
  And the next releases ?

  Start lk©

  Amiga library support
  Auto-init/exit
  Automatic WITH files
  Archived libraries
  Command line
  Internal variables
  Keywords list
  Known bugs
  Memory manager
  Overlays
  Preferences file
  Special hunk names
  '_stub' function

  All lk© errors

  QuickDOS© library
  lkopts© program
  About my other products
```

1.2   lk V1.04 - (About)

```
                                lk V1.04
                          The no limit linker.


   After I started with a huge (About 512Kb) program which needed a  linker,  the
current linker I used crashed. I quickly saw (Before the  guru)  that  a  memory
problem was the source of that horrible crash. At that time I  used  BLink V7.2
and my program was about 420,000 bytes. Another linker was necessary...


   lk is born from that time and is more convenient for me  and,  hopefully,  for
you. The next explainations show you the tremendous possibilities offered by lk.
Actualy, a total of 15741 lines (317092 bytes) of code  were  written  in  68000
with GenIm2 from DevPac. And this does not take in account the large  amount  of
lines I wrote for my library which is 90508 bytes when compiled...


   Why use lk anyway ?
   After several tests, checking the speed between lk and BLink, I saw that lk is
more than twice faster. (Note that I tested that with my  biggest  codes  for  a
better approximation.) Even so, I have several things to do to improve the speed
some more.
   And lk supports new hunks (HUNK_RELOC32SHORT, HUNK_DREL<size>, ...)


   lk is a shareware also, if you like this program and you use it,  please  send
me US $15.00 or more at:


                            Alexis WILKE
                       1511, SW Park Ave #615
                        Portland, OR 97201
                             U.S.A.


                                 or


                            Alexis WILKE
                      3, rue du docteur Tuefferd
                         25200 Montbéliard
                              FRANCE


     Phone: (503) 248-5607
     (Pacific hours 9 to 9, thanks!)


                   e-mail: alexis@@netcom.com


       Note: I accept money orders and checks from U.S.A. and France, or the
Euro-Check, otherwise use cash money! Only U.S. dollars and francs are accepted
                              (FF90,00).
    This will allow me to make it faster and more confortable. Also, if you have
any requests, concerns, questions, comments or any problem with this linker, do
 not hesitat to contact me. I will be glad to make improvements and, of course,
                     correct any bug or disfunction.


                     This product is a Copyright of
                      Alexis WILKE (c) 1993-1994.


   The use of lk editor is at your own risk.
```

No copy of the registered version is permitted except for personal back-up.
Any copy of the demonstration version MUST be a copy of the entire archive file
unaltered in any way which ever it could be.

No commercial use can be provided by any one without the written and express
consent of Alexis WILKE.

                    Credits

        Programming   Alexis WILKE

        Assemblers   BAsm V1.99
                           & DevPac© V2.00

        Linker   lk©

        Text Editor   CygnusEd© V2.12

        Icons   Alexis WILKE
                           & Patrik Lundquist

        Documentation   AHelp© V1.00
                           & Icon® V8.00
                           & AmigaGuide® V39.11
          Written by   Alexis WILKE
          Some Corrections
        & Ameliorations by   Patrik Lundquist

Special thanks to the following people which helped me to provide the last
touches to lk:
        Michael van Elst
        Patrik Lundquist
        Ralph Schmidt

lk© is a copyright of Alexis WILKE.
                                                          All rights reserved.
AHelp© is a copyright of Alexis WILKE.
                                                          All rights reserved.
AED® is a copyright of Alexis WILKE.
                                                          All rights reserved.
II® is a copyright of Alexis WILKE.
                                                          All rights reserved.
Commodore® is a registered trade mark of
                                               Commodore Business Machines.
Amiga® is a registered trade mark of
                                               Commodore Business Machines.
DevPac© is a copyright of HiSoft.
                                                          All rights reserved.
BAsm© is a copyright of Ralph Schmidt.
                                                          All rights reserved.
CygnusEd© is a copyright of CygnusSoft Software.
                                                          All rights reserved.
DICE© is a copyright of Matthew Dillon.
                                                          All rights reserved.

1.3   lk V1.04 - (Addrall)

                                                               Page 11-1
                         ADDRALL or -ma or -mw <value>
                            Default: no addresses

   Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

   This instruction defines the address of the code hunk and set the SINGLEHUNK
flag. This way one big block is saved with a static address.
   Thank you to refer you to ADDRESS CODE and SINGLEHUNK.

   See also:
     ADDRESS
     SINGLEHUNK
     Become Registred

1.4   lk V1.04 - (Address)

                                                               Page 11-2
                      ADDRESS CODE or DATA or BSS <value>
                            Default: no addresses

   Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

   This instruction defines an origin for the code, data and BSS hunks. This is
especially nice for those which are making games, ROMs or Unix like links.
   The memory requirements can not be taken in account in this case.

   Each type of hunk is supposed to be unique. You can use the SINGLE
instructions for this purpose. Any data which is read-only should be given
within the CODE hunk. The special instruction ONEDATA will suppress the BSS
hunks and link them within the data hunks. The special instruction SINGLEHUNK
will create one hunk of code. No data neither BSS will exist in that last case.

   When several hunks exist, they are supposed to be one after another into the
resulting files. The address is computed that way. One file is created per hunk,
those files receive the following extensions:
        code: <file name>.c<value>
        data: <file name>.d<value>
        bss: <file name>.b<value>
where the <file name> is the destination file name and <value> is the hunk

number (in decimal.)
  Note: because your compiler may move some hunks, the hunk numbers might change
from one link to another.

  ADDRESS CODE <value>
    Defines the absolute address where the code will be loaded.

  ADDRESS DATA <value>
    Defines the absolute address where the data will be loaded.

  ADDRESS BSS <value>
    Defines the absolute address where the BSS hunk will be loaded.

  See also:
    ADDRALL
    ONEDATA
    SINGLE
    Become Registred

1.5   lk V1.04 - (Addsym)

                                                              Page 11-3
                                  ADDSYM
                 Default: keep symbol tables and known debugs

  Create simple LINE debugs from the unit name and the missing symbols. This is
also an equivalent to CREATESYMBOL and CREATEDEBUG both used at the same time.

  See also:
    ADDSYMBOL
    CREATESYMBOL
    KEEPDEBUG
    NOBSSDEBUG
    NOCODEDEBUG
    NODATADEBUG
    NODEBUG
    NOLIBDEBUG
    NOLOCALSYMBOL
    NOOVLDEBUG
    NOSYMBOL
    SETADVISORY
    Become Registred

1.6   lk V1.04 - (Advisory)

                                                              Page 11-4
                              CLEARADVISORY
                               HUNKADVISORY
                            LEFTADVISORY or LA
                               SETADVISORY
                          Default: HUNKADVISORY

Only the commercial version will have this instruction fully supported. Registred people will receive that version.

SETADVISORY asks for the advisory bit. This instruction will force the advisory bit to be set in all hunks of the following object files, not taking in account the possible flags setting defined inside each hunks, until another advisory instruction appears.

The CLEARADVISORY instruction will clear that bit instead of setting it. HUNKADVISORY restores the normal state.

The LEFTADVISORY instruction enables you to suppress hunks from the object files having the advisory bit set. This instruction should not be used conjoined with any other advisory instruction.

This bit is used to create some kind of debug hunks (ignored by AmigaDOS and also the 'LoadSeg()' function.)

```
See also:
    ADDSYM
    CREATEDEBUG
    CREATESYMBOL
    KEEPDEBUG
    NOBSSDEBUG
    NOCODEDEBUG
    NODATADEBUG
    NODEBUG
    NOLIBDEBUG
    NOLOCALSYMBOL
    NOOVLDEBUG
    NOSYMBOL
    STRIPDEBUG
    Become Registred
```

1.7 lk V1.04 - (Alv)

Page 11-5
ALV or ALVS
Default: NOALV

lk is build to link codes of any sizes. You still may use the BSR and JSR/JMP d16(PC) instructions and lk will create a list of JMP instructions to handle those which are too large or defined between hunks. The usage of automatic-load vectors is usuly safe. Now you must know that works in hunks of code only and lk suppose that those hunks are read-only. If those two points are not respected, the change to have a bug to show up are high.
lk has a more powerful system than Slink to create the load vectors. For lk to produce less relocations in some cases.

When you link a small program, you should avoid ALV and save one or more passes into the reloc and reference tables.

You may use the command MAKERELATIVE if you want to create some relative BSR/BRA from JSR/JMP instruction. This will reduce the size of your relocation information. Now this instruction may generate a lot of bugs into your program.

Please refer to its documentation before to use it.

Note: the BSR.B instruction is supported, but it should never be used with an external pointer. And lk also supports the GenIm2 bug which generate the code $6xFF rather than $6x00.

See also:
    MAKERELATIVE
    NOALV
    SMALLCODE
    VALIDNOP
    Become Registred

1.8   lk V1.04 - (Amigalibrary)

Page 11-6

AMIGALIBRARY or AL
Default: normal executable

NOTE: if this flag is set (or one of the LIBFD, LIBID, LIBPREFIX, LIBREVISION or LIBVERSION instructions is used,) lk supposes that you want to use the default 'library.o' file. Any file which contains the '_LibraryBase' symbol (XDEF) will replace the default lk object. lk will link that file as the first object and it must be of type code. If such symbol does not exist, lk will first check for 'LK:LIB/library.o', if that file does not exist, the internal one will be used instead. The default lk header file does not handle Slink or DICE required initialisations.
    Another solution is to use the DEFINE instruction to forbid the usage of the default library with:
    DEFINE _LibraryBase=0

This instruction supposes that you are creating an Amiga library (like those present into your LIBS: path.) lk will automatically generates the table list for the function jumps from the corresponding file description ('.fd' file.) The name of the '.fd' file will be defined with the LIBFD instruction or from the destination file name. The name of the library will be the destination file name.
    If you have to define private functions into the file description, you should define them with the names 'private1', 'private2', etc... and use the DEFINE instruction to redirect them on your functions:
    DEFINE private1=<my function1 name>
    DEFINE private2=<my function2 name>
    ...

If you need to receive the parameters on the stack you will use LIBSTACKPARAM instruction. By default lk supposes that your functions receive registers.

The version and revision numbers of the library are defined through the instructions LIBVERSION and LIBREVISION. Those will also define the __LIBRARYVERSION and __LIBRARYREVISION symbols respectivly. Both values are used to define the library identification as well than the library version used to bump an open.

The string identification will be defined throught one of the COPYRIGHT or LIBID instructions. Only the copyright is required because the remained is

always generated by lk. If no copyright is defined, the default identification
string is still created by lk.

   If you use the default library header given with lk, you must define the
following functions, and the variable '_LibraryBase' is available to you. This
global variable contains the pointer of your library which does not need to be
opened by again, and can always be used in replacement to the variable called
'mylib' given in any of the following functions. To avoid the usual
auto-init/exit open/close code, you may add a definition when you link:
    DEFINE <lib base name>=_LibraryBase
like for instance:
    DEFINE _BOSBase=_LibraryBase

  1. 'LibInit' which contains all necessary initialisations for your library.
     Its C proto-type is:

     extern struct Library * _LibInit(
                                                     struct Library *mylib);

     If you have nothing to create, just type:
     long _LibInit(mylib)
       struct Library *struct;
       {
          return (mylib);
       }

     This function must return the library pointer (mylib) to accept the library
     installation, otherwise return null (failed) and the complete library will
     be freed by the system. This function should avoid calls to other functions
     of the library.
     If your compiler support register definition you may defines 'mylib' as
     register D0 (and A6 is SysBase.)

  2. 'LibOpen' which contains the necessary initialisations for each new open
     intented by user. The system will increment the open counter by one when
     this routine does not fail. Its C proto-type is:

     extern struct Library * _LibOpen(
                                                     struct Library *mylib);

     If you have nothing to allocate, just type:
     long _LibOpen(mylib)
       struct Library *struct;
       {
          return (mylib);
       }

     This function must return the library pointer if the library has been
     opened with success.
     If your compiler supports register definitions you may defines 'mylib' as
     register D0.

  3. 'LibClose' which contains the necessary destructors. The system will
     decrement the open counter by one. Its C proto-type is:

     extern BOOLEAN _LibClose(
                                                     struct Library *mylib);

```
    If you have nothing to free, just type:
long _LibClose(mylib)
  struct Library *struct;
  {
     return (0);
  }
```

This function must return FALSE if the library does not have to be
expunged. Do not forget that you cannot expunge your library until all
users closes it (however the expunge code check the counter.)
If your compiler supports register definitions you may defines 'mylib' as
A6.

4. 'LibExpunge' which contains the necessary destructors to free the library
   from memory. If this function is not defined lk assume that no expunge is
   available and the default LibExpunge will be used (it does nothing!) Its C
   proto-type is:

```
extern long _LibExpunge(

                                        struct Library *mylib);
```

```
    If you have nothing to destroy, just type:
long _LibExpunge(mylib)
  struct Library *struct;
  {
     return (1);
  }
```

This function must return FALSE if the library cannot be expunged.
Otherwise it will return any non zero value and the system expunge function
will free the library memories (base and segment list.) The system expunge
function automatically prevents the destruction while some users still
access the library.
If your compiler supports register definitions you may defines 'mylib' as
A6.

    IMPORTANT:

  In the LibInit() and LibExpunge() functions, you MUST avoid the usage of the
DOS filer system, otherwise the Forbid()/Permit() of the system are lost and the
system may crash.

  See also:
    COPYRIGHT
    FDLIB
    LIBFD
    LIBID
    LIBPREFIX
    LIBREVISION
    LIBVERSION
    VERSION
    Become Registred

1.9   lk V1.04 - (Anyrelative)

```
                          ANYRELATIVE
                       Default: only code
```

   Enable lk to generate some code for too large relatives founded in data hunks.
This works only when MAKERELATIVE was specified.

   See also:
      MAKERELATIVE
      Become Registred

1.10   lk V1.04 - (Arc)

```
                       Archived Libraries
```

   Note: before to read this, you have to know that you will need the tool "LhA"
to be able to use the archive supports.

   Because librairies are very big files, it takes a lot of rooms on your hard
drive. If you want to save some space, you may archive all libraries. lk
supports two different archive modes, and a mix of both:

   1. one library by file, the name of the archive is the name of the library
      with the '.lib' changed into '.a'.

   2. all librairies in one single file, the name of the archive must be
      'libraries.a'.

   3. a mix of both, single and multiple libraries; lk searchs for a single
      library archive at first, if it does not exist, the multiple libraries
      archive is scanned.

   In any case, an archive file name can have one of the following extensions:

   1. ".a"   searched first
   2. ".lha"
   3. ".lzh"   searched last

   Because 'LhA' tool scan LHAOPTS, you can use that environment variable to add
some flags.

   There is nearly no change to do to use archived libraries:

   1. archive your libraries; this is done by the use of "LhA" tool in a CLI; the
      archives will be put in one of the directories defined in the library
      paths;

         lha a <filename>.a <filename>.lib
         (single library)

      or

```
    lha a libraries.a <lib1>.lib <lib2>.lib ...
    (multiple libraries)
```

the starting small letter 'a' means add to the archive, you will refer to
the archiver documentation for more information.

note: no path can appear inside the archive, otherwise the resulting files
      will not be found.

2. ensure that the device "T:" exists; this is usually an assign on "RAM:T" (T
   being a directory); it is better to have "T:" in a ram disk, while it would
   take a lot more time to uncompact a file and you would use hard disk space;

```
    assign T: RAM:T
```

3. change the extension of the libraries on your command lines or into your
   WITH files; also the '.lib' will become '.a'; libraries present in the
   commun file named 'libraries.a' will keep their original names (only the
   extension is changed) on command lines and WITH files, lk automatically
   makes the necessary transformations.

   note: your object libraries MUST have had the extension '.lib' because lk
   assumes it so. Of course you might just renamed them before to archive
   them.

Example:

  If you use lk with Amiga functions, you probably use the 'amiga.lib'
  library. There are the steps to follow into your CLI to transform that
  library into an archived library:

  [assign T: RAM:T]   ;if it does not exists

  lha a lib:amiga.a [<path>]amiga.lib

or

  lha a lib:libraries.a [<path>]amiga.lib

command line changed:

  lk ... lib ... amiga.a ...

See also:
    KEEPARC
    LIBPATH
    LIBRARY

1.11   lk V1.04 - (Archiver)

                              ARCHIVER
                        Default: 'lha -q e '

If you use the archived library feature, you may want to use your own archiver
rather than 'LhA'. This command enables you to define the archiver command line.
This command line will have the file names added to it this way:

        <archiver> <library name> T:

  1. The <archiver> is the startup of the command line that you define,
     including the necessary options.

  2. The <library name> is the name of the archive file (Usually 'libraries.a'.)

  3. The is the name of the file which has to be extracted, but it will appear
     only if the library name differ.

  4. The 'T:' at the end is the destination directory.

  Your archiver will have to support those file names in this order to be used
with 'lk'. The default command line for 'LhA' is also:

    'lha -q e '

  Note: the space after the 'e' is very important and you may not put one if
your archiver does not need it.

  For more information about the archive feature of lk, please refer
to~'Archived Libraries'.

  See also:
     KEEPARC
     Archived Libraries
     Become Registred

1.12   lk V1.04 - (Askundef)

                                ASKUNDEF
                         Default: process an error

  This instruction make lk react in asking a value for undefined symbols. This
will be a little be more like BLINK/SLINK does. It will execute the WARUNDEF
command at the same time.
  After this instruction is encountered, lk will wait on user to type a value
for the each unknown symbol. The default will be zero for an absolute and
'_stub' for a label.

  See also:
     MAXREF
     WARUNDEF
     Become Registred

1.13   lk V1.04 - (Attributes)

                              ATTRIBUTES <value>
                               Default: DEFAULT

   Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

   Link all hunks of specified object files to be loaded in the given memory.
This is done regarless of any information present in those file hunks. It will
last with the next memory instruction (CHIP/DEFAULT/FAST or PUBLIC.) The value
represent a combinaison of the memory attributes:

        MEMF_ANY   ($00000000)

        MEMF_PUBLIC   ($00000001)
        MEMF_CHIP   ($00000002)
        MEMF_FAST   ($00000004)

        MEMF_LOCAL   ($00000100)
        MEMF_24BITDMA   ($00000200)
        MEMF_KICK   ($00000400)

        MEMF_CLEAR   ($00010000)
        MEMF_LARGEST   ($00020000)
        MEMF_REVERSE   ($00040000)
        MEMF_TOTAL   ($00080000)

        MEMF_NO_EXPUNGE ($80000000)

Note: you should avoid the REVERSE bit while some exec.library are bugged and
will crash the Amiga. The bits LARGEST and TOTAL are useless to allocates a
buffer of memory.

   Because this is a user specification, lk will not send any warning.

   lk will change simple attributes into the commonly hunk definition available.
This means the following will never appear:

        MEMF_ANY   no memory flag used
        MEMF_FAST   becomes FHNK_FAST
        MEMF_CHIP   becomes FHNK_CHIP

   This instruction can appear into the OVERLAY/END block.

   See also:
      CHIP
      DEFAULT
      FAST
      <filename>
      HUNKMEMORY
      PUBLIC
      Become Registred

1.14 lk V1.04 - (Auto)

```
                              Auto-Initialisation
                                 Auto-Exit
```

lk actually supports two types of intialisation and exit code. Those are from:
```
                                  Slink
                                   and
                                  DICE
```

. Slink's autos:

This mode needs the use of the option SLINK.

To have auto-initialisation, SAS/C creates functions named __STI<function name> or @@_STI<function name> (The second one correspond to the same function called with registers not saved on the stack.) Those are called 'CREATORS.'
To have auto-exit, SAS/C creates functions named __STD<function name> or @@_STD<function name>. Those are called 'DESTRUCTORS.'
The code (or data) which needs those initialisation and exit functions is created into the same unit.
The disaventage of this method is to create a relocation pointer for each function. This has a high time and space cost. The following initialisation method does not have that disaventage.

The following example is an illustration of the functions of SAS/C. Note that's not a copy of it. Nothing oblige you to use those models.

To call each of those functions, you need a function called '___construct'. lk will create two lists one after another both terminated with a null pointer and before the first one a pointer to the '___construct' function will be saved. In C, the declarations are:

```
    int (*)(void) __ctors[];
    void (*)(void) __dtors[];

 /* construct is called with ctors and dtors */
    extern int __construct(int (*)(void)[]);
```

with '__ctors[-1] == __construct'
and '__ctors[LASTFUNC + 1] == 0'

Example of the functions which will call '___construct':

```
int __init()
{
  return (__construct(_ctors));
}

int __exit()
{
  return (__construct(_dtors));
}
```

Of course you will have to call those two functions.

Example of '___construct' function:

```
int __construct(int (_tors*)(void))
{
  int r = 0;

  while(*_tors) {
/* This suppose destructors always returns 0 */
    if(!r) r = (*_tors)();
    _tors++;
  }

  return (r);
}
```

Note: SAS/C has a flag to know if the called functions are the creators or the destuctors.

The object file looks like:

```
hunk_unit  <unit name>
hunk_name  <hunk name>
hunk_code  <680x0 code>
           (it could be a hunk of data)
hunk_ext   <ext_def: user function(s)>
           and other externals
hunk_end
hunk_name  <init hunk name>
hunk_code  <680x0 code>
hunk_ext   <ext_def: __STI<user function>
           and other externals
hunk_end
hunk_name  <exit hunk name>
hunk_code  <680x0 code>
hunk_ext   <ext_def: __STD<user function>
           and other externals
hunk_end
```

Note: there is no need to have STI/STD functions into another hunk of code, each function or hunk name can be different, some data or bss hunks can exist and any hunk can have the usual debug and symbols.

To ensure that the order is respected, the creators and destructors names can include a level. The creator function with the lower level will be called first and the higher level will be called last. For the destructor functions the order is inverted; functions with the lower level will be called last and functions with the higher level will be called first. When no level is included the default is used, also 30000. The level is a long word value. Functions with the same level are called in the order they were defined (Now lk does not garantie it.)
  There is the generic syntax:
    __ST?[_]<value><user function name>
  any number of underscore (_) may appear before the value.

  Example of names:
    __STI3890_MyInit   (Called before defaults)

```
   @@_STD__81956_MyExit   (Called after defaults)
```

  The initialisation function, in C, is declared as:
```
    extern int _STImyinit(void);
```
  when a creator returns TRUE (a value other than null) following creators will
not be called.

  The exit function, in C, is declared as:
```
    extern void _STDmyexit(void);
```
  with the default SAS/C '___construct' function all descriptors are always
called, even the corresponding initialisation has never took place.

  How to use it:

    For instance, if you define a function which needs the IntuitionBase, you
will write:

```
  external void *Intuition;
  ...
    #asm
      MOVEA.L _Intuition(A6)
      JSR _LVOOpenWorkBench(A6)
    #endasm
  ...
```

    Then the corresponding initialisation and exit will be:

```
  void *Intuition;
  _STI_OpenIntuition()
  {
    Intuition = OldOpenLibrary
                                                    ("intuition.library");
    return (!Intuition);
  }

  void *Intuition;
  _STD_CloseIntuition()
  {
    if(Intuition) {
      CloseLibrary(Intuition);
      Intuition = (void *) 0;
    }
  }
```

  . DICE's autos:

  This mode needs the use of the option BLOCKUNIT (and eventually CC.) The use
of DICE is needed only if you use DICE object files.

  dlink, to have auto-initialisation, uses the hunk names autoinit and autoexit.
Those names might be modified as you want with lk, there is no restriction.
  Because all hunks with the same name are linked together, all autoinit and
autoexit will be. Those hunks will be found in the same unit than the
function/data which needs to be initialized. You need to have one hunk for the
user function or data, one hunk for the initialisation and one hunk for the exit
code. To produce several levels, several names are used (like autoinit0,
autoinit1, ...) The header will call each of those groups of hunks one after

another.

   Actually DICE uses autoinit0, autoinit1, autoexit0, autoexit1 and autoconfig.
The only problem in DICE is the file 'c.o' which have some relocation pointing
to empty hunks. This could be avoided by creating hunks with NOP's instructions.
Anyway lk handled them, but it needs the use of the option DICE.
   The difference for DICE between the autoinits and autoconfig is that autoinits
may fail.

   The functions of initialisation or of exit does not have the usual RTS at the
end. This also enable, with a single call, to execute all functions of a given
level. To create that kind of hunks you may use DICE which enables
initialisation and exit functions or use your prefered assembler. You will have
to verify that your assembler finish your hunks of code with a NOP when your
code is not long word aligned.

   The last object file will also contains all init/exit hunks with that simple
hunk of code which is the RTS. In DICE that file is the 'x.o' object. An RTS
instruction will be used while initializing if you need to report a failure.
DICE have a function named '__AutoFail' for this purpose.

   Note: all hunks having one of those names must be of type code.

   Example:

   file 'c.a'

```
   ...
   JSR      _autoinit0
   BEQ      .error      ;Exit now
   JSR      _autoinit1
   BEQ      .error
   JSR      __main
   JSR      _autoexit0  ;The order is user choice
   JSR      _autoexit1
   ...
```

   file 'myinit.a'

```
   LEA      _cnttable,A0
   MOVE.W  #$00FF,D0
.next
   MOVEQ   #$00,D1
   MOVEQ   #$07,D2
.cnt
   BTST.B  D2,D0
   BEQ.B   .iszero
   ADDQ.B  #$01,D1
.iszero
   DBF     D2,.cnt
   MOVE.B  D1,(A0)+
   DBF     D0,.next
                     ;<- NO RTS!
```

   file 'x.a'

```
   RTS
```

See also:
    BLOCKUNIT
    BLOCKHUNK
    CC
    DICE
    SLINK
    Become Registred

1.15   lk V1.04 - (Autochip)

                                    AUTOCHIP
                Default: no check on extensions '.oc' and '.chip'

   This enables lk to recognize the '.oc' and '.chip' extensions. Then any file
with those extensions will be linked with the chip memory attribute. See also
the help file about CHIP instruction.

   See also:
    AUTOFAST
    AUTOFD
    AUTOLIBRARY
    AUTOOVERLAY
    CHIP
    Become Registred

1.16   lk V1.04 - (Autofast)

                                    AUTOFAST
                Default: no check on extensions '.of' and '.fast'

   This enables lk to recognize the '.of' and '.fast' extensions. Then any file
with those extensions will be linked with the fast memory attribute. See also
the help file about FAST instruction.

   See also:
    AUTOCHIP
    AUTOFD
    AUTOLIBRARY
    AUTOOVERLAY
    FAST
    Become Registred

1.17   lk V1.04 - (Autofd)

                                    AUTOFD
                   Default: no check on extension '.fd'

This enables lk to recognize the '.fd' extension. Then any file with that
extension will be used as an FD file to defines offsets. See also the FD
instruction for more information.

```
    See also:
        AUTOCHIP
        AUTOFAST
        AUTOLIBRARY
        AUTOOVERLAY
        FD
        Become Registred
```

1.18   lk V1.04 - (Autolibrary)

```
                        AUTOLIBRARY or AUTOLIB
                  Default: no check on extension '.lib'
                The option DICE, set the recognizion of '.lib'
```

This enables lk to recognize the '.lib' extension. Then any file with that
extension will be used as a library file. See also the LIBRARY instruction for
more information.

This option is automatically set when you use DICE. This is necessary because
libraries and objects are given in any order.

```
    See also:
        AUTOCHIP
        AUTOFAST
        AUTOFD
        AUTOOVERLAY
        LIBRARY (LIB)
        Become Registred
```

1.19   lk V1.04 - (Autooverlay)

```
                            AUTOOVERLAY
                  Default: no check on extension '.ovl'
```

This enables lk to recognize the '.ovl' extension. Then any file with that
extension will be used as an overlaid file. See also the OVERLAY instruction for
more information.

```
    See also:
        AUTOCHIP
        AUTOFAST
        AUTOFD
        AUTOLIBRARY
        OVERLAY
        The Overlays
```

```
        Become Registred
```


1.20   lk V1.04 - (Autorun)

```
                        AUTORUN or BACKGROUND
                         Default: no auto-detach
```

   Asks lk to put a startup hunk to run the task. This means the task immediatly
returns to the CLI. This is transparent (like inexistant) to your program and
should also work with any other startup (like 'c.o'.)
   If you want to create your own 'autorun' object file, save it into one of the
library path. Now lk will automatically load it and put it at the beginning of
the executable file (when AUTORUN command has been used, of course.) Please,
refer to LIBPATH instruction for more informations about the default paths.
   Note: to ensure that your own code will be linked at start, you have to use
the

   This function does not actually work when you are creating a library or using
an overlaid program.

   Note: this is not compatible to have a resident code working with this kind of
autorun.

   See also:
       AUTOCHIP
       AUTOFAST
       AUTOFD
       AUTOLIBRARY
       LIBPATH
       OVERLAY
       Become Registred
```


## 1.21   lk V1.04 - (Block)

```
                         BLOCKHUNK or BH
                         BLOCKUNIT or BU
                       Default: BLOCKHUNK
```

   Enables the selection of the block size for the link.

   A hunk block link means:
     Any hunk which is needed from a library will be linked alone into the
     resulting file. If other hunks are present into the same unit, they will not
     be linked into the resulting file; accept if they have at least a reference,
     of course.

   A unit block link means:
     Any hunk which is needed from a library will be linked with the other hunks
     present in that unit. This is the usual way used to enable   some
     auto-initialisation and auto-exit.

   DICE and SLINK instructions automatically call the command BLOCKUNIT.

   See also:
      Auto-Init/Exit
      CC
      DICE
      SLINK
      Become Registred

1.22   lk V1.04 - (Bounds)

                           BOUNDS "<...>" ...
                           Default: no bounds

   Defines hunks which needs auto-bounds. For each of the specified hunk lk
generates four symbols. Like you know, hunks of the same name are linked
together. This way lk enables you to know where a specific hunk group starts and
stops. Bounded hunks will not be linked with other list of hunks (The SMALLCODE,
SMALLDATA and SMALLBSS are useless with them.) However, if you want one code,
data or bss, you might link the result again.
   There is the list of the generated symbols:

      __START<hunk name>
      __END<hunk name>
      __SIZE<hunk name>   (long word size)
      __BYTESIZE<hunk name>

   This function might not work properly if you use one of the FRAG???
instructions.

   Note: this instruction has been created to support the '_COVER' hunk of Slink.
There are the necessary things needed to enable the programming of a COVER
program:

   BOUNDS "_COVER"
   DEFINE __CoverStart=__START__COVER
          __CoverLength=__SIZE__COVER
                   or
          __CoverLength=__SIZE__COVER

   See also:
      Become Registred

1.23   lk V1.04 - (Break)

                        BREAK [C][,[D]][,[E]][,[F]]
                             Default: BREAK C

   Defines break controls. lk can be stopped by Control-C. You can also change

this default with BREAK command. The four AmigaDOS controls may be specified at
a time or separately.

   The control letters are case insensitive. No letter means that no user break
is possible.

   See also:
      Become Registred

1.24   lk V1.04 - (Bugs)

                                                                         Page 5
                       Frequently Asked Questions ... and answers!
                              And the known bugs

   Note: at anytime, if you think there is a problem you may make a first check
using the WARNING instruction. This way you may discover a lot of strange
things...
                                     WARNING

   It says lk was compatible to Slink and DICE, but I checked once without
success!

     lk has been build to replace any linker (other linker object files and
documentations are welcome...) For this reason you need a preferences file to
ensure one mode or another. This is available into 'LK:PREFS/' directory. You
can simply rename (or copy) the needed file into: 'lk.prefs' and run your link
again. Usuly the file 'lk.prefs' remains in 'LK:PREFS/' directory, but you may
move that file into the directory from which you link.
       Please refer to the specific linker for more detailed informations.

   My file is bigger with SD (or SC) !

     This may happend. The best way to make the smallest file than you can is not
always to use SD. Some files will be smaller with only XDATA. lk is able to
suppress zeroes at the end of each data hunk, but SMALLDATA force all hunks to
be linked one after another. This means zeroes will be suppressed only in the
last data hunk.
       The best way to know the best solution is to check each one.

   My resident crash or does not run properly:

     You probably forgot to use the PURE flag. This flag will force lk to
generate a table of relocation offset for the near data hunk. That table is
saved at the end of the hunk (Note that it usuly starts where the BSS was.)
       This table is needed if you use SAS/C 'cres.o' object file.
       Because of this table, link an executable file with any option which change
the data hunk organization may generate an invalid program.

   The command line generates errors:

The command line is not completly compatible to any other linker. This would be hard to do so while each of them has a specific way to do that. Then please, refer to the documentation if you have any particuliar problem.

For instance Slink enable you to use any number of FROM or ROOT instructions. On my side I create a warning and files following the second FROM or ROOT will not be part of the link.

Automatic file type:

If you use the AUTOLIBRARY flag you should avoid the LIBRARY (and also LIB/-l) instruction. This will ensure you that the libraries are used in the given order.

The near data hunk:

Because lk tries to make its best in order to produce the smallest possible executable, the near data hunks may not be ordered like you thought. This is not a problem if you do not have to care about the DATA hunks order. This is the case most of the time.

Strange things happend when I installed QuickDOS!

This is possible. QuickDOS is used by lk in a special way which some times generates some kind of problems. Those might be a misfunction of patterns, or the impossibility to load the same file twice (which should not be useful.) Usuly this can simply be avoid with the use of QDSTOP keyword:
                                        QDSTOP

Compatibility with DICE 2.06.35

The option '-frag' of dice will not split the autoinit and autoexit hunks. lk does not actually do that, it really split every hunks.

The NOEMPTYHUNK option will generate a wrong output file when you link with DICE object files. Those uses initilisation and exit code in a very special way and emptyhunks are necessary for that purpose (Note that no empty hunk is generated into the output file however.)

Compatibility with Slink of SAS any version

You may find some instructions which differ from Slink. All functions from Slink are available but all cannot be used the same way. There is a brief list:
     CHIP
     DEFINE
     FAST
     MAP
     MAXHUNK
     NODEBUG (Use STRIPDEBUG instead)
     OVERLAY/END
     PRELINK
     QUIET
     XREF

The overlays of lk are very specific to it-self and are for sure a lot more
viable. If you had some overlaid executables, you may have some good suprises...

If you created your own startup file, you will probably have some trouble if
you use the default preferences files 'slink.prefs'. This is due to the set-up
available in that preferences file, it suppose that you use the default 'c.o' or
another default SAS startup. Thank you to refer you to each instruction present
into that preference file to see which are and which are not correct with your
own startup file.

Compatibility Blink/Slink with SAS 4.xx

I know, because I worked out on that particular point, that lk may include
the DATA long word '_DOSBase' within the BSS hunk. This means it will be cleared
by the little loop which reset the BSS hunk. To be sure to avoid that bug you
may clear the BSS hunk at the beginning of your 'c.a' file, before to open
dos.library (like DICE does) or put another value than zero inside the DOSBase
variable (which is an ugly technic) to be sure it will not be part of the BSS
hunk.
This can happend because lk, to create XDATA hunks, does not take care of
DATA/BSS hunks but check the end of the final hunks and check back cleared
longs.

1.25   lk V1.04 - (Calm)

                                      CALM

  Sets the warning level to the lk's default. The actual default is 5.

  See also:
     NOWARNING
     QUIET
     WARNING
     WARNINGLEVEL
     Become Registred

1.26   lk V1.04 - (Caseinsensitive)

                            CASEINSENSITIVE or CI
                     Default: symbols are case sensitive

  Test symbols regardless of character case. Any letter with and without accent
will be tested in upper case.

  See also:
     HUNKCASEINSENSITIVE
     Become Registred

1.27   lk V1.04 - (Cc)

CC
Default: does not accept multiple symbols

  C compatible option enable lk to receive multiple label definition from
libaries. No error is emited and the last definition is used.

  This is set by lk when the option DICE is used in which case you should not
type CC.

  The instruction USELASTDEFINE will be used to change the usage mode of the C
libraries.

  In order to make lk compatible to Blink and Slink you have to type the
following define on the top:

    DEFINE _LinkerDB=__DATA_START
            __BSSBAS=__BSS_BASE
            __BSSLEN=__BSS_LENGTH

  See also:
    DICE
    USELASTDEFINE
    Become Registred

1.28   lk V1.04 - (Chip)

CHIP or -CHIP
Default: DEFAULT

  Link all hunks of the following object files to be automatically loaded in
chip memory. This is done regarless of any information present in the object
file   hunks.   It   will   last   with   the   next   memory   instruction
(ATTRIBUTES/DEFAULT/FAST or PUBLIC.)
  You may avoid the usage of the CHIP keyword using the AUTOCHIP command.
Please, refer to that instuction.

  Because this is a user specification, lk will not send any warning.

  This instruction can appear into the OVERLAY/END block.

  Note: this instruction is not an equivalent the the CHIP instruction of BLINK
or SLINK or -chip of dlink. In this case only following files are moved to CHIP
memory, until another memory specification appears.

  See also:
    ATTRIBUTES
    AUTOCHIP
    DEFAULT
    FAST

```
      HUNKMEMORY
      PUBLIC
      Become Registred
```

1.29 lk V1.04 - (Color)

```
                                  COLOR
                           Default: no color
```

  Permit colours errors display.

  Note: I love colors...

  See also:
      Become Registred

1.30 lk V1.04 - (Command)

```
                            The command line
```

  Please select one of the following points:

      Automatic WITH File
      Command Line Contents
      Commentaries
      Instructions Order
      Preferences File

  . Command Line Contents

  A command line is a list of words. A word is a string of any character except
spaces, tabulations, carriage return and line feed; a carriage return is taken
as a space character and should be followed by a line feed. A special case
exist: this is the the quoted words, which may contains spaces but no line feed.

  Keywords are reserved and cannot be used as a file name. The pool of reserved
keywords is supposed to grow in the future. Also you may write the name of your
files between quotes to avoid any confusion with any upcoming version or
release. To see all keywords actually available click on next line:
                            Keywords list

  A file name or parameter is any word which is not a keyword. Any word
explicitly written between quotes will never be used as a keyword. Available
quotes are '...' or "...". Opposite cotes can be used inside the other ones.
Quotes are useful if you need to use specific characters like a space or a sharp
(#) sign.
  Note that the following command line will be parsed as two words:
                            word1"word 2"
is also 'word1' and 'word 2'.

All keywords and file names are case insensitive. But the parameters may have to be written in upper and/or lower case.

. Instructions Order

The order of instructions on your command line is totally insignificant, except in one special case. See also:
                              FROM/ROOT keyword

The order of files might be very significant if you link C objects and libraries. The first file contains the startup and libraries must usuly be given in a specific order. Please, refer to the documentation of you C to know how it works.

. Commentaries

In your WITH files you may include some commentaries. Those are like in assembly language; starting with a semi-colon (;) and ending with the end of the line. The commentaries of your makefile are also supported; those start with a sharp (#) sign and end with the end of the line. Because the diese sign can be used for AmigaDOS patterns, it has to be followed by a space or a tabulation character to be taken as a commentary entry. A diese used alone (followed by a new line) will be taken as the end of the overlay section if you are listing overlaid file names; otherwise it is taken as a commentary (This is done this way to keep the compatibility with Slink.)

. Preference File

The name of the preference file can be defined through 'LK/PREFS' global variable or, when you start lk from the Workbench, 'PREFS' tool type (The tool type will overwrite the global variable definition.)
If none of those definition is given, or if files does not exist, lk uses the default preference file names (in the order it is searched):
                              lk.prefs
                    blinkwith (For Blink compatibility)
The preference file is searched and exectuted before anything else (even the command line!)
The preference file will be searched into all directories defined with the 'WITHPATH'. The default paths are:
. Current Directory
   (directory of lk icon from the Workbench)
. LK:PREFS/
. S:
. SLINKWITH:
Paths can be redefined into the variable 'LK/WITHPATH'. Please refer to the following instruction for more information:
                              WITHPATH

The file which is found by lk can contain all your defaults; like SC, SD and SB.
Note that the preference file can include WITH instructions, but those included files will be executed AFTER the command line. This can be used to always override some user instructions!

. Automatic WITH File

When lk is used without any arguments, it checks for a default WITH file. This

file is named 'Automatic WITH File'. lk searchs for one file only, its name is
one of the following:

                                  lkfile
                                 lk.file
                                 lk.with

lk searchs in this order. The first existing file will be used as a WITH file,
if other files exists they are ignored.   Like said above for the preference
WITH file, each file is searched into the corresponding list of paths; see also:
                              The With Paths

1.31   lk V1.04 - (Copyright)

                                                                   Page 11-25
                            COPYRIGHT or CR "<...>"
                                 LIBID "<...>"
                             Default: no copyright

   Define this executable copyright notice. The string will also be saved into
the destination file with the version number when that one is defined.
   This function also permit to the CLI function VERSION of Commodore to find a
version into the final output.

   When more than one copyright is given, then all of them are concatened one
after another.

   Example 1:
     COPYRIGHT "$VER: lkopts 2.34 (21.6.94) by Alexis WILKE"

   In this example I wrote everythings like I want it to appear into the
executable file. In that case I did not use the instruction VERSION and no
symbol VERSION (and RELEASE) does exist in any of my object files.
   Note: the date is static.

   Example 2:
     TO lkopts
     VERSION 2.34
     COPYRIGHT "by Alexis WILKE"

   will generate the version string:
     "$VER: lkopts 2.34 (21.06.94) by Alexis WILKE"
   Where the date if the date at the link time. In this second example you may
add the instruction TIME to have the link time added into the string:
     "$VER: lkopts 2.34 (10:53:23 21.06.94) ..."

   The instruction LIBID is used the same way but will set the Amiga library
flag. See the instruction AMIGALIBRARY to have all necessary informations.

   See also:
     AMIGALIBRARY
     LIBREVISION
     LIBVERSION
     TIME
     VERSION

Become Registred

1.32 lk V1.04 - (Createdebug)

CREATEDEBUG or CD
Default: keep known debugs

   Create simple LINE debugs from the unit name. This is useful to keep a track
of the original units within the final file.

   See also:
      ADDSYMBOL
      CLEARADVISORY
      CREATESYMBOL
      HUNKADVISORY
      KEEPDEBUG
      LEFTADVISORY
      NOBSSDEBUG
      NOCODEDEBUG
      NODATADEBUG
      NODEBUG
      NOLIBDEBUG
      NOLOCALSYMBOL
      NOOVLDEBUG
      NOSYMBOL
      SETADVISORY
      Become Registred

1.33 lk V1.04 - (Createlibrary)

CREATELIBRARY or CL or PRELINK
Default: executable

   Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

   Like FOR instruction, this flag can be used to force the generation of a
'.Lib' file.

   In that case, the library name will automatically be generated, using the
FROM/ROOT or first object file name and changing extension with '.lib'. If any
of your object file name is similar, a new extension will be concatened
('.obj'.)

   See also:
      FOR
      SINGLEUNIT
      SMALLUNIT
      Become Registred

1.34   lk V1.04 - (Createsymbol)

```
                           CREATESYMBOL or CS
                                 or -S
                       Default: keep symbol tables
```

Create symbol hunk from XDEF definitions. This does not disturbe already
existing symbols while a check is done to avoid any double symbol at the same
address.

```
  See also:
      ADDSYMBOL
      CLEARADVISORY
      CREATEDEBUG
      HUNKADVISORY
      KEEPDEBUG
      LEFTADVISORY
      NOBSSDEBUG
      NOCODEDEBUG
      NODATADEBUG
      NODEBUG
      NOLIBDEBUG
      NOLOCALSYMBOL
      NOOVLDEBUG
      NOSYMBOL
      SETADVISORY
      Become Registred
```

1.35   lk V1.04 - (Default)

```
                               DEFAULT
                           Default: DEFAULT
```

Restart with the default mode; lk will also use the memory informations like
defined in object files.
   If you use the WITH instruction, you can be sure that none of ATTRIBUTES,
CHIP, FAST or PUBLIC keywords are currently in effect.

This may be used after the keywords which disable default file names (lk does
not recognize word(s) as a default object file name after some instructions,
this is done to be sure you want that word to be taken as an object file.)

This instruction can also appear into the OVERLAY/END block.

```
  See also:
      ATTRIBUTES
      CHIP
      FAST
      <filename>
      PUBLIC
      Become Registred
```

1.36   lk V1.04 - (Deficon)

```
                                DEFICON
                            Default: NOICON
                 (DEFICON if lk is started from the Workbench)
```

Ask for the default icon when creating the destination file. If an icon
already exists, it will not be replaced by the default icon. lk tries to pick-up
default icons inside one of the directories defined into the icon path list. lk
will look for one of the following icon:

```
   Destination              Default Icon Name
 AMIGALIBRARY                def_amigalibrary
 ERRORFILE                   def_errorfile
 FD/FDLIB + FDOBJECT         def_library
 FD/FDLIB + FDINCLUDE        def_text
           FDHEADER          def_text
           FDPASCAL          def_text
 FOR (CREATELIBRARY)         def_library
 LIST                        def_list
 TO (normal mode)            def_tool
    (Used with ADDRESS)      def_binary
 XREF                        def_xref
```

Note: the "def_with" icon is used by lkopts© tool.

This command will be overridden by the ICON one.

See also:
    ICON
    NOICON
    UNSNAPSHOT
    Become Registred

1.37   lk V1.04 - (Define)

```
            DEFINE or DEF <label=symbol> or <label=value> ...
             Default: see the list of internal symbols below
```

Add a declaration. This function may be used to declare an equivalent or to
set a specific value dynamically. A value can be specified in decimal or
hexadecimal. When you declare an hexadecimal value, you must start it with the
dollar ($) sign.
  When lk find a wrong value (Too large, with invalid digits.) the value zero
will be used instead (No warning is generated!)

A label cannot start with the dollar ($) sign or any digit (0-9.) This
equivalent will not generates a 'defined twice' error and will be used only when
required (Which means, while this define is present in one of your object file,
the one defined on command line remains useless.)

While those defines are not recursive, a define must be or a value or a symbol

which exist in your object files.

  Labels are case sensitive except if CASEINSENSITIVE was specified.
  lk has some intern variables. Those are:
  (Note: all DATA and BSS variables are defined with only the relative data.
Other DATA and BSS hunks are not considered.)

    __autostartup
        This label is automatically defined when required. It points on the
      first hunk of code (except where it has a reference) found or to the
      startup label when it has been given. This enables a startup code like
      those of 'priority.o' or 'stack.o'.

    __BSS_BASE
        Defines the pointer at the beginning of the BSS block. This is equal to
      the sum of __DATA_BASE and __DATA_LENGTH.

    __BSS_LENGTH
        Defines the total length of the BSS block.
        (This length is given in long words)

    __COPYRIGHT
        The COPYRIGHT (or LIBID) instruction generate that symbol as being a
      pointer on the automatically created string identification. You can use
      it as you like.

    __DATA_BASE
        Defines the pointer at the beginning of the DATA block.

    __DATA_BYTESIZE
        Defines the total length of the near DATA block includind the BSS
      block. This is the sum of __BSS_LENGTH and __DATA_LENGTH. This length is
      given en byte.

    __DATA_LENGTH
        Defines the length of the DATA block.
        (This length is given in long words)

    __DATA_OFFSET
        Defines the offset for (An) to access DATA buffer. You may fix its
      value with the OFFSET command.

    __DATA_POSITION
        Defines the position for (An) from the beginning of the buffer. If you
      use the instruction OFFSET, you must first negate the value
      (__DATA_POSITION = – __DATA_OFFSET.)

    __DATA_SIZE
        Defines the total length of the near DATA block including the BSS
      block. This is the sum of __BSS_LENGTH and __DATA_LENGTH.
        (This length is given in long words)

    __DATA_START
        Defines the pointer needed for An. This is also the sum of __DATA_BASE
      and __DATA_OFFSET.

    __ISDICE

Is TRUE (1) when the keyword DICE was used, otherwise its FALSE (0).

__ISPURE
    Is TRUE (1) when the keyword PURE was used, otherwise its FALSE (0).

__ISSLINK
    Is TRUE (1) when the keyword SLINK was used, otherwise its FALSE (0).

__LIBRARYPRIORITY
    Takes the value given through the LIBPRIORITY instruction. This symbole
can be used to automatically define the Amiga library priority.

__LIBRARYREVISION
    Takes the value given through the LIBREVISION instruction. This is to
support the compatibility with Slink. This symbol is always define and
receive the value 0 by default.

__LIBRARYVERSION
    Takes the value given through the LIBVERSION instruction. This is to
support the compatibility with Slink. This symbol is always define and
receive the value 0 by default.

__NEGATIVESIZE
    Negative size for shared Amiga libraries. This is the number of
function multiply by 6. Note that from V36 of Exec, libraries are
automatically long word justified and this value might be wrong by 2.
It's preferable to use the system defined value which is saved in
LIB_NEGSIZE.

__NUMCTORS
    Total number of constructors.
    (Valid only when the instruction SLINK was used)

__NUMDTORS
    Total number of destructors.
    (Valid only when the instruction SLINK was used)

__NUMFUNCS
    This symbol contains the number of functions available into a shared
  Amiga library.

PRIORITY
    Symbol created when the PRIORITY keyword is used, please refer to that
  command for more information.

REVISION
    Symbol created when the VERSION keyword is used, please refer to that
  command for more information.

__STACKSIZE
    Symbol always available. By default set to 4096 and modified via the
  instruction STACKSIZE.

VERSION
    Symbol created when the VERSION keyword is used, please refer to that
  command for more information.

See also:
    CASEINSENSITIVE
    LIBREVISION
    LIBVERSION
    OFFSET
    PRIORITY
    PURE
    SLINK
    STACKSIZE
    Become Registred

1.38   lk V1.04 - (Dice)

                                                      Page 11-32
                                DICE
                        Default: normal link

   DICE compatible option enable lk to link dice objects. This does not make lk
100% compatible. DLINK is really very special... Note that file with '.lib'
extension will automatically be linked as library files, the option AUTOLIBRARY
is not necessary.

   This option automatically set the CC flag for you. Anyway you have to use all
the following instruction to make a dice link:

        DICE   ALV
        SMALLCODE   SMALLDATA
        OFFSET -$7FFE
        DEFINE   __DATA_BAS = __DATA_BASE
                 __DATA_LEN = __DATA_LENGTH
                 __BSS_LEN = __BSS_LENGTH
                 __RESIDENT = __ISPURE

   The instruction USELASTDEFINE will be used to change the usage mode of the C
libraries.

   See also:
    Auto-Init/Exit
    BLOCKHUNK
    BLOCKUNIT
    CC
    SLINK
    USELASTDEFINE
    Become Registred

1.39   lk V1.04 - (Drel2reloc)

                                                      Page 11-33
                            DREL2RELOC

   lk, by default, uses all DREL hunks as relative offset to the data hunk.
However SAS/C does not do the same. Then I had to implement this instruction to

fulfill the compatibility with SLINK linker.
  When this instruction has been used, lk transforms all HUNK_DREL32 in
HUNK_ABSREF32 (relative in absolute.)

  The usage of this instruction is really not adviced if you are not using
SAS/C, this could cause link errors.

  See also:
    SLINK
    Become Registred

1.40   lk V1.04 - (Echo)

                                ECHO "<...>" ...
                                Default: nothing

  Print the following names/strings into the output until another lk keyword is
encountered.

  See also:
    Become Registred

1.41   lk V1.04 - (Error)

                                lk errors

                              Table of Contents

  001- *** INTERNAL ERROR ***
  002- Out of memory.
  003- Undefined reference '<symbol name>'. <type>
  004- No hunk of type code. Cannot create a valid executable.
  005- Not a hunk code at start. Use TEXT hunk as first hunk.
  006- Hunk at start is not of type CODE ($03E9). Get following hunk.
  007- Two hunks named '<hunkname>' have different type and/or memory and will
        not be linked together. (In files: "<filename>" and "<filename>")
  008- Hunk with CHIP and hunk with FAST requirements are linked together.
  009- Some errors occur during cross reference check.
  010- Cannot create destination file '<filename>'.
  011- Error while writing in destination file '<filename>'.
  012- Relative XDEF defined across different hunks.
  013- Relocation offset <symbol name> out of bounds.
  014- Relative offset <symbol name> out of bounds.
  015- Odd offset in reloc 16/32.
  016- Found relative relocation.
  017- Error while checking object files.
  018- '<filename>' is not an object or executable file.
  019- File '<filename>' is invalid (Error at position: <value>.)
  020- Only ONE executable file can be specified.
  021- Some dirty bytes remains at the end of file '<filename>'.

```
022-  Cannot create a library from executable file.
023-  Resident  libraries  no  longer  supported  by  Commodore  Amiga  (Name:
      "<libraryname>")
024-  Absolute  value  '<symbolname>'  is  defined  as  $<hexvalue>  in   file
      "<filename>", and as $<hexvalue> in file "<filename>"
025-  Absolute/Define  '<symbolname>'  conflict  in  files  '<filename>'   and
      "<filename>".
026-  Define/Absolute  '<symbolname>'  conflict  in  files  '<filename>'   and
      "<filename>".
027-  Definition  '<symbolname>'  defined  twice  in  files  '<filename>'  and
      "<filename>".
028-  Absolute  '<symbolname>'  defined  twice  in  files   '<filename>'   and
      "<filename>".
029-  Symbol '<symbolname>' from file '<filename>' have an unknown type.
030-  Error while parsing command line/file(s).
031-  Skip unknown keyword '<keyword>'.
032-  Only one root available per executable file (<filename>/<filename>).
033-  Destination defined twice (<filename>/<filename>).
034-  Overlay in overlay.
035-  Overlay object defined twice '<filename>' (Keep <filename>.)
036-  Closing overlay without overlay.
037-  Bad label definition '<label>'.
038-  External file defined twice (<filename>/<filename>).
039-  <flag name> flag defined twice.
040-  Two startup definitions. The startup is '<startup 2>' ('<startup 1>'.)
041-  OFFSET for relative data hunk is defined twice. The  offset  is  <value>
      (<value>.)
042-  OFFSET <value> is odd.
043-  The PC-Relative of <value> bits '<symbol name>' uses an absolute value.
044-  The absolute '<symbol name>' of <value> bits uses a PC-Relative.
045-  Invalid hunk number in relocation table.
046-  Invalid reloc to a zero length hunk '<hunkname>'.
047-  Can not find '<symbol name>' symbol of startup definition.
048-  lk made a relative instruction at offset $<value>.
049-  Two relocations at the same position.
050-  No '___construct' symbol for '___ctors' and '___dtors' calls.
051-  Unknown instruction  '<instruction>'  in  FD  file  <filename>  at  line
      #<value>.
052-  No ##public or ##private instruction before line  #<value>  in  FD  file
      <filename>.
053-  Base value missing in FD file <filename> at line #<value>.
054-  Invalid base name in FD file <filename> at line #<value>.
055-  Symbol table size defined twice (Size is <value>).
056-  Symbol table size to small (Defaulted to <value>).
057-  Reloc table size defined twice (Size is <value>).
058-  Reloc table size to small (Defaulted to <value>).
059-  Reloc table size to large (Defaulted to <value>).
060-  Hunk table size defined twice (Size is <value>).
061-  Hunk table size to small (Defaulted to <value>).
062-  Hunk table size to large (Defaulted to <value>).
063-  No object file specified.
064-  Overlays are forbidden in library files.
065-  NODEBUG and CREATESYMBOL cannot be used together. Only NODEBUG is keept.
066-  No root file defined. Cannot generate file.
067-  Error while opening/reading file '<filename>'.
068-  File '<filename>' skipped (Empty).
069-  Code hunk transformed and saved as BSS.
```

```
070-  Data hunk transformed and saved as BSS.
071-  Code hunk (#<hunknumber>) truncated  at  <length>  bytes  and  saved  as
      XCODE.
072-  Data hunk (#<hunknumber>) truncated  at  <length>  bytes  and  saved  as
      XDATA.
073-  Invalid overlay header object file.
074-  More than one file match, keep '<filename>' (Skip: '<filename>').
075-  No match for '<filename>' pattern.
076-  No HUNK_CODE to jump to from overlay header.
077-  File name missing.
078-  Wild character(s) used for single file purpose (In: '<filename>').
079-  Destination name will be '<filename>'.
080-  Two defines with the same name '<symbol>'.
081-  NOLOCALSYMBOL is useless when NOSYMBOL is specified.
082-  Bad FD function declaration at line #<value>.
083-  No library name given in FD file at line #<value>.
084-  Cannot create external file named '<filename>'
085-  Bad break letter (Only C to F are valid).
086-  Width too small. The minimum will be used (<value>).
087-  Width too large. The maximum will be used (<value>).
088-  Unknown width keyword.
089-  Height too small. The minimum will be used (<value>.)
090-  Height too large. The maximum will be used (<value>.)
091-  Unknown margin keyword.
092-  Invalid margin value. It must be defined between 0 and 255 included.
093-  Bad map letter (See documentation.)
094-  Base name defined twice in FD file at line #<value>.
095-  No base defined in FD file at line #<value>.
096-  References '<symbol name>' defined with two  different  types  ($<value>
      and $<value>.)
097-  COLOR flag defined twice.
098-  The base offset is not a multiple of 6 in FD file at line #<value>.
099-  Base value overflow in FD file at line #<value>.
100-  Left/Right members differ  for  function  <function  name>  in  FD  file
      <filename> at line #<value>.
101-  Invalid DREL32 in executable file.
102-  Invalid debug skipped.
103-  Different memory flags in header and hunk.
104-  Hunk type conflict for relative links in file '<filename>' (CODE/DATA).
105-  No library hunk before HUNK_INDEX.
106-  No overlay header needed without overlay.
107-  Math conflict between hunks.
108-  Integer conflict between hunks.
109-  Bad hunk for executable file. It contains memory informations.
110-  Two overlay object handlers defined.
111-  Libraries should not be defined in overlay.
112-  The overlay link '<label name>' not within a hunk of code.
113-  lk cannot work with odd pointer with your CPU.
114-  Relative relocation not in a CODE hunk. No ALV possible.
115-  The overlay link '<label name>' is used with an unknown instruction.
116-  Relative relocation '<symbol name>'  uses  an  unknown  instruction.  lk
      cannot create an ALV.
117-  Hunk too large to create an ALV for '<symbol name>' (<offset>.)
118-  Unknown address keyword.
119-  No destination file deleted while some errors occur.
120-  BSS hunk has some relocation information in file: '<file name>'.
121-  Address $<hexa value> is odd.
```

122- The code header was moved during relative organization.
123- Icon file name defined twice (<file name>/<file name>).
124- <file name> is not a valid icon file name.
125- Fragmented <type> can not be SMALL or SINGLE...
126- Hunk of type CODE named '<hunk name>'. This hunk will not be merged this
     way.
127- Invalid entry hunk '<hunk name>'. Not of type CODE.
128- Multiple entry hunks, skip the last ('<hunk name>'.)
129- Entry hunk can not be in an overlay ('<hunk name>'.)
130- Be careful, MAKERELATIVE may create an invalid program.
131- Overlay handler found in overlay hunks ('<hunk name>'.)
132- ONEDATA should be used with [CLEAR]XDATA and SMALLDATA.
133- Auto-Run is not allowed with overlaid program.
134- Symbol '<symbol name>' defined into library '<library file name>' and
     object "<object file name>".
135- The symbol '<symbol name>' is already defined, lk can not create the
     array.
136- Error file name defined twice (<file name>/<file name>.)
137- Relocation(s) pointing on the near data hunk.
138- The relative '<symbol name>' is defined in a hunk of code.
139- Overlaid file with a corrupted first header.
140- Overlay hunk number larger than indicated in first header.
141- lk found more hunks than indicated in first header.
142- Invalid overlay table size.
143- The number of overlaid unit changed.
144- From/Root '<file name>' should not be defined in overlay.
145- Value missing for '<keyword>' instruction.
146- Label '<symbol>' is undefined, lk cannot create the Slink array.
147- All of the relative DATA/BSS hunks have not the same memory
     requirements.
148- String missing for '<keyword>' instruction.
149- List file name defined twice (<file name>/<file name>.)
150- '<file name>' is not a valid FD file (line #<value>.)
151- Create an ALV for '<symbol name>' symbol.
152- Two library definition files defined (<file name>/<file name>.)
153- Two library prefix defined (<prefix>/<prefix>.)
154- Invalid icon type (Type: '<type name>'.)
155- This file is not an icon.
156- Invalid stack size, the minimum required is <value>.
157- The value <value> is not a valid priority (Bounds -128 and 127.)

001- *** INTERNAL ERROR ***

  When this error happend my algorithm is wrong in some places and should be
reviewed. This can be called a bug!

002- Out of memory.

  Not enough memory for link purpose. This may happen on small memory systems.

003- Undefined reference "<symbol name>". <type>

  A needed reference cannot be satisfied. Four cases may occur:

1. This message is an error, lk will not save the result and also forget
   about that reference. If you use the WARUNDEF command this error will
   become a warning.
       (<type> = <nothing>)
2. The reference is used with a Bcc/JSR/JMP instruction and a default
   '_stub' function exist.
       (<type> = Use the "_stub" function.)
3. The reference is used with a Bcc/JSR/JMP instruction and no default
   '_stub' function exist.
       (<type> = Create a "_stub" function.)
4. The reference is used with another instruction.
       (<type> = Use absolute value $00000000.)

  The '_stub' function is nothing more than:

```
        XDEF     _stub

_stub   MoveQ    #$00,D0
        Rts
```
or
```
    extern _stub;

    int _stub()
    {
        return 0;
    };
```

  This error is usuly followed by a large number of references. If you want to
limit that list of references, use the instruction MAXREF.

  See also:
                              MAXREF
                              WARUNDEF

004- No hunk of type code. Cannot create a valid executable.

  AmigaDOS have to find a hunk of type CODE at start of any executable file.
Also lk will not create a file which cannot be loaded by AmigaDOS.
  Note that this error will occur only if all of your files contain DATA and/or
BSS hunks only.

005- Not a hunk code at start. Use TEXT hunk as first hunk.

  The first file should contain a hunk of type CODE at start. When this is not
the case, lk will search through each hunk to find a hunk of type CODE and named
TEXT. This will become the CODE header.
  The test of the hunk name is case insensitive.

006- Hunk at start is not of type CODE ($03E9). Get following hunk.

  The first file should contain a hunk of type CODE at start. When this is not
the case, lk will search through each hunk to find the first hunk of type CODE.
This will become the CODE header.

007- Two hunks named "<hunk name" have different type and/or memory and will
       not be linked together. (In files: "<filename>" and "<filename>")

   Because each hunk receives its own name, you can give two hunks the same name.
All hunks with the same name are automatically linked togehter, while they are
of similar type and the memory requirements are equivalent.
   If this error occur, you should change the name of one of your hunks.

008- Hunk with CHIP and hunk with FAST requirements are linked together.

   When one of the SINGLE commands is invoked, this warning may happen. To avoid
this warning use SMALL command instead.
   The instruction SINGLE was keept because in a lot of cases it may not
generates any error. Anyway this is not the best way to link hunks together.

009- Some errors occur during cross reference check.

   When some errors occur, the process is stopped. You should therefor correct
errors and rerun lk.

010- Cannot create destination file "<filename>".

   Any DOS error may happen, bad path, invalid file name, file already in use,
etc... Check those errors and try again.

011- Error while writing in destination file "<filename>".

   This should happen only when your disk have some bad blocks or is full.

012- Relative XDEF defined across different hunks.

   A symbol is defined into a hunk and refer to another. This will happen only
when a relative define is encountered.
   In the future this may disappear because lk may have the ability to
automatically create some JMP instructions.
   For some C compiler, you may have to use a large model.

013- Relocation offset "<symbol name>" out of bounds.

   Recompile, regenerate your file and try again! If this error occur again,
delete your assembler/compiler/... it fails.

014- Relative offset "<symbol name>" out of bounds.

   A relative symbol refer to a position to far to enable a consistant
definition.
   This may be overriden by lk in some cases which are branchements and JSR/JMP
relative to PC. In thoses cases you can use the ALV instruction, knowing that

may generates some bugs.
  For some C compiler, you may have to use a large model.

  See also:
                                    ALV

015- Odd offset in reloc 16/32.

  Recompile, regenerate your file and try again! If this error occur again,
delete your assembler/compiler/... it fails.

016- Found relative relocation.

  This warning tells you when some relative symbols exist.

017- Error while checking object files.

  When an error occur while lk is checking object files, this message will
arrive at last and stop the process.
  You will have to correct all errors and rerun lk.

018- "<filename>" is not an object or executable file.

  <filename> must be a valid object file, or an executable.

019- File "<filename>" is invalid (Error at position: <value>.)

  <filename> seems to be an object or an executable file but contains some bad
information or hunks not supported by lk. The position gives you where the error
occured into the file.

020- Only ONE executable file can be specified.

  When lk is invoked to delete debug/symbols or link hunks all together into an
executable, only one file can be specified.

021- Some dirty bytes remains at the end of file "<filename>".

  The file is taken, but one to three unknown bytes were founded at the end.

022- Cannot create a library from executable file.

  Libraries need to have references which are not present in executable files.
  This may change in the future when some debug and/or symbol tables exist.

023- Resident libraries no longer supported by Commodore   Amiga   (Name:
      "<libraryname>")

In old object files some resident library may be founded and deleted by lk.

024- Absolute value "<symbolname>" is defined as $<hexvalue> in file
      "<filename>", and as $<hexvalue> in file "<filename>"

  An absolute is defined twice with two different values. This error shows you
the symbol name and in which files it can be founded.
  One of the absolute should be deleted or both should receive the same value
(In this case you will still be prompted by a warning.)

025- Absolute/Define "<symbolname>" conflict in files "<filename>" and
      "<filename>".

  A symbol was used as an absolute and as a define value. The file names define
the sources, where you will have to modify one of those symbols at least.
  The first file contains the absolute, the second the define (Note: in this
case the absolute was founded first.)

026- Define/Absolute "<symbolname>" conflict in files "<filename>" and
      "<filename>".

  A symbol was used as an absolute and as a define value. The file names define
the sources, where you will have to modify one of those symbols at least.
  The first file contains the define, the second the absolute (Note: in this
case the define was founded first.)

027- Definition "<symbolname>" defined twice in files "<filename>" and
      "<filename>".

  The same symbol was used for two different functions. This will occur even
that symbol is unused. You will have to modify one of the two specified files at
least.

028- Absolute "<symbolname>" defined twice in files "<filename>" and
      "<filename>".

  The same symbol was defined in two files. This will occur even that symbol is
unused. You will have to modify one of the two specified files at least.
  Note that this error will not occur if you do not forbid multiple absolute
definition.

  See also:
                              NOMULTIPLE

029- Symbol "<symbolname>" from file "<filename>" have an unknown type.

  Supported types are EXT_DEF and EXT_ABS for definitions and EXT_REL32,
EXT_REL16, EXT_REL8 and EXT_COMMON for relocation.
  The EXT_DEXT8, EXT_DEXT16 and EXT_DEXT32 types are enable if only a hunk named
data exists or one hunk of type DATA results.
  Any other type will generate this error and the link process will end up.

030- Error while parsing command line/file(s).

  This error may occur while a bad command line or a WITH file line generated an
error, or because one of the specified files was unreadable.
  Note that empty source files will generates an error.

031- Skip unknown keyword "<keyword>".

  Some keywords (Like VERBOSE) cannot be followed by anything else than another
system keyword. If this was a file name also you should add DEFAULT keyword to
enable lk to know about it.

032- Only one root available per executable file (<filename>/<filename>).

  Because lk cannot choose between two files, this error will let you know which
file should be the right root.

033- Destination defined twice (<filename>/<filename>).

  Because lk cannot choose between two files, this error will let you know which
file should be the right destination.

034- Overlay in overlay.

  Only one overlay can be created at a time.

035- Overlay object defined twice "<filename>" (Keep <filename>.)

  Only one object file can be given as being the overlay handler. You will have
to select the one you want.

036- Closing overlay without overlay.

  One overlay can be opened at a time, then only one can be closed. You will
have the suppress this second end.

037- Bad label definition "<label>".

  The define syntax was not fully respected. Check DEFINE instruction for more
information.

  See also:
                                    DEFINE

038- External file defined twice (<filename>/<filename>)

  Because lk cannot choose between two files, this error will let you know which

file should be the right external.

039- <flag name> flag defined twice.

  This warning can be avoid using QUIET instruction. lk will display a warning
for a lot of flags, while some of them might not be used somewhere.

040- Two startup definitions. The startup is "<startup 2>" ("<startup 1>".)

  The startup name is defined twice and differ. lk will take the last definition
("startup 2" also.)

041- OFFSET for relative data hunk is defined twice. The offset is <value>
      (<value>.)

  The relative data hunk usuly receive an automatic offset. This means the
offset may change with your object between several links. To avoid the offset to
change you may use the OFFSET instruction once.

042- OFFSET <value> is odd.

  The offset for relative data hunks is odd and may cause some problems for
68000/010/012 CPUs...

043- The PC-Relative of <value> bits "<symbol name>" uses an absolute value.

  With old linker and assembler, no reference type distinction could be made for
16/8 bits symbols. On V37 and over Commodore added the EXT_ABSREF16 and
EXT_ABSREF8. Anyway lk will support both of them correctly making a warning to
prevent you of an eventual mistake.

044- The absolute "<symbol name>" of <value> bits uses a PC-Relative.

  With old linker and assembler, no reference type distinction could be made for
16/8 bits symbols. On V37 and over Commodore added the EXT_ABSREF16 and
EXT_ABSREF8. Anyway lk will support both of them correctly making a warning to
prevent you of an eventual mistake.

045- Invalid hunk number in relocation table.

  Hunk number must refer to an existing hunk, an overflow (a number too large)
will generate this error. The current hunk will be used instead.

046- Invalid reloc to a zero length hunk "<hunkname>".

  The hunk offset is not null but point into a zero length hunk.

047- Can not find "<symbol name>" symbol of startup definition.

  You gave a symbol name for lk to use as the startup code, but that name is not
available in any of the linked object files.

048- lk made a relative instruction at offset $<value>.

  This warning gives you an information which might be useful if you want to
correct the instruction into your sources. Anyway this can easiely be forgotten.

049- Two relocations at the same position.

  When relocation tables are sorted, this error will tell you that thoses tables
are invalid.

050- No "___construct" symbol for "___ctors" and "___dtors" calls.

  When compiling a SAS/C 6.xx program you need to have some constructors and
destructors. To call those functions, you need the '___construct' function. This
error may happen because you forgot to include 'sc.lib' into your command line
or WITH file.

051- Unknown instruction "<instruction>" in FD file <filename> at   line
      #<value>.

  This warning tells you about skipped instructions in an FD file.

052- No ##public or ##private instruction before line #<value> in FD file
      <filename>.

  lk cannot decide if the first encountered function has to be public or
private. An instruction ##public or ##private must be added before any function.

053- Base value missing in FD file <filename> at line #<value>.

  After the instruction ##bias a value must appear to define the library first
offset. The increment is 6. Note that the physical value is -30 but the fd files
contain 30.

054- Invalid base name in FD file <filename> at line #<value>.

  The name of the library is not valid (bad characters, end of file, etc...)

055- Symbol table size defined twice (Size is <value>).

  Because only one size can be used, the first size (Which is here displayed)
will be keept.

056- Symbol table size to small (Defaulted to <value>).

  A minimum value (for internal reasons) have to be used.

057- Reloc table size defined twice (Size is <value>)

  Because only one size can be used, the first size (Which is here displayed)
will be keept.

058- Reloc table size to small (Defaulted to <value>).

  A minimum value (for internal reasons) have to be used.

059- Reloc table size to large (Defaulted to <value>).

  A maximum value (for internal reasons) have to be used.

060- Hunk table size defined twice (Size is <value>).

  Because only one size can be used, the first size (Which is here displayed)
will be keept.

061- Hunk table size to small (Defaulted to <value>).

  A minimum value (for internal reasons) have to be used.

062- Hunk table size to large (Defaulted to <value>).

  A maximum value (for internal reasons) have to be used.

063- No object file specified.

  No object/executable file, lk cannot do anything.

064- Overlays are forbidden in library files.

  A library is just a list of unit/hunk with references. This cannot include an
overlay hunk which is valid only in an executable file.

065- NODEBUG and CREATESYMBOL cannot be used together. Only NODEBUG is keept.

  This warning can be avoid by deleting one of the instruction CREATESYMBOL or
NODEBUG.

066- No root file defined. Cannot generate file.

  You may have some library/overlay files defined, but stil no consistant
objects. Try to give some object files to lk!

067- Error while opening/reading file "<filename>".

   This file is unreadable. This may occur when a low amount of memory is actualy
available. Otherwise check the filename and path or access validity.

068- File "<filename>" skipped (Empty).

   An empty file will only generates this warning. Anyway you may avoid to use
empty files as object files.

069- Code hunk transformed and saved as BSS.

   lk founded a hunk of type code which contains only zeroes.
   (This should never be true?!?)

070- Data hunk transformed and saved as BSS.

   lk founded a hunk of type data which contains only zeroes. A single BSS hunk
(8 bytes) will be created replacing data hunk. The function modifying hunks take
care of eventual relocation.

071- Code hunk (#<hunknumber>) truncated at <length> bytes and saved as XCODE.

   The end of this hunk of type CODE contains only zeroes and is not saved in
resulting file. The hunk number and length may enable your code to clear that
remainder. Any relocation will forbid a such truncation.

   See also:
                                   XCODE

072- Data hunk (#<hunknumber>) truncated at <length> bytes and saved as XDATA.

   The end of this hunk of type DATA contains only zeroes and is not saved in
resulting file. The hunk number and length may enable your code to clear that
remainder. Any relocation will forbid a such truncation.

   See also:
                                   XCODE

073- Invalid overlay header object file.

   Your overlay header file is not valid. It must have a HUNK_CODE at start, a
reference to '_ovl_root' function, an export to '_ovl_call' function and an
overlay table at start ($0000ABCD.)

   See also:
                                OVERLAY/END

074- More than one file match, keep "<filename>" (Skip: "<filename>").

  For some instructions, only one file name can be specified. But, because the
file name can contains some wildcards, more than one file may match and only the
first one will be loaded.

075- No match for "<filename>" pattern.

  This error will be display rather than the error '067' when you are working
with version V36.00 or more of AmigaDOS.
  See also:
     067- Error while opening/reading file '<filename>'.

076- No HUNK_CODE to jump to from overlay header.

  The overlay header should not be the complete program. Then another hunk of
type CODE must exist to enable lk to create a reloc to that code.

077- File name missing.

  Some instruction needs to be followed by a file name. This prevent some
command line mistake (Because a file name, when equivalent to a keyword, must be
written between cotes.)

078- Wild character(s) used for single file purpose (In: "<filename>").

  This warning will be displayed each time some wildcards are used with an
instruction which can receive only one file.

079- Destination name will be "<filename>".

  When a problem occur creating destination file name (Usuly because an object
file have the same name), the destination name will be displayed.

080- Two defines with the same name "<symbol>".

  While you are defining some define with the DEFINE/VERSION instructions, this
error will prompt you for double definition and a modification will have to take
place.

081- NOLOCALSYMBOL is useless when NOSYMBOL is specified

  While NOSYMBOL deletes all symbols, NOLOCALSYMBOL is really unneccessary.
Suppress NOLOCALSYMBOL or NOSYMBOL to avoid this warning.

082- Bad FD function declaration at line #<value>.

  A function declaration is defined with:

     function name

```
    parameter list
    register list
    cariage return
```

   The parameter list MUST have the same number of parameters than the number of
registers defined into the register list.
   Any name (function and parameter) can contain any letter (Upper and lower
case) plus the underscore (_) sign. The parameters should be separated only by a
coma (,) and the registers can be separated by a coma (,) or a slash (/). The
slash is used to define registers which could be extract from the stack with a
68000 MOVEM.L instruction.

083- No library name given in FD file at line #<value>.

   After the instruction ##base a name must appear to define the library name.

084- Cannot create external file named "<filename>"

  Check the path and file name validity, and rerun lk.
  lk will try to create the external references file when its completly
finishing with all checking, whenever an error occur during check processing
this file will be created.

085- Bad break letter (Only C to F are valid).

   When you change the break controls, only the four DOS break letters may be
specified. Change the following string.

086- Width too small. The minimum will be used (<value>).

   The width of a line have some restiction to enable lk to run finely. This
warning will prompt you if you are asking for a too small width.

087- Width too large. The maximum will be used (<value>).

   The width of a line have some restiction to enable lk to run finely. This
warning will prompt you if you are asking for a too large width.

088- Unknown width keyword.

  WIDTH instruction may be followed by one the next keywords:

        OBJECTNAME
        UNITNAME
        HUNKNAME
        SYMBOLNAME

  any other keyword will generates this error.

089- Height too small. The minimum will be used (<value>.)
```

The height of a page have to be consistant enough to enable lk to create the
external file. Anyway a value of zero is a valid value which disable page usage.

090- Height too large. The maximum will be used (<value>.)

The height of a page cannot be larger than the field to hold it.

091- Unknown margin keyword.

MARGIN instruction can be followed only by next keywords:

        LEFT
        RIGHT
        TOP
        BOTTOM

Themselfes must be followed by a value.

092- Invalid margin value. It must be defined between 0 and 255 included.

The margin are saved into a byte. What a limitation!

093- Bad map letter (See documentation.)

A letter specified after MAP keyword is invalid and must be suppressed.

See also:
                                MAP

094- Base name defined twice in FD file at line #<value>.

The base name is defined twice for the same library. You might verify if you
use the ##end instruction of the previous library.

095- No base defined in FD file at line #<value>.

A base definition is required before any function definition.

096- References "<symbol name>" defined with two different types ($<value> and
     $<value>.)

A references was defined in two different hunks with two different types. This
means you might have some errors, because one of those two references may be
wrong. lk will check, whenever possible, to avoid the emission of this warning.
Not that this warning is more an information than an error.

097- COLOR flag defined twice.

This warning can be avoid using QUIET instruction.

098- The base offset is not a multiple of 6 in FD file at line #<value>.

  The base offset into a library should always be a multiple of 6. Because the
Amiga system uses only the 68000 JMP instruction which uses 6 byte per function
call.

099- Base value overflow in FD file at line #<value>.

  The base value can be defined between -32767 and 32767 and should be a
multiple of 6.

100- Left/Right members differ for function <function name> in FD file
      <filename> at line #<value>.

  lk verify that the left and right members have the same number of parameters
(Than you have the same number of named parameters than registers.)
  This is a warning become libraries using floating points might use less named
parameters than registers.

101- Invalid DREL32 in executable file.

  This warning will alarm you about the existance of a bad hunk into an
executable file. This may happen with programs which are compatible with the V37
of DOS.
  You can avoid this warning using the instruction:
                              READSRV37

102- Invalid debug skipped.

  A debug can not be handled by lk when some data inside are wrong. In that
case, those debugs are just suppressed.

103- Different memory flags in header and hunk.

  A memory type was defined into the hunk list and that type is not equal to the
one defined into the hunk declaration. Some compilers/linkers save the memory
flags in the hunk it-self, use the HUNKMEMORY command to do the same.

  Only the memory type of the hunk list is keept.

  Note: The hunk is supposed to have the type PUBLIC, also this is not checked
against other values.

104- Hunk type conflict for relative links in file "<filename>" (CODE/DATA).

  Any relative information (HUNK_DREL or EXT_DREL) must be used with a link to
the same type of hunk (also supposed to be DATA.) In future releases I may check
the used register to know if CODE or DATA is accessed (For instance, A4 accesses
the DATA and A5 the CODE.)

105- No library hunk before HUNK_INDEX.

   No HUNK_LIB exist before to find a HUNK_INDEX. lk cannot find the code, data
and bss header this way. Your file is corrupt and cannot be used.

106- No overlay header needed without overlay.

   No overlay file was defined, then no overlay header has to be.

107- Math conflict between hunks.

   SAS let's some debug information about the math library to use. lk know how to
check this and also emit an error when two files uses incompatible format. This
check does not occur while linking two libraries together.

108- Integer conflict between hunks.

   SAS let's some debug information about the integer library to use. lk know how
to check this and also emit an error when two files uses incompatible format.
This check does not occur while linking two libraries together.

109- Bad hunk for executable file. It contains memory informations.

   This error should never happen, because it is forbidden to write memory
requirements into hunks within executable files. Those requirements should
appear only into the header list.

   Note: some compiler/assembler and linkers produce code with such information.
lk has an option (HUNKMEMORY) to do it, if required.

110- Two overlay object handlers defined.

   Only one object can be defined as being the overlay handler. See the overlay
function to have more information.

   See also:
                                  OVERLAY

111- Libraries should not be defined in overlay.

   This warning tell you that you use the LIBRARY keyword within the overlay/end
block definition. Anyway the libraries will be loaded correctly.

112- The overlay link '<label name>' not within a hunk of code.

   lk will link only JSR/JMP and Bcc instructions between the generic program and
different overlay units. And instructions stands only in hunks of codes.

113- lk cannot work with odd pointer with your CPU.

   The flag ODD is available only on 68020 CPUs and over. This would require a
special programing to handle odd pointers on 68000.

114- Relative relocation not in a CODE hunk. No ALV possible.

   When relative relocation exist and generates an overflow, lk will try to
create some code to enable the link to work. This might be an enormous code
generation, but it works in most cases.
   Anyway lk will be quiet in any hunk which is not of type CODE. lk cannot
modify data to fits a requirement. Thus data hunks should contain only data no
code has to be modified in there. But the instruction ANYRELATIVE will make lk
smarter and let him work in data hunks. This may just produce a lot of problems.
Note that BSS hunks will still generates this error.

   See also:
                             ANYRELATIVE

115- The overlay link '<label name>' is used with an unknown instruction.

   The link to that label into a different overlay unit is not done while it does
not refer to a valid instruction. The supported instructions are the JSR/JMP and
Bcc.

116- Relative relocation "<symbol name>" uses an unknown instruction. lk cannot
     create an ALV.

   In the first release lk supports only Bcc and JSR/JMP d16(PC) instructions.
This
will be enlarged to any d16(PC) addressing mode in time.

   Note that the usage of ALVs might generates some bugs.

117- Hunk too large to create an ALV for "<symbol name>" (<offset>.)

   lk cannot create an ALV for the the symbol <symbol name> because the hunk is
so large than the beginning and the end of the hunk are both too far.

118- Unknown address keyword.

   ADDRESS keyword must be followed by CODE, DATA or BSS keywords.

119- No destination file deleted while some errors occur.

   Some errors occur while lk saved the result. This means your make program file
may not deal with the next operations correctly.

   This may happen only if you fix the hunk addresses.

120- BSS hunk has some relocation information in file: "<file name>".

  Signal that a BSS has some relocation informations. This means it should not
be saved just as a four bytes file but as a file of zeroes except for the
relocated addresses.

  This can occur only if the BSS had a reloc to the beginning of another hunk.

121- Address $<hexa value> is odd.

  DATA and BSS hunks might be relocated at a odd address, but code should never
be at a odd address on any 680x0 CPUs. Make your change.

122- The code header was moved during relative organization.

  To permit relative to be linked properly, lk try to organize hunks which are
accessed as relative. This will enable 8 bit relatives the most of the time.

  This warning should never happen, while the code header should never be used
as a 'relativly pointed hunk.'

123- Icon file name defined twice (<file name>/<file name>).

  Only one icon file can be defined while there is only one destination file.

124- <file name> is not a valid icon file name.

  The file name of the icon file should have the usual '.icon' at the end. This
ensure the usage of the 'icon.library' functions.

125- Fragmented <type> can not be SMALL or SINGLE...

  A FRAG instruction has been used with an SMALL/SINGLE instruction of the same
type. Only the FRAG flag is keept. Note that '-frag' is equivalent to FRAGBSS,
FRAGCODE and FRAGDATA all together.

126- Hunk of type CODE named "<hunk name>". This hunk will not be merged this
     way.

  A hunk of code can not receive one of the 'merge' name. Those are:
     MERGE, MERGED, _MERGE, _MERGED, __MERGE, __MERGED
  The hunk is not merged to anything except if SMALLCODE was used.

  See Also:
     Hunk names

127- Invalid entry hunk "<hunk name>". Not of type CODE.

  An entry hunk must be a hunk of CODE, or it cannot be an entry.

128- Multiple entry hunks, skip the last ("<hunk name>".)

   Only one entry is available per program. Then only the first fiven entry will
be keept.

129- Entry hunk can not be in an overlay ("<hunk name>".)

   You can not have the entry of the program within the overlay. It has to be
included in the root.

130- Be careful, MAKERELATIVE may create an invalid program.

   This warning is just to remenber you that the instruction MAKERELATIVE is not
conventionnal... and may generate some errors.

131- Overlay handler found in overlay hunks ("<hunk name>".)

   The overlay handler must be in root program. It will be forced at start by lk
but not if present in overlay.

132- ONEDATA should be used with [CLEAR]XDATA and SMALLDATA.

    This warning is to remember you that ONEDATA option is better used when used
with XDATA or CLEARXDATA and SMALLDATA.

133- Auto-Run is not allowed with overlaid program.

   Overlaid programs keeps a lock and a handle from the executable file. This
means you have to remains linked to it. If you want to play with overlaid
programs and an auto-run, change the 'overlay.s' code to handle that logic.
(Note: this is something 100% possible.)

134- Symbol "<symbol name>" defined into library "<library file name>" and
      object "<object file name>".

   This warns you about a symbol which is redefined into your object files. This
might be a problem. That warning level is 8, which means that you can easily
hide it.

135- The symbol "<symbol name>" is already defined, lk can not create the
      array.

   The symbol '___ctors' or '___dtors' has been defined by user. Thoses symbols
are strictly reserved for the linker internal use. You must delete that
definition and recompile and relink.

136- Error file name defined twice (<file name>/<file name>.)

Only one file can be used as the destination for error strings.

137- Relocation(s) pointing on the near data hunk.

  You can have as much relocations as you want except relocations to the
relative data hunk when you link to produce a resident executable for DICE or
SLINK. Only the first hunk in file 'c.o' or 'cres.o' have the possibility to
reference the near data hunk. After, the near data hunk is suppositivly in an
allocated buffer.
  SLINK accepts relocations from the near data hunk to the near data hunk. DICE
does not accept those relocations.
  This error can usuly be recovered using the SMALLDATA instruction. Otherwise
you will have to find another way to create the links. For instance you can
create an initialisation function which will set some far data variables to near
data pointers.

  Note: SMALLDATA works only if you have a single memory model.

138- The relative "<symbol name>" is defined in a hunk of code.

  This is not normal to have relative symbol pointing in any other hunk than
data and BSS. This warning remains you about that problem.

139- Overlaid file with a corrupted first header.

  Your overlay file is no correct. This does not means it does not work, but lk
will change that into a file a little bit more valid. This error usuly happen
when the very first header does not say that the file is overlaid, however an
overlay hunk exists...

140- Overlay hunk number larger than indicated in first header.

  Into an overlay file you have several hunk_header. If one of them gives a hunk
number larger than the one which is defined into the very first header, this
warning will prompt you.

141- lk found more hunks than indicated in first header.

  Each hunk has a number. When too much hunks appear in entire file, lk tell you
about that 'overflow.' If you use overlaid program, you can consider this error
as a very low level warning.

142- Invalid overlay table size.

  All overlay table sizes I saw up to now are invalid (Except those of lk.) Then
this warning has a level of 4.

143- The number of overlaid unit changed.

When you link an executable which is defined in overlay, lk may change the
number of overlay unit, also creating an invalid overlay table.

144- From/Root "<file name>" should not be defined in overlay.

  The given file name will be used as the root file, but you should extract its
definition from the overlay/end block.

145- Value missing for "<keyword>" instruction.

  The <keyword> instruction needs a value to be correctly defined. Have a look
to the documentation to know the exact purpose of that value.

146- Label "<symbol>" is undefined, lk cannot create the Slink array.

  lk needs '_ctors,' '_dtors' and '_construct' symbols to be able to build the
constructors and destructors for Slink.

147- All of the relative DATA/BSS hunks have not the same memory requirements.

  Relative (or near) data should all be in the same memory format. It says
PUBLIC. However lk does not require you to put those data in a specific memory
model. But lk generate this warning when all hunks have not the same memory
requirements. The warning has a level of 12.

148- String missing for "<keyword>" instruction.

  The <keyword> instruction must be followed by a string. You should refer you
to that instruction to know the purpose of the string.

149- List file name defined twice (<file name>/<file name>.)

  The file name for the list output has been defined twice. Only the first one
will be used.

150- "<file name>" is not a valid FD file (line #<value>.)

  A file defined with FD, FDLIB or LIBFD instruction is not correct. The syntax
of FD files is explained into:
                                        FD

151- Create an ALV for "<symbol name>" symbol.

  This warning will appear only if the warning level is 4 or less. This might be
useful if you want to modify the order of your files or change some call from
near to far, etc...

152- Two library definition files defined (<file name>/<file name>.)

   The instruction LIBFD has been defined twice. It cannot be. Only the first
file name is used.

153- Two library prefixes defined (<prefix>/<prefix>.)

   Two prefixes has been defined, only the first one will be keept and used by
lk.

154- Invalid icon type (Type: "<type name>".)

   The icon has a type which does not match the required type. The type must be
TOOL for executables and also PROJECT for libraries.

155- This file is not an icon.

   The icon file is not correct. lk supports all icons from V1.x to V3.x. I
suppose icons will not change.

156- Invalid stack size, the minimum required is <value>.

   The Amiga system requires a minimum of about 4Ko of stack; this minimum is
also repected by lk and the majority of good C compiler. When an invalid stack
is defined, lk readjust it to the minimum.

157- The value <value> is not a valid priority (Bounds -128 and 127.)

   The value of the priority is no valid because this must be signed byte (a
value of height bits maximum.) When an invalid priority is defined, the previous
value is keept.

1.42   lk V1.04 - (Errorfile)

                                                                  Page 11-35
                        ERRORFILE or VERIFY or VER <filename>
               Default: standard I/O or '>>filename' on command line

   Defines the destination error file name. That file will be created only if lk
generate an error or more. If the file cannot be created, the errors will be
sent to the usual output console.

   See also:
      Become Registred

1.43   lk V1.04 - (Fancy)

```
                              FANCY
                         Default: FANCY
```

   Enable all printer/console characters (CSI) to be inluded in XREF file. To
forbid those characters, use PLAIN keyword.

```
   See also:
      HEIGHT
      MAP
      MARGIN
      PLAIN
      SWAPFH
      WIDTH
      XREF
      Become Registred
```

1.44   lk V1.04 - (Fast)

```
                               FAST
                         Default: DEFAULT
```

   Link all hunks of the following object files to be automatically loaded in
fast memory. This is done regarless of any information present in those file
hunks. It will last with the next memory instruction (ATTRIBUTES/CHIP/DEFAULT or
PUBLIC.)
   You may avoid the usage of the FAST keyword using the AUTOFAST command.
Please, refer to that instuction.

   Because this is a user specification, lk will not send any warning.

   This instruction can appear into the OVERLAY/END block.

```
   See also:
      ATTRIBUTES
      AUTOFAST
      CHIP
      DEFAULT
      <filename>
      HUNKMEMORY
      PUBLIC
      Become Registred
```

1.45   lk V1.04 - (Fd)

```
                         FD [<filename> ...]
                         Default: OBJECT
```

   Reads the following files as description or '.fd' files to create a list of

XDEF from the function names. You may avoid the usage of this instruction by
using the AUTOFD instruction. Please, refer to that instruction.

   The '.fd' files have the following format:

       ##base <library name>
       ##bias <value>
       * <comment>
       ##public
       <function name>([<parameter name>,...])
                                                       ([<register>,...])
       ##private
       <function name>([<parameter name>,...])
                                                       ([<register>,...])
       ##end

   Any number of function may appear. The coma (,) may be replaced by a slash
character (/) and the multiply (*) sign is used as a comment delimiter. The
public/private switcher can be used any number of time as required. The value
defined behind '##bias' represent the first function's offset, which is 30 by
default. All private functions may have the same names, except if you are
creating an Amiga library. No spaces can appear within the function definition.

   Note: the function offset can be modified through '##bias' instruction as
required, except if you are creating an Amiga library.

   See also:
       AUTOFD
       FDHEADER
       FDINCLUDE
       FDLIB
       FDOBJECT
       FDPASCAL
       FDQUICK
       FDSMALL
       <filename>
       LIBFD
       LIBRARY
       Become Registred

1.46 lk V1.04 - (Fdflg)

                                                                 Page 11-39
                              FDHEADER
                             FDINCLUDE
                              FDOBJECT
                              FDPASCAL
                              FDQUICK
                              FDSMALL
                          Default: no flag set

   Only the commercial version will have all those instructions fully supported.
Registred people will receive that version.

   '.fd' files might be used in a lot of different ways. There is a list of the

```
available flags and their usage:

  FDHEADER
     This flag enables the creation of a C like header file. When  this  flag  is
used with the FD instruction, a list of offset is  created  with  the  following
format:
                       #define _LVO<function name> <offset>
     When this flag is used with  the  FDLIB  instruction,  a  list  of  external
instruction call is created. lk cannot recognize the  type  of  each  parameter,
then this can be used as a  first  step  to  a  header  creation.  The  external
functions have the following format:
                      extern _<function name>(<parameter 1>[, ...]);
     You may use the FDQUICK flag to suppress any parameter definition.

  FDINCLUDE
     This flag enables the creation of an include file. It can be used only  with
FD instruction, a list of offset is create with the following format:
                      _LVO<function name> EQU <offset>

  FDOBJECT
     lk creates an object from the '.fd' file to use it as any other object. This
flag will enable you to save that object and use it rather than the  '.fd'  file
(Making the future usage quicker.)
     When used with the FD  instruction,  only  offsets  are  saved  as  external
definitions.
     When used with the FDLIB instruction, lk creates a small call code to enable
any C language like to call the library.

  FDPASCAL
     This flag enables a Pascal type header to be created from  the  '.fd'  file.
When it is used with the FD instruction only a list of offset  is  defined  with
the following format:
                      const _LVO<function name> = <offset>;
     When used with the FDLIB instruction, lk creates a sub-functionnal header of
external definitions with the following format:
                      extern _<function name>(A: <parameter1>[, ...]);

  FDQUICK
     This flag can be used to avoid  the  parameter  list  into  the  C  external
definitions (See also FDHEADER flag.)

  FDSMALL
     Ask for a small code into the current library. This will force lk to use the
instruction:
                      MOVEA.L   <library base offset:16>(A4),A6
                                rather than:
                      MOVEA.L   <library base pointer:32>,A6
This makes library calls smaller and a bit quicker.

  See also:
     FD
     <filename>
     FDLIB
     LIBRARY
     Become Registred
```

1.47   lk V1.04 - (Fdlib)

```
                        FDLIB [<filename> ...]
                           Default: OBJECT
```

   This function must be used with C program likes, it generates a library (.lib
file) from a '.fd' file.
   It also reads the following files as '.fd' files. Please refer to FD
instruction to have complete informations about the '.fd' file syntax.

   The resulting code is something like:

```
   MOVEM.L  ...A6,-(A7)
   MOVEM.L  ...(A6),...
   MOVEA.L  <library base>(A4),A6
   JSR      <function offset>(A6)
   MOVEM.L  (A7)+,...A6
   RTS
```

   Note: A4 is used only when the flag FDSMALL has been used.

   See also:
      FD
      FDHEADER
      FDINCLUDE
      FDOBJECT
      FDPASCAL
      FDQUICK
      FDSMALL
      <filename>
      LIBFD
      LIBRARY
      Become Registred

1.48   lk V1.04 - (Filename)

```
                         <filename>
```

   You directly can start your command line with the name of an object file or an
executable. lk is in DEFAULT state at start. You may see that instruction for
more informations:
```
                           DEFAULT
```

   The file types are defined in following table:

```
 Precedent Keyword                 File Type
<none> .................... (s) multiple objects
ATTRIBUTES <value> ......... (s) multiple objects
CHIP ...................... (s) multiple objects
DEFAULT ................... (s) multiple objects
ERRORFILE ................. (d) error output
FAST ...................... (s) multiple objects
```

```
FD ....................... (s) multiple deffiles
FDLIB .................... (s) multiple deffiles
FD/FDLIB + FDHEADER ....... (d) include file
           FDINCLUDE ...... (d) include file
           FDOBJECT ....... (d) library object
           FDPASCAL ....... (d) include file
FOR ...................... (d) object
FROM/ROOT ................ (s) object or exec.
ICON ..................... (s) icon (.info)
LIBFD .................... (s) one deffile
LIBRARY .................. (s) object
LIST ..................... (d) list output
OVERLAY/END .............. (s) multiple objects
OVERLAYOBJECT ............ (s) object
PUBLIC ................... (s) multiple objects
TO ....................... (d) executable
WITH ..................... (s) lk source file
XREF ..................... (d) reference file
```

        (s) stands for source file (Always safe)
        (d) stands for destination file (Will be overwritten)

   Other keywords cannot be followed by a file name (Note that equivalent
keywords were not listed here.) See each of them for more explanations.
   The reference, list and error files are text files.

   With AmigaDOS V36.00 and over, a keyword followed by multiple sources may use
a file name containing wild cards. All normal DOS wild cards are supported. The
research is also made into each directory of a multiple assign and the assign
might be defered.
   A keyword, which has to receive only a single source file, will generate a
warning (Except with QUIET), and keep only the first file responding to the wild
card(s).
   Only source file names can include wild card(s).

   IMPORTANT NOTE ABOUT WILD CARDS:

   The next command line is fully supported:
     LK FROM main.o #?.o TO myprog LIB amiga.lib

even the pattern #?.o will return main.o, lk will not link that file twice.
However, only FROM and ROOT instructions enables a such comparaison. For
'CHIP/FAST programs' it should become:
     LK ROOT main.o #?.o CHIP chip/#?.o
                                       FAST fast/#?.o TO myprog LIB amiga.lib
   or
     LK ROOT main.o #?.o CHIP #?.chip
                                        FAST #?.fast TO myprog LIB amiga.lib

(Note: The FROM/ROOT instructions are declared before '#?.o' and the AUTO???
keywords could be used instead of CHIP/FAST keywords, thank you to refer you to
those commandes.)

   For any source file, lk uses paths to read files without patterns. For more
informations about paths, please refer you to:
                                  PATHS

   lk have a generator for the destination file name. You can refer to FROM/ROOT
                              for more informations.

   The memory requirement or advisory state must appear before a file name to be
effective. You can change those as much as you want. The following are possible
                                 usages:
                      CHIP SETADVISORY <filename>
                   FAST CLEARADVISORY ROOT <filename>
                      HUNKADVISORY ATTRIBUTES $10040

                                                       LIBRARY <filename>

   See also:
       ATTRIBUTES
       AUTOCHIP
       AUTOFD
       AUTOFAST
       AUTOLIBRARY
       AUTOOVERLAY
       CHIP
       CLEARADVISORY
       DEFAULT
       ERRORFILE
       FAST
       FD
       FDHEADER
       FDINCLUDE
       FDLIB
       FDOBJECT
       FDPASCAL
       FDPATH
       FOR
       FROM/ROOT
       ICON
       ICONPATH
       HUNKADVISORY
       LEFTADVISORY
       LIBFD
       LIBPATH
       LIBRARY
       LIST
       NOPATH
       OBJPATH
       OVERLAY/END
       OVERLAYOBJECT
       PUBLIC
       SETADVISORY
       TO
       WITH
       WITHPATH
       XREF
       Become Registred

1.49   lk V1.04 - (For)

                          FOR <filename>
                          Default: TO

   Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

   Generates a library with specified file name. This will load each object and
order hunks and units to produce a library object file. You may ask for
SINGLEUNIT to have a universal library.

   When specified, the instruction SMALLUNIT will enable a link of some hunks
together. This is done when hunk A needs hunk B and hunk B needs hunk A (This
may be more complex while hunk A may need hunk B going through hunk C and D, and
hunk B may need hunk A through hunk E and F.) Hunks of different types are not
linked together, whenever they need each other.

   If the given file name is a directory name, the usual auto creator file name
will be used and the result will be saved into the given directory. The source
file name will have its path suppressed before the concatenation to the source
file name.
   Example:
     FROM prg:object/myprog.o
     FOR ram:
   generates the destination file name:
     ram:myprog.lib

   See also:
       CREATELIBRARY
       <filename>
       SINGLEUNIT
       SMALLUNIT
       Become Registred

1.50   lk V1.04 - (Frag)

                              FRAGBSS
                              FRAGCODE
                              FRAGDATA
                               -FRAG
                          Default: NORM???

   This instruction forbids any link between hunks, even their names are equal.
This will also generate a big resulting file. Those instructions are useless for
libraries.
   Note that small data code (HUNK_DATA access with A4 register) will anyway be
linked together or there would be no use of this function.

   Note: the instruction -FRAG ask for all type of hunk to be fragmented. This is
to ensure a compatibility with DLINK.

   See also:
       NORM

```
    NORMBSS
    NORMCODE
    NORMDATA
    NORMHUNK
    NORMUNIT
    SMALL
    SMALLBSS
    SMALLCODE
    SMALLDATA
    SMALLHUNK
    SMALLUNIT
    SINGLE
    SINGLEBSS
    SINGLECODE
    SINGLEDATA
    SINGLEHUNK
    SINGLEUNIT
    Become Registred
```

1.51   lk V1.04 - (From)

FROM or ROOT <filename>
Default: first file specified

   Defines the first file, which should start with a hunk of type code. This will
be linked at start in destination executable file (Usuly a C or pascal startup
or main assembly code.) The first file is not always the first if you use one of
the instructions STARTUP or OVERLAY (Please refer to those instructions for more
informations.)
   The use of FROM instruction returns lk in default OBJECT mode. Please see that
instruction for more informations.

   When no destination is defined or only a directory is given, the root (or
first) file name is used as the destination file name (Erasing the extension
'.o' or '.exe' and eventualy appending '.exe' when a source file have that
name.) If the root (first) file is 'c.o', 'cback.o', 'cres.o', 'catch.o' or
'catchres.o' the following file name will be used instead, enabling   C
programmers to avoid the TO instruction.

   In the case of a library, one of this instruction may be used only to have the
destination file name automatically computed. Also the specified name will
receive the usual '.Lib' extension. Note this may not always be supported.

   All following files will be defined though DEFAULT keyword was used (See.)
There is no problem to use wild cards in those file names, lk will automatically
check and prevent the ROOT file to be loaded twice.
   All hunks of each file will be present into the result.

   If you need to put the root into a specific memory type you will have to
specify that type at first and then use the instruction ROOT. For instance to
put the root in chip memory:
    CHIP ROOT my_root_file

   See also:

```
      DEFAULT
      <filename>
      OBJECT
      OVERLAY
      STARTUP
      Become Registred
```

1.52   lk V1.04 - (Height)

```
                         HEIGHT <value>
                       Default: 60 (10..255)
```

   Defines the page height. The height does not include the top
and bottom margins and the four lines used for foot and header
titles. A height of zero (0) will elimites any pagination.

```
   See also:
      FANCY
      MAP
      MARGIN
      PLAIN
      SWAPFH
      WIDTH
      XREF
      Become Registred
```

1.53   lk V1.04 - (Help)

```
                   lk V1.04 -- Help about Help
```

   There is a skeleton of the organization of my help files. This hopefully will
help you a little bit more to quickly find the requiered information.

```
   1.  About

  2.  Generic information

     2.1  Purpose
          (For beginners)

     2.2  Release informations
          (For those who know about lk©)

     2.3  Command
          (For advanced users)

     2.4  Start

        2.4.1  CLI
```

1.54   lk V1.04 - (Hlp)

```
                        HELP or -HELP or ?? or ?
                            Default: nothing
```

   Print the complet list of available instructions with a quick reference. This
is helpful if you forgot about a specific keyword. The single question mark will
list only important keywords.
   After the help has been displayed, you will enter the prompt mode. Please
refer to the PROMPT instruction.

   This keyword should never be used inside a WITH file.

   See also:
      PROMPT
      VERBOSE
      Become Registred

1.55   lk V1.04 - (Hunk)

```
                            HUNK <value>
                            Default: 50
```

   Defines the number of hunk structures needed. The default is 50. Like SYMBOL,
the number of hunk used by a link may be known using VERBOSE keyword.

   The default value is really small if you use some libraries.

   See also:

```
    RELOC
    SYMBOL
    VERBOSE
    Become Registred
```

1.56   lk V1.04 - (Hunkcaseinsensitive)

```
                      HUNKCASEINSENSITIVE or HCI
                Default: hunk names are case sensitive
```

   Test hunk names regardless of character case. Any letter with and without
accent will be tested in upper case.

```
   See also:
      CASEINSENSITIVE
      Become Registred
```

1.57   lk V1.04 - (Hunkmemory)

```
                           HUNKMEMORY
                Default: no memory flags in hunks
```

   Force the memory requirements to be copied into the hunk declaration. This is
useless as long as I know. Some programs (like basm) does it, also I propose the
option to do the same.

   Note that when a memory requirement appear into the file it take 4 bytes,
repeated twice it take 4 more bytes for nothing I know about... and the
executable is not valid.

```
   See also:
      ATTRIBUTES
      CHIP
      DEFAULT
      FAST
      PUBLIC
      Become Registred
```

1.58   lk V1.04 - (Hunknames)

```
                        The special hunk names
```

   To enable lk to link properly any kind of program, special hunk names may be
used. You have to be careful when you use those name, while they are specific to
lk.

```
   BSS
```

This is the default name for a hunk of type BSS. You should use this name,
except if you want specific hunks of BSS (Hunks to be loaded in CHIP memory
for instance.)

COVER or _COVER or __COVER

  By default, lk does not support this special hunk. You must use the BOUNDS
instruction in order to generate the startup pointer and the size. Please
refer to this instruction for more informations:
                                  BOUNDS

DATA

  This is the default name for a hunk of type DATA. You should use this name,
except if you want specific hunks of data (Hunks to be loaded in CHIP memory
for instance.)
  BSS hunks should not be called DATA, while they can not be linked with DATA
hunks.

ENTRYHUNK or NTRYHUNK or _ENTRYHUNK

  This name is useful to define the very first hunk of a program (The second
name is implemented to accept Blink/Slink special hunk.) This hunk must be of
type code and have to be a valid startup.
  With this hunk, you should not use the instruction STARTUP. Anyway the
STARTUP label override this hunk.

MERGE or MERGED
_MERGE or _MERGED
__MERGE or __MERGED

  Only DATA and BSS hunks can receive that name and all of them will
automatically be linked together, even ONEDATA is unused. Note that the option
XDATA is still needed and recommended. Hunks of DATA or BSS with one of those
names which are present into an overlay hunk will automatically be extracted
from the overlay hunk. Do not forget that when a FRAG??? instruction has been
used, no merge is effective for the specified types.
  This name is used by Blink and Slink to force relative hunks to be linked
together. lk does not need that extra information and is intelligent enough to
know when that kind of hunks have to be linked together.
  The number of underscore defines a new set of merged block.
  Note: A hunk of code with that name will generate a warning, and will not be
linked with data hunks by any means.

NOMERGE or NOMERGED or _NOMERGE or _NOMERGED

  Any hunk may have this name. This way it can not be merged with another
hunk. This is useful to create a hunk that has to be freed or when a linked
list is necessary. This is used for the background task startup code.

OVERLAYENTRY

  This defines the overlay handler. This hunk must be of type code and be a
valid overlay startup, thus have the overlay magic long word $0000ABCD and the
following null pointers (4) for the overlay table, file stream and more.
  This hunk will call one of the following function. The order is important is

```
that it's check into that given order:
    1. '_ovl_root' (May not exist)
    2. function defined with STARTUP instruction
    3. first function of ENTRYHUNK hunk
    4. the first hunk of CODE
```

RDATA

   This is the default name for a hunk of type DATA used with read only
operation. This might be used to automatically protect the program when loaded
with the special MMU loader of lk.
   Note: this loader is not actually available.

TEXT

   This is the default name for a hunk of type CODE. When the first given hunk
is not of type CODE, lk will search the first hunk named TEXT and moved it at
start.

```
See also:
    FRAGBSS
    FRAGCODE
    FRAGDATA
    ONEDATA
    OVERLAY
    STARTUP
    Become Registred
```

1.59   lk V1.04 - (Icon)

```
                 ICON or INFO <filename>
                      Default: NOICON
            (DEFICON if lk is started from the Workbench)
```

   Define the icon you want to use for your executable. This file will be read
and copied into the file named like the destination with the extension '.info'.
A check is done on the icon file to ensure a correct type. An icon for an
executable must be of type TOOL and an icon for a library muts be of type
PROJECT. If this is not the case a warning of level 15 is displayed and the
correct type is set into the icon.
   If the keyword ICON does no appear and the file is a correct icon, it will be
reconized and used as an icon.

   When the icon file name is terminated with '.info' lk will try to make the
copy through the 'icon.library' also enabling the notification message of the
workbench.

   When a icon is specified, it will overwrite the existing icon.

```
See also:
    DEFICON
    NOICON
    UNSNAPSHOT
```

Become Registred

1.60   lk V1.04 - (Keeparc)

                                 KEEPARC
                 Default: delete files extracted from archive(s)

   If you use the archived library feature, you may want to keep the uncompacted
files in memory for future links (do not forget that each compacted library is
extracted from an archive and it takes a lot of time.)

   Note: you should not use that function if you installed the QuickDOS© shared
library, because the extract files are automatically saved through QuickDOS©.

   For more information about the archive feature of lk, please   refer
to~'Archived Libraries'.

   See also:
      ARCHIVER
      Archived Libraries
      QuickDOS© V1.00
      Become Registred

1.61   lk V1.04 - (Keepdebug)

                             KEEPDEBUG or KD
                   Default: keep the debugs which are known

   When this instruction is used all debugs, even if their are not known, will be
keept and saved into the destination file. The unknown debug hunks remains
unmodified into the destination file, then most of the time it will certainly
create invalid hunks of debug.

   The instructions NODEBUG and NOLIBDEBUG have a higher priority and will be
taken in account rather than KEEPDEBUG.

   See also:
      ADDSYM
      CLEARADVISORY
      CREATEDEBUG
      CREATESYMBOL
      HUNKADVISORY
      LEFTADVISORY
      NOBSSDEBUG
      NOCODEDEBUG
      NODATADEBUG
      NODEBUG
      NOLIBDEBUG
      NOLOCALSYMBOL
      NOSYMBOL

```
        NOOVLDEBUG
        SETADVISORY
        STRIPDEBUG
        Become Registred
```

1.62 lk V1.04 - (Keywords)

Page 11
Keywords

   All keywords actually available in lk are listed below. Click on any of them
to receive the corresponding informations. For more generic informations about
keywords see also:
                            Command Line

   A very large amount of keywords is available. For this reason I propose two
kind of listes. One with all commandes defined in alphabetical order and one
list for each type I could think of. Click on the following to reach one of
those listes:

        Complete alphabetical list

        Amiga libraries (shared)
        Automation
        Compatibility
        Copyright
        Debug
        DICE flags
        Error
        File description
        File type
        Help
        Hunk attributes
        Map
        Memory (at link time)
        Object library
        Overlays
        Paths
        QuickDOS
        ROM
        Sizes
        Workbench

   In the following documentation a special syntax has been used to indicate
different ways to use arguments. For this purpose specific characters are used,
those characters will never appear in the final command line. On the top of
that, the upper case words are reserved (they cannot be used as a file name for
instance) and the lower case words represent the words that you have to replace
with the correct name. The used characters are:

<>   Angle brackets enclose arguments that must be provided. For instance,
        <filename> means that you must enter the appropriate filename in that
        position. Unless square brackets surround the argument (see below), the
        argument is required. lk will not work as you want unless it is
        specified. Note that only arguments of commands are necessary, a

command is never necessary.

[]   Square brackets enclose arguments and keywords that are optional. They
          will be accepted by the command but are not required.

...   Three points means that the previous option can be repeated any number
          of times.

or   The keyword 'or' is used to separate commands or options of which you
          can choose only one.

   In the following list a star (*) shows the startup state (Instructions set by
default.)

    Keywords:
    ADDRALL
    ADDRESS
    ADDSYM
    ALV (ALVS)
    AMIGALIBRARY
    ANYRELATIVE
    ASKUNDEF
    ATTRIBUTES (ATTR)
    AUTOCHIP
    AUTOFAST
    AUTOFD
    AUTOLIBRARY (AUTOLIB)
                                                (AUTOLIBRARY is set with DICE)
    AUTOOVERLAY
    AUTORUN
    BACKGROUND
*   BLOCKHUNK (BH)
    BLOCKUNIT (BU)
                                             (BLOCKUNIT is set with DICE or SLINK)
    BREAK
*   CALM
    CASEINSENSITIVE (CI)
    CC
                                                (CC is set with DICE or SLINK)
    CHIP
    CLEARADVISORY
    CLEARXCODE (CXC)
    CLEARXDATA (CXD)
    CODE2BSS
    COLOR
    COPYRIGHT (CR)
    CREATEDEBUG (CD)
    CREATELIBRARY (CL)
    CREATESYMBOL (CS)
    DATA2BSS (D2B)
*   DEFAULT
    DEFICON
    DEFINE (DEF)
    DICE
    ECHO
    END (#)
    ERRORFILE

```
*   FANCY
    FAST
    FD (Offsets definition)
    FDHEADER
    FDINCLUDE
    FDLIB (Function definition)
    FDOBJECT
    FDPASCAL
    FDPATH
    FDQUICK
    FDSMALL
    <filename>
    FOR
    FRAGBSS
    FRAGCODE
    FRAGDATA
    FROM
    FWIDTH
    HEIGHT
    HELP (??)
    HUNK
*   HUNKADVISORY
    HUNKCASEINSENSITIVE (HCI)
    HUNKMEMORY
    HWIDTH
    ICON
    ICONSPATH
    INFO
    KEEPARC
    KEEPDEBUG
    KEEPXDATA
    LEFTADVISORY (LA)
    LIBFD (To save function definition)
    LIBID
    LIBPATH
    LIBPREFIX
    LIBPRIORITY
    LIBRARY (LIB)
    LIBREVISION
    LIBVERSION
    LIST
    MAKERELATIVE (MR)
    MAP
    MARGIN
    MAXREF
    NEWOBJECTS
*   NOALV (NOALVS)
    NOBSSDEBUG (NBD)
    NOCODEDEBUG (NCD)
    NODATADEBUG (NDD)
    NODATAOVERLAID (NDO)
    NODEBUG (ND)
    NOEMPTYHUNK (NEH)
    NOEXE
*   NOICON (NOICONS)
    NOLIBDEBUG (NLD)
    NOLOCALSYMBOL (NLS)
```

```
      NOMULTIPLE (NM)
      NOOVERFLOW32
      NOOVLDEBUG (NOD)
      NOPATH
*     NORM
*     NORMBSS
*     NORMCODE
*     NORMDATA
      NORMHUNK
*     NORMUNIT
      NOSYMBOL (NS)
      NOWARNING (NW)
      OBJECT (OBJ)
      OBJPATH
      ODD
      OFFSET
      ONEDATA (OD)
      ORDER
      OVERLAY ... END
      OVERLAYOBJECT (OO)
      OVLYMRG
      PLAIN
      PRELINK (CL)
      PRIORITY (PRI)
      PROMPT
      PUBLIC
      PURE
      PWIDTH
      QDDISCARD
      QDLIST
      QDPREPARE
      QDREMOVE
*     QDSTART
      QDSTOP
      QUICKFATAL (QF)
      QUIET
      READSRV37
      RELOC
      ROOT (À partir de)
      SETADVISORY
      SETLKPRI
      SHORTRELOC (SR)
      SHORTRELOCOVERLAY (SRO)
      SINGLE
      SINGLEBSS (SGB)
      SINGLECODE (SGC)
      SINGLEDATA (SGD)
      SINGLEHUNK
      SINGLEUNIT (SGU)
      SLINK
      SMALL
      SMALLBSS (SB)
      SMALLCODE (SC)
      SMALLDATA (SD)
      SMALLHUNK
      SMALLUNIT (SU)
      SORT
```

```
    STACKSIZE
    STARTUP
    STRIPDEBUG (STRIP)
    SWAPFH
    SWIDTH
    SYMBOL
    <symbols>
    TIME
    TO
    UNSNAPSHOT
    USELASTDEFINE (ULD)
    VALIDNOP
    VERBOSE
    VERIFY (VER)
    VERSION
    WARNING
    WARNINGLEVEL (WL)
    WARUNDEF
    WIDTH
    WITH
    WITHPATH
    WRITESRV37
    XCODE (XC)
    XDATA (XD)
    XREF
    XRELATIVEDATA (XRD)

     Amiga libraries (shared):

    AMIGALIBRARY (AL)
    LIBFD
    LIBID
    LIBPREFIX
    LIBPRIORITY
    LIBREVISION
    LIBVERSION

     Automation:

    AMIGALIBRARY (For 'library.o')
    AUTOCHIP
    AUTOFAST
    AUTOFD
    AUTORUN (For 'autorun.o')
    AUTOLIBRARY (AUTOLIB)
    AUTOOVERLAY
    BACKGROUND
    KEEPARC
    PURE
    PRIORITY (PRI) (For 'priority.o')
    OVERLAY (For 'overlay.o')
    STACKSIZE (For 'stacksize.o')

     Compatibility:

    AUTOLIBRARY (AUTOLIB)
*   BLOCKHUNK (BH)
```

```
       BLOCKUNIT (BU)
       CASEINSENSITIVE (CI)
       CC
       DICE
       HUNKCASEINSENSITIVE (HCI)
       NEWOBJECTS
       NOEMPTYHUNK
       ODD
       OFFSET
       READSRV37
       SLINK
       SORT
       STARTUP
       USELASTDEFINE (ULD)
       XRELATIVEDATA (XRD)
       WRITESRV37

        Copyright:

       COPYRIGHT (CR)
       LIBID
       LIBREVISION
       LIBVERSION
       TIME
       TO (Use to define the file name)
       VERSION

        Debug:

       ADDSYM
       CREATEDEBUG (CD)
       CREATESYMBOL (CS)
       CLEARADVISORY
  *    HUNKADVISORY
       KEEPDEBUG (KD)
       LEFTADVISORY (LA)
       NOBSSDEBUG (NBD)
       NOCODEDEBUG (NCD)
       NODATADEBUG (NDD)
       NODEBUG (ND)
       NOLIBDEBUG (NLD)
       NOOVLDEBUG (NOD)
       NOSYMBOL (NS)
       NOLOCALSYMBOL (NLS)
       SETADVISORY
       STRIPDEBUG (STRIP)

        DICE flags:
       -chip
       -frag
       -help (-h/??/?)
       -l
       -ma
       -mw
       -o
       -p
       -r
```

```
    -s
    # (END)

     Error:

    ALV (ALVS) (Suppress a lot of errors)
    ASKUNDEF
    BREAK
*   CALM
    COLOR
    DEFINE (DEF) (Give equivalence)
    ERRORFILE
    MAXREF
    NOMULTIPLE (NM)
    NOOVERFLOW32
    NOWARNING (NW)
    ODD
    QUICKFATAL (QF)
    QUIET
    VERIFY (VER)
    WARNING
    WARNINGLEVEL (WL)
    WARUNDEF

     File description:

    FD
    FDHEADER
    FDINCLUDE
    FDLIB
    FDOBJECT
    FDPASCAL
    FDQUICK
    FDSMALL

     File type:

    END
    FD
    FDLIB
    FROM/ROOT
    KEEPARC
    LIBRARY
    NOEXE
    OVERLAY
    OBJECT (OBJ)
    ORDER
    WITH

     Help:

    ECHO
    HELP (??)
    LIST
    PROMPT
    VERBOSE
```

Hunk attributes:

```
    ATTRIBUTES (ATTR)
    CHIP
*   DEFAULT
    FAST
    HUNKCASEINSENSITIVE (HCI)
    HUNKMEMORY
    PUBLIC
```

Map:

```
*   FANCY
    FWIDTH
    HEIGHT
    HWIDTH
    MAP
    MARGIN
    PLAIN
    PWIDTH
    SWAPFH
    SWIDTH
    WIDTH
    XREF
```

Memory (at link time):

```
    HUNK
    QDSTOP
    RELOC
    SYMBOL
    SETLKPRI
```

Object library:

```
    CREATELIBRARY (CL)
    FOR
    PRELINK
    SMALLUNIT
    SINGLEUNIT
```

Overlays:

```
    END
    NODATAOVERLAID (NDO)
    OVERLAYOBJECT
    OVLYMRG
    OVERLAY
    SHORTRELOCOVERLAY
```

Paths:

```
    FDPATH
    ICONPATH
    LIBPATH
    NOPATH
    OBJPATH
```

WITHPATH

 QuickDOS:

QDDISCARD
QDLIST
QDPREPARE
QDREMOVE
* QDSTART
QDSTOP

 ROM:

ADDRALL
ADDRESS
SINGLEHUNK
SMALLHUNK
ONEDATA (OD)

 Sizes:

ALV (ALVS)
ANYRELATIVE
CLEAREDXCODE
CLEAREDXDATA
CODE2BSS
DATA2BSS (D2B)
FRAGBSS
FRAGCODE
FRAGDATA
KEEPXDATA
MAKERELATIVE
* NOALV (NOALVS)
NOEMPTYHUNK
* NORM
* NORMBSS
* NORMCODE
* NORMDATA
NORMHUNK
* NORMUNIT
ONEDATA (OD)
SHORTRELOC
SHORTRELOCOVERLAY
SINGLE
SINGLEBSS (SGB)
SINGLECODE (SGC)
SINGLEDATA (SGD)
SINGLEHUNK
SINGLEUNIT (SGU)
SMALL
SMALLBSS (SB)
SMALLCODE (SC)
SMALLDATA (SD)
SMALLHUNK
SMALLUNIT (SU)
VALIDNOP
XCODE (XC)

```
   XDATA (XD)
   XRELATIVEDATA (XRD)

    Workbench:

   DEFICON
   ICON
   INFO
*  NOICON (NOICONS)
   UNSNAPSHOT
```

   The following keywords exists and then don't generate an error, but they are
unused:
   BUFSIZE, FASTER, FNDATAM, IGNORE, NEWOCV, NOOCVHNK.

   For those who used the NEWOCV option of Slink: lk© is a lot more powerful
   while the flags returned by the user function are also returned to the caller,
   and the function parameters might be defined into the stack or in registers.
   And all this by default.

1.63   lk V1.04 - (Libfd)

```
                            LIBFD <filename>
                            Default: OBJECT
```

   This instruction defines the list file with the definition of the library
functions. This file has the usual library definition syntax; like defined
within FD instruction.
   The file is transformed into a table of function (defined into a hunk of code)
and will be linked with the pre-defined memory requirements. You should also
preceed LIBFD by DEFAULT if you do not want to have that table in a specific
memory.

   The fact this instruction is used set the flag AMIGALIBRARY. Please refer you
to that instruction to have all necessary informations.

   The default file name is the destination file name with the '.library'
extension modified into '.fd'.

   See also:
      AMIGALIBRARY
      COPYRIGHT
      FDLIB
      <filename>
      LIBID
      LIBPREFIX
      LIBREVISION
      LIBVERSION
      VERSION
      Become Registred

1.64   lk V1.04 - (Libprefix)

```
                        LIBPREFIX "<...>"
                          Default: "_"
```

   This instruction defines a common prefix for the name of your library
functions. This prefix will be added to the name present into the definition
file. For instance for QuickDOS library all library functions are all defined
with '_QD' lettres at the beginning. If the .fd files was defined as follow:

```
    .fd file              object file

  OpenFile              _QDOpenFile
  CloseFile             _QDCloseFile
  CloseAllFiles         _QDCloseAllFiles
  ...
```

lk will not be able to find the labels present into the object files. But with
the prefix "_QD" the symbols are now correct. Note that the prefix is not
applied to the system functions: '_LibInit', '_LibOpen', '_LibClose' and
'_LibExpunge'.

   This instruction is useful if you want, in C, to make some calls of your
function not going through the library base pointer. This is faster but not
conventional.

   The fact this instruction is used, set the flag AMIGALIBRARY. Please refer you
to that instruction to have all necessary informations.

```
  See also:
     AMIGALIBRARY
     COPYRIGHT
     LIBFD
     LIBID
     LIBREVISION
     LIBVERSION
     VERSION
     Become Registred
```

1.65   lk V1.04 - (Library)

```
              LIBRARY or LIB or -L [<filename> ...]
                         Default: OBJECT
```

   Defines files which will be used as libraries. This means that only necessary
hunks (or units) will be included in the destination file. The selection between
hunk or unit basis is done via BLOCKHUNK or BLOCKUNIT. The default is BLOCKHUNK
until you specify DICE or SLINK.

   The file name may include wild cards. Wild card should be avoid when you use
CC, DICE or SLINK instructions because the order of libraries objects is VERY
important (The first symbol found is the symbol which will be used.)

The AUTOLIBRARY command will be used to avoid the usage of the LIBRARY
keyword. If the DICE flag was used, all files with the extension '.lib' will be
taken as libraries.

   For each given name specified after the LIBRARY instruction, lk search the
file in:

      . the current directory
      . the LK:LIB/ directory when it exists
      . the LIB: directory when it exists

lk will search only the current directory if the file name include a path. A
path is included when a slash (/) or colon (:) character appear in the file
name.
   This feature does not actually work when you use AUTOLIBRARY flag.

   To ensure that your libraries are loaded with the normal memory requirement
you should use the instruction DEFAULT.

   When you are creating a library, this function enables you to generate an
independant library. If the library A needs the library B to be linked, because
it uses some hunks from B, joining library A with library B (With JOIN
instrucion in your CLI) may generates a big file, linking library A with file B
specified as a library should generate a smaller result.

      See also:
         AUTOLIBRARY
         BLOCKHUNK
         BLOCKUNIT
         DEFAULT
         FD
         <filename>
         FDLIB
         OBJECT
         Become Registred

1.66   lk V1.04 - (List)

                                    LIST <filename>
                                    Default: nothing

   Ask for a list of the hunks saved into the resulting file. That list will be
written into the specified file. You may type "*" (or "CONSOLE:" under V36+) to
have the list appearing into the current CLI window.

      See also:
         Become Registred

1.67   lk V1.04 - (Makerelative)

                          MAKERELATIVE or MR
                            Default: nothing

   Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

   Ask lk to transform, whenever possible, the RELOC32 in a relative addressing
mode. Of course only relocation from one hunk to it-self will be transformed by
this function, making some executable files smaller.

   This instruction is also available for executable programs only.

   To receive the WARNING which informs you about the modified hunks, type
'WARNING 4'.

                    BUG BUG BUG BUG BUG BUG BUG BUG BUG BUG BUG BUG

   This may create a lot of bugs... lk will check and suppose that the word
before the RELOC32 is an instruction (2 words before for MOVEM.W/L and BTST.B.)
   You should never use this instruction if you compiled your program in 68020
mode, or when a lot of data appear within the hunk of code. Usully the hunk of
code contains only code, right? And in any case, try your program extensivly
when you used that instruction to ensure it was correct.
   An instruction which uses a relocation for its second operand (Destination)
may create some errors.

   Note: if you only used RELOC32 for long branchements (Also JSR and JMP,) you
can use this function without any trouble.

   Supported instructions:

                                                (Size field: ?=B,W,L/S=W,L)

        ADD.?   (x.L),Dn
        ADDA.S   (x.L),An
        AND.?   (x.L),Dn
        BTST.B   #07,(x.L)
        BTST.B   Dn,(x.L)
        CHK.W   (x.L),Dn
        CMP.?   (x.L),Dn
        CMPA.S   (x.L),An
        DIVU.W   (x.L),Dn
        DIVS.W   (x.L),Dn
        JMP   (x.L)
        JSR   (x.L)
        LEA.L   (x.L),An
        MOVE.?   (x.L),Dn
        MOVE.?   (x.L),(An)
        MOVE.?   (x.L),(An)+
        MOVE.?   (x.L),-(An)
        MOVE.W   (x.L),CCR
        MOVE.W   (x.L),SR
        MOVEM.S   (x.L),reglist
        MULU.W   (x.L),Dn
        MULS.W   (x.L),Dn
        OR.?   (x.L),Dn
        PEA.L   (x.L)

```
     SUB.?      (x.L),Dn
     SUBA.S     (x.L),An
where the field (x.L) become (d16,PC).
```

```
  See also:
     ALV
     ANYRELATIVE
     NOWARNING
     WARNING
     Become Registred
```

1.68   lk V1.04 - (Map)

```
              MAP [H|h][,[R|r]][,[S|s]][,[U|u]]
                       Default: all flags
```

   This function enables you to declare what you want to write into the external
file (See XREF.)

   Each letter will enable (in upper case) or disable (in lower case) a specific
table. All tables are enable at start. Those letters may be written in any order
and can be separated by comas.

   The signification of each letter is:

```
    H     hunks table
    R     references table
    S     symbol table
    U     unit table
```

```
  See also:
     FANCY
     HEIGHT
     MARGIN
     PLAIN
     SWAPFH
     WIDTH
     XREF
     Become Registred
```

1.69   lk V1.04 - (Margin)

```
            MARGIN LEFT or RIGHT or TOP or BOTTOM <value>
                         INDENT <value>
                         Default: all 0
```

   Defines one of the four margins. lk never writes in margins. A margin is added
to the width or the height values (See.)

   Usuly margin values are defined with the printer preferences tool only. The

default lk margin are all null.

   The instruction INDENT <value> is an equivalent to:
                           MARGIN LEFT <value>

   See also:
      FANCY
      HEIGHT
      MAP
      PLAIN
      SWAPFH
      WIDTH
      XREF
      Become Registred

1.70   lk V1.04 - (Maxref)

Page 11-59

                      MAXREF <value>
                        Default: 5

   Defines the maximum number of hunk and unit pair references displayed when a
symbol does not exist. lk will, by default, display 5 references. The value can
be written in decimal, or in hexadecimal when started with a dollar ($) sign.
Set this to zero to ask all existing references.

   See also:
      ASKUNDEF
      WARUNDEF
      Become Registred

1.71   lk V1.04 - (Memory)

Page 12

                    Memory Management

   To make it quick, lk allocates large buffers of memory even a very small (1
byte!) amount of memory is required. Then each new little structure is allocated
inside the last large buffer taken.
   With small program, no modification is necessairy. But if you are working with
really large program or large libraries, you should change the large buffer
sizes. This is done with one of the following instruction:
                            HUNK
              (Number of hunks, default 50)
                        SYMBOL
         (Total of symbols names, default 16Kb)
                        RELOC
        (Number of reloc structure, default 1000)

   lk has a low level memory allocation routine which enables it to display a
requester when a memory allocation fails. This will enable you to ends some
programs, if you have some running in the background, and retry the lk

allocation.
   If you click on 'Cancel' button, lk will stop and return the main to user.

   If a memory error happend very often on your system, you may suppress the
QuickDOS.library or use the following instruction:
                                    QDSTOP

1.72   lk V1.04 - (Newobjects)

                              NEWOBJECTS
                          Default: old objects

   Most of the existing object files uses some external definitions which are no
proper. A current mismatch is the similarity given to relative and absolute
symbols. lk will check those mistakes and print a warning when the NEWOBJECTS
flag is given.

   See also:
      Become Registred

1.73   lk V1.04 - (Noalv)

                            NOALV or NOALVS
                            Default: NOALV

   This instruction suppress the effect of ALV. This is the default.

   See also:
      ALV
      MAKERELATIVE
      NOWARNING
      WARNING
      Become Registred

1.74   lk V1.04 - (Nodataoverlaid)

                            NODATAOVERLAID
                  Default: data in overlay are allowed

   Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

   This instruction will suppress any DATA or BSS hunk from overlaid units. This
is useful when you do not want to know how DATA and BSS hunks are managed. Note
that you still have to forget the usage of any kind of data within your hunks of
code (except for case tables or that kind of 'code' things.)
   This instruction is useless on executable files. Those files are supposed

functional and cannot be changed.

   In one way this is not the most intelligent usage of overlaid program. But
your C may not give you a choice.

   Example of code:

   This first example will need the NODATAOVERLAID:

```
     char *getdefaultstring()
     {
         return ("I am the default string");
     }
```

     using a compilation mode which forbid strings to be compiled within the code
     hunk.

   or

```
        section text,code

     getdefaultstring:
       move.l #defaultstring,d0
       rts

        section data,data

     defaultstring:
       dc.b "I am the default string",0
```

   This second example is not a valid overlay object:

```
     char *getdefaultstring()
     {
         return ("I am the default string");
     }
```

     using a compilation mode which forces strings to be compiled within the code
     hunk.

   or

```
        section text,code

     getdefaultstring:
       move.l #defaultstring,d0
       rts

     defaultstring:
       dc.b "I am the default string",0
```

   Note: in both cases the function is suppositivly called from an external hunk.
This kind of function may be called by the overlaid unit it-self.

   This third example shows the best way to do it:

```
    char *defaultstr = "I am the default string";

    char *getdefaultstring()
    {
        char *s;
        s = malloc(strlen(defaultstr));
        ... /* handle memory errors */
        strcpy(s, defaultstr);
        return (s)
    }
```

or

```
    char *defaultstr = "I am the default string";

    char *getdefaultstring(char *s) /* s has to be large enough */
    {
        strcpy(s, defaultstr);
        return (s)
    }
```

or

```
      section text,code

    ;input: a0 as the destination buffer
    getdefaultstring:
      lea     defaultstring,a1
      move.l  a1,d0
    .cpy
      move.b  (a1)+,(a0)+
      bne.b   .cpy
      rts

      section data,data

    defaultstring
      dc.b "I am the default string",0
```

```
  See also:
     ALV
     AUTOOVERLAY
     <filename>
     NODATAOVERLAID
     NOOVLDEBUG
     OVERLAY
     OVERLAYOBJECT
     SHORTRELOCOVERLAY
     Become Registred
```

1.75   lk V1.04 - (Nodebug)

Page 11-63

```
                              NODEBUG or ND
                  Default: keep the debugs which are known
```

Eliminates all debug tables. This will not destroy symbols. However only known
symbols are keept by default. If you want to keep all debugs, you will have to
use the KEEPDEBUG instruction.

  NODEBUG is not compatible to the same instruction of Slink. The instruction of
Slink will suppress debugs and symbols. lk will suppress only debugs. You should
use the STRIPDEBUG (STRIP) instruction which is the new and correct instruction
of Slink V6.xx (You can read the documentation of Slink about the instruction
'nodebug'.)

  See also:
      ADDSYM
      CLEARADVISORY
      CREATEDEBUG
      CREATESYMBOL
      KEEPDEBUG
      HUNKADVISORY
      LEFTADVISORY
      NOBSSDEBUG
      NOCODEDEBUG
      NODATADEBUG
      NOLIBDEBUG
      NOLOCALSYMBOL
      NOOVLDEBUG
      NOSYMBOL
      SETADVISORY
      STRIPDEBUG
      Become Registred

1.76   lk V1.04 - (Noemptyhunk)

                                                            Page 11-64
                          NOEMPTYHUNK or NEH
                        Default: keep empty hunks

  This instruction enables the systematical suppression of empty hunks.

  Those are very needed when you link objects from DICE. The automatic init/exit
code will not be properly linked when NOEMPTYHUNK is used.
  This of course does not affect executable files.

  See also:
      NODEBUG
      NOMULTIPLE
      NOSYMBOL
      NOWARNING
      Become Registred

1.77   lk V1.04 - (Noexe)

                                NOEXE
                 Default: put a '.exe' extension

   Forbid the automatic '.exe' extension to appear when the destination file name
exists within the list of source file. This might be dangerous when present into
your preferences file ('lk.prefs'.)

   See also:
      TO
      FOR
      Become Registred

1.78   lk V1.04 - (Noicon)

                          NOICON or NOICONS
                         Default: NOICON
                 (DEFICON if lk is started from the Workbench)

   Forbid the création of an icon.
   By default, when you start lk from your CLI, no icon is created. But when you
start lk from the Workbench, the default icon will be automatically selected.
You can also use this command to forbid the default icon of appearing.

   See also:
      DEFICON
      ICON
      UNSNAPSHOT
      Become Registred

1.79   lk V1.04 - (Nolocalsymbol)

                        NOLOCALSYMBOL or NLS
                       Default: keep all symbols

   Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

   Go through symbol tables and deletes any symbol which starts with a period (.)
or ends with a dollar ($). This is useful because some compilers keep all
symbols, even those are internal to the given function.

   This may be used to have smaller executables and quicker debugging.

   See also:
      ADDSYM
      LEFTADVISORY
      CREATEDEBUG
      CREATESYMBOL

```
      KEEPDEBUG
      HUNKADVISORY
      LEFTADVISORY
      NOBSSDEBUG
      NOCODEDEBUG
      NODATADEBUG
      NODEBUG
      NOLIBDEBUG
      NOOVLDEBUG
      NOSYMBOL
      SETADVISORY
      STRIPDEBUG
      Become Registred
```

1.80   lk V1.04 - (Nomultiple)

```
                        NOMULTIPLE or NM
          Default: absolute symbol can be defined several times
```

  No multiple absolute symbol definition allowed.

  Even the same symbol is defined twice with exactly the same value, an error
will be generated (See error #028.)

```
   See also:
      Error #028
      NOWARNING
      Become Registred
```

1.81   lk V1.04 - (Nooverflow32)

```
                          NOOVERFLOW32
                Default: error on overflow 32 bits
```

  lk, like peviously said, checks a lot of things. Also it does an overflow
checking on the reloc32 and ref32. If this is a problem, use the NOOVERFLOW32
flag to forget that error.

```
   See also:
      NOWARNING
      Become Registred
```

1.82   lk V1.04 - (Nopath)

```
                          NOPATH
                Default: See any of the ???PATH instruction
```

Suppress all default paths. Only the current directory will be checked while
none of the PATH instructions are used.

    See also:
        FDPATH
        ICONSPATH
        LIBPATH
        OBJPATH
        WITHPATH
        Become Registred

1.83   lk V1.04 - (Norm)

                                NORM
                              NORMBSS
                              NORMCODE
                              NORMDATA
                              NORMHUNK
                              NORMUNIT
                          Default: NORM

   This instruction forbids any link between hunks, even their names are equal.
This will also generate a big resulting file. Those instructions are useless for
libraries.
   Note that small data code (HUNK_DATA access with A4 register) will anyway be
linked together or there would be no use of this function.
   The instruction NORM is an equivalent to all other NORM??? called at the same
time, including NORMHUNK.

   Note: the instruction -FRAG ask for all type of hunk to be fragmented. This is
to ensure a compatibility with DLINK.

    See also:
        FRAGBSS
        FRAGCODE
        FRAGDATA
        SMALL
        SMALLBSS
        SMALLCODE
        SMALLDATA
        SMALLHUNK
        SMALLUNIT
        SINGLE
        SINGLEBSS
        SINGLECODE
        SINGLEDATA
        SINGLEHUNK
        SINGLEUNIT
        Become Registred

1.84   lk V1.04 - (Nospecialdebug)

                           NOBSSDEBUG or NBD
                          NOCODEDEBUG or NCD
                          NODATADEBUG or NDD
                           NOLIBDEBUG or NLD
                           NOOVLDEBUG or NOD
                    Default: keep known debugs

   Eliminates all debug tables from the specific type. This will not destroy
symbols.

   NOCODEDEBUG   forget all debugs from any CODE hunk.
   NODATADEBUG   forget all debugs from any DATA hunk.
   NOBSSDEBUG   forget all debugs from any BSS hunk.
   NOLIBDEBUG   forget all debugs from libraries. (This is unused when you link
               executable.)
   NOOVLDEBUG   forget all debugs from overlaid hunks.

   See also:
      ADDSYM
      CLEARADVISORY
      CREATEDEBUG
      CREATESYMBOL
      KEEPDEBUG
      HUNKADVISORY
      LEFTADVISORY
      NODEBUG
      NOLOCALSYMBOL
      NOSYMBOL
      SETADVISORY
      STRIPDEBUG (No debug and symbol)
      Become Registred

1.85 lk V1.04 - (Nosymbol)

                          NOSYMBOL or NS
                    Default: keep all symbols

   Eliminates all symbol tables. This will not destroy the debug tables.

   This flag is automatically set when you ask for NODEBUG and SLINK has been
used. This is to enable a real Slink compatibility.

   For executable file, there is no way to restore any symbol (Except if you have
the sources!)

   See also:
      ADDSYM
      CLEARADVISORY
      CREATEDEBUG
      CREATESYMBOL
      KEEPDEBUG
      HUNKADVISORY

```
        LEFTADVISORY
        NOBSSDEBUG
        NOCODEDEBUG
        NODATADEBUG
        NODEBUG
        NOLIBDEBUG
        NOLOCALSYMBOL
        NOOVLDEBUG
        SETADVISORY
        STRIPDEBUG (No debug and symbol)
        Become Registred
```

1.86   lk V1.04 - (Nowarning)

```
                          NOWARNING or NW
                          Default: QUIET
```

   Suppresses absolutely all warnings regardless of their level. This might be
smarter to use the WARNINGLEVEL instruction and to keep the lower level you can
(NOWARNING instruction also places the level at 100.)

   See also:
      QUIET
      WARNING
      WARNINGLEVEL
      Become Registred

1.87   lk V1.04 - (Object)

```
                    OBJECT or OBJ [<filename> ...]
                          Default: OBJECT
```

   Following object files will be linked into the resulting file. This is the
opposite of LIBRARY keyword which loads files which are not automatically linked
into the resulting file. OVERLAY keyword changes the stream as well.
   Note: the keywords FROM or ROOT automatically restore this mode.

   Usully this keyword is unused. The following format should be used:

```
   FROM  <fichier racine>
         [<fichier objet> ...]
   LIB   <fichier librairie> [...]
   OVERLAYOBJECT <fichier objet>
   OVERLAY
         <fichier objet> [...]
   END
   FD    <fichier definition> [...]
   FDLIB <fichier definition> [...]
   WITH  <fichier WITH> [...]
```

See also:
    FROM/ROOT
    LIBRARY
    Become Registred

1.88 lk V1.04 - (Odd)

                                    ODD
                    Default: accept only even relocs

  Odd relocation values are usuly forbidden. Those would generates addresses
errors on 68000 based machines. But now that more and more are programming in
68020, it could happen that some pointer where oddly defined (in data hunks
usuly.) Then you may specify ODD and enable odd relocation values.

  Note: this is not supported on CPU below the 68020.

  See also:
    Become Registred

1.89 lk V1.04 - (Offset)

                            OFFSET <value>
                          Default: automatic

  Defines the offset at which the relative informations will be linked.

  This will force the relative to be at that position which means the 8 bits
relative pointers may certainly not be linkable. This should only be used if you
do not use 8 bit relative.

  The OFFSET value should be defined between -32768 and +32768 and be even. No
check is provided to let user make its choice; except the warning which is sent
when the OFFSET is odd.

  For DICE an offset of -$7FFE must be given or the link will not work properly
(When you link executable the offset is unused then it is safe to let it
undefined.)

  See also:
    DICE
    Become Registred

1.90 lk V1.04 - (Onedata)

                            ONEDATA or OD
                  Default: keep data and bss hunks distinct

   Transform BSS hunks into DATA hunks and link them at the end. This is useful
to build pure programs, while this way only one section has to be copied.
   To work properly ONEDATA has to be used with SMALLDATA and XDATA. Then the
result will be a single hunk of data per type of memory. Without SMALLDATA, only
hunks of the same name will be linked together and without XDATA, the BSS hunk
will be expanded as a large area of zeroes.
   Note that BSS and DATA hunks may have the same name when this instruction is
used.

   This instruction is not valid when a library is created.

   See also:
      SMALL
      SMALLBSS
      SMALLCODE
      SMALLDATA
      XCODE
      XDATA
      Become Registred

1.91   lk V1.04 - (Opts)

                                                                      Page 13
                               lkopts V1.01
                         The linker options interface.

   This program is useful for the one which work with its Workbench a lot. All
tasks can be done into the Workbench.

   This utility has been created to facilitate the option selection. There is a
so large number of options and so many ways to make a mistake, that this program
not only enables you to make your choices, but it also prevent you to use
redondant keywords or the right values at the right place.

   After lkopts loads a first window appear into the top-left corner. It contains
the following buttons:

      About
        display the current version
      Help
        try to load this help file
      Quit
        quit lkopts program
       Set Default
        enables you to select a default setting
       Destination
        definition of the destination file name
       External Map
        symbol tables preferences for output
       Load
        it loads a new WITH file
       Object
        defines the list of objects, WITHs, libraries...
       Defines

```
     enables the definition of external labels
   Save
    it saves the current WITH file
   Flags
    defines specific flag setting
   Address
    gives the program location
   Verify
    verify your selection
  Info
    gives information about each instruction
```

At any time, when a file name as to be declared you may drop an icon into the
string gadget. The name will automatically appear with the entire path.

This utility has been created with 'Intuition Interface' tool which was
written by Alexis WILKE (c) 1993-1994.

1.92   lk V1.04 - (Opts_address)

                                                                       Page 13-1
                                 lkopts V1.01
                                   Address

This window gives you the possibility to type the address where each type of
hunk has to be relocated. This is useful to link programs at a specific address
(For games, ROMs or Unix like systems.)

Each type of hunk needs to have an address, except:

     if ONEDATA is used no BSS hunk will exist, then no BSS address is needed.
     if SMALLHUNK or SINGLEHUNK is used no data and BSS hunk will exist, then
    data and BSS address is needed.

See Also:
   lkopts helps
   ADDRESS
   ONEDATA
   SMALLHUNK
   SINGLEHUNK

1.93   lk V1.04 - (Opts_default)

                                                                       Page 13-2
                                 lkopts V1.01
                                   Default

This window enables you to ask 'lkopts' to reset to a specific default
setting.
A default setting is also used to make lk compatible to a specific type of
link.

Available defaults:

   lk
        Resets 'lkopts' to lk as used directly into the CLI or from the
        Workbench without any WITH file.

   No Dbl. Symb.
        Resets 'lkopts' to link simple assembler object files. This will
        suppose that no symbol is duplicated anywhere. Libraries used with C
        language like will usuly have several functions with exactly the same
        name.

   C
        Resets 'lkopts' to link C like code and libraries. This will also
        enable multiple symbols definition into the libraries.

   DICE
        Resets 'lkopts' to link like dlink of DICE. This may not be fully
        compatible, while I do not have any documentation about that very
        specific linker. Note that you should never use DICE instruction if
        you do not compile DICE object files.

   Blink
        Resets 'lkopts' to be compatible to Slink. This will not be fully
        compatible because lk does not crash when too much symbols are
        defined...

   Slink
        Resets 'lkopts' to be compatible to Slink. The greatest difference
        with Blink is in the fact the Slink support more hunk types and have
        more internal variables (like '___ctors' and '___dtors'.)

See also:
   lkopts helps
   CC
   DICE
   SLINK

1.94   lk V1.04 - (Opts_defines)

                                                                  Page 13-3
                            lkopts V1.01
                              Defines

   This window displays the list of defines. Defines are mostly used for
compatibility with other linkers. It can also be used to define some symbols
externally. See also the DEFINE instruction for more information.

   The button NEW create a new symbol. This one appear in bright color and will
become black when valid. No space or sign should appear. Type the external
symbol name in the left edit box and the value or the internal symbol name in
the right edit box.

   The DELETE button will deletes the currently selected definition. There is no

way to restore a delete definition.

   The CLEAR button will clear all definitions.

   The HIDE button will hide the window. A click on DEFINE will re-open the
window.

   The use of the 'Default' window may add some definition.

   See alo
      lkopts helps
      DEFINE
      VERSION

1.95   lk V1.04 - (Opts_destination)

                                                                          Page 13-4
                                     lkopts V1.01
                                     Destination

   This window will be used to define the destination file name (Into the box
named 'File Name') and type which is 'Executable' or 'Library'. Library means
you want to create an object file (this is also some kind of pre-link.)
   When a library is created no external map is available. lk does not really
support this actually.
   Note: you may drop an icon into the file name gadget box; its name will
automatically appear.

   You may type the name of an icon or drop an icon over the big rectangle from
your workbench. No check is done to be sure the icon is of type TOOL, you have
to take care of that. Select one of the check gadget 'Normal' or 'Selected' to
see the two states of the icon.
   The default of lk is NOICON, but if you have a preference file you may select
the check button named 'No Icon' to forget about the icon from the preference
file.

   If you define the version and copyright lk will automatically create the
version string. To be valid, the copyright must be the program name only. The
version must be two values separated by a period. lk will take the date at the
time of the link to define the date of the version string. For instance:
      Version: 1.03
      Copyright: lk
   Note: you may drop an icon into the copyright gadget box to have its version
appearing inside. The icon must refer to a file with a valid version string.

   See also:
      lkopts helps
      filer of lkopts©
      COPYRIGHT
      CREATELIBRARY
      DEFICON
      FOR
      ICON
      NOICON
      TO

VERSION

1.96   lk V1.04 - (Opts_filer)

```
                        lkopts V1.01
                         Load/Save
```

  When you want to load or save a file, you will have a normal requester file
which appear.

  The file name edit box will be used to type the name of the WITH file.
  The Drawer file name edit box will be used to type the path to reach the WITH
file.
  The parent button can be used to reach the parent (previous level) directory.
  The list of devices, assigns and volumes can be used to list their contains.
Click on one of them for this purpose.
  The list of file and drawers will be used to reach another drawer (next level)
or select the WITH file. A double click on the WITH file will correspond to a
click on the button 'Load' or 'Save'.
  The button 'Cancel' will stop the selection and cancel the eventual effect. No
file will be loaded or saved.
  The file 'S:lk.prefs' can automatically be loaded when it does exist because a
button named 'Prefs' will appear. When you are saving a WITH file, the button
'Prefs' will always appear.
  When you save a WITH file and no icon exists, the default icon is
automatically added. lkopts© tries first to load 'def_with' file from the
current directory, 'LK:ICONS/', 'SC:ICONS/' and 'ENV:SYS/'. If it is not
founded, the internal icon is used.

  The same window will also appear when you want to select object files, the
destination file and the icon file.

  See also:
    lkopts helps

1.97   lk V1.04 - (Opts_flags)

```
                        lkopts V1.01
                          Flags
```

  This window includes all flags. You will have to select the flag to modify in
the list on the left, then make the selection on the right side of the window.
The title gives you the name of the currently modified flag.

  Depending upon your selection, some comments may appear. Comments give a
complete information for beginners. Those will be increased in the future. If
more information is needed, the 'Information' window can be used for this
purpose. If you are not sure about a selection, you may keep the 'Verify' window
open, it will automatically be refreshed with each new choice and it directly
gives you a list of your 'mistakes'.

The check boxes present on the right side of the window will be enable when needed. It's not possible to unselect all of them at a time. You will quickly see that some choices are not possible, you cannot ask to link in mode DICE and SLINK at the same time!

The edit box is needed for many flags which need a more complex choice. The startup label has to be defined by you, like the offset value if you want to specify it.

Be careful! If you want to select a default state, you MUST select that state first and make your modifications after.

See also:
    lkopts helps
    All lk© keywords

1.98   lk V1.04 - (Opts_object)

Page 13-7

lkopts V1.01
Object

A click on this button will open the object window. This window contains three parts:
    1. the object list (left)
    2. the object definition (right)
    3. the actions (buttons at the bottom)

To enter new object into the list you can:
    1. click on 'New Object' to create an object by your-self
    2. take an icon from Workbench and drop it into the list (*). If you drop a drawer the form '#?' (all files of that drawer) will be appened to its name. It's up to you to modify the pattern into '#?.o' or '#?.lib' or whatever else.
    3. click on 'Get File' and search the file into the requester

(*) a dropped icon is usuly directly copied into the list. If you select the 'Select dropped files' check box, you will have a pre-selection window opened. That secondary window enables you to drop a complete directory and select only a few files of the entire drop.

To be able to modify an object information it has to be selected. For this you have to click on it with the mouse, it will appear with a different color and the file name will appear into the edit box.
You may change the file type by selecting a type into the first selection button. The available types are:

    Object
        Normal AmigaDOS object file, present in result.

    Root
        Object to be linked at start of executable file.
        Note that the overlay handler will always be linked in front of it.

Library
  Library object, linked only when necessary.

Executable
  Only one executable can be defined, it will apply the selection onto the
  selected file.

Overlay
  Object files which will be saved in the overlay units; each file will be
  into a distinct unit.
  (please refer to overlay documentation for more informations)

Overlay Handler
  The object file which will be used as the overlay handler. The default is
  'LK:LIB/overlay.o' or the internal handler when that file file does not
  exists.

FD
  Source files must be library definition files. It will be used to create
  include or library (object) files with the offsets of each function.

FD Library
  Source files must be library definition files. It will be used to create
  header or library (object) files with a C call for each function.

With
  WITH files which will be executed after this one. No check is done to
  ensure that a WITH file does not call it-self. If you do it, lk will loop
  forever.

You can change the memory requierement for Object, Root, Library, Executable,
    Overlay and Overlay Object files. This is done with the menu selection of
    the middle. The available memory requirements are: Default (take the
    definition from the file), Chip, Fast and Attributes. The last one will
    enable you to type the requirement flags as you like. If you do not know
    your memory flags, you must use 'Select Attributes'. A new window will be
    opened, make your selection and click the OK button to accept. The
    requierement flags appear into the edit box.

You can change the new debug mode for each file; like the memory
    requierements. There are three possible choices:

  1. 'Use Hunk Advisory'
      Takes the definition present in the source files.

  2. 'Set Advisory Bit'
      Forces the advisory bit to be set into all hunks of the source files.
    Those hunks will not be automatically loaded by the function 'LoadSeg()'
    of the AmigaDOS.

  3. 'Clear Advisory Bit'
      Clears the advisary bit from any hunk of the source files. All hunks
    will then be loaded by the AmigaDOS function 'LoadSeg()'.

See also:
  lkopts helps
  filer of lkopts©

```
CHIP
CLEARADVISORY
DEFAULT
FAST
FD
FDLIB
FROM/ROOT
HUNKADVISORY
LEFTADVISORY
OVERLAY/END
OVERLAYOBJECT
SETADVISORY
WITH
```

1.99   lk V1.04 - (Opts_verify)

Page 13-8

```
                      lkopts V1.01
                        Verify
```

  This window shows you the 'invalid' selections. Because lk is a complex
linker, it's easy to make a mistake. This window has been created to support you
into your selections.

  You have three levels of invalid states:

    1. Remarks. QUIET instruction which is not compatible to Slink instruction
         is shown as a remark.
    2. Incompatible selection. Definition of a startup label with Slink mode.
    3. Wrong or dangerous selection. Ask for fragmented code hunks in DICE mode
         will generate invalid init/exit routines.

  To recover any of those problems, you may select the one you want and then
         click on the 'Correct' button. All remarks may not be recoverable.

  See also:
    lkopts helps

1.100   lk V1.04 - (Opts_xref)

Page 13-9

```
                      lkopts V1.01
                      External Map
```

  This window enables a quick and simple selection for the external map. You
should keep all default values, except if you really need a specific size.
  While a file name is not specified, no external map is created.

  Into the 'Print' group you can select which kind of reference you need:

    Symbol
      A list of all symbols will be written to the map file. Each symbol is

displayed with each of its references. The total number of reference is
given as well as the position and size into the resulting executable.

  Hunks
    Display the list of hunks. That list gives the position and size of each
    hunk into the destination file. This way you may find out, when you are
    debugging, which source file the code is from (Note: the use of
    CREATEDEBUG might be another way.)

  Unit
    Display the list of units. This is similar to the hunk list, but usuly
    quicker because a unit may contains several hunks.

  References
    This gives the list of the referenced symbols.

  Into the 'File/Hunk' group you can select the title and the foot of your page.
There is nothing very important.

  Into the 'Console' group you will accept (Fancy) or forbid (Plain) the usage
of special modes. Into a console or with your printer, it will be nicer to use
'Fancy', because you will see the bold and colored characters. If you just want
to edit the resulting map file, you should probably use 'Plain', especially if
your editor does not interpret the CSI codes

  See also:
    lkopts helps
    FANCY
    FWIDTH
    HEIGHT
    HWIDTH
    MAP
    MARGIN
    PLAIN
    PWIDTH
    SWAPFH
    SWIDTH
    WIDTH
    XREF

1.101   lk V1.04 - (Order)

<div align="right">Page 11-79</div>

<div align="center">ORDER
Default: given order</div>

  The ORDER keyword has been added to enable specific preferences files. If you
want to give default libraries in your preference file, those will be linked
before the object files. This will generate problems especially if you are
linking objects and libraries from SAS.

  When you give file names to lk, all files are linked into one list. The ORDER
instruction will force object files to be at start, overlaid files to be in the
middle and the libraries at the end.

The use of this instruction to link DICE codes will generate invalid programs.

See also:
    LIBRARY
    OBJECT
    OVERLAY
    Become Registred

1.102   lk V1.04 - (Other)

Page 14
                            About my other products

   If you want to know about my products, do not hesitate to contact me (Please
send me stamped letter with your address.) If you want to receive my programs on
a diskette, send a disk with your mail. The disk does not need to be formatted.
If only text files is what you want, tell me the computer Amiga or PC.
                          Click here to have my address!

   The next products of mine, soon available, are:

    An intuition editor
       . supports 12 different gadgets (buttons, edit boxes, complete lists,
         check box, menus, etc...)
       . creates object files (Nothing is to be compiled with a specific compiler
         or assembler.)
       . possesses a forth generation language object oriented (4GL)
       . the shareware version is coming soon on your prefered BBS

    A map editor
       . for any game development, you always need a map editor to create the
         background, define position of animations, etc...
       . enables any programmer to have an IFF file containing a map created by
         any graphist!
       . the map size is limited only by memory (Also sizes are limited to 16383,
         but 16384 * 16384 * 2 is the memory requirement to create the larger
         map, enough memory ?)
       . the brushes are not limited to one block, but (like maps) are limited to
         a size of 16384x16384...
       . the blocks are 'created' from any IFF picture, having up to 16383 blocks
         (block zero is the background color) which can be swapped horizontally
         and vertically (Also this is 65536 blocks...)
       . this product will be commercial only.
       . if you want more informations, please contact me, I'll provide a
         complete documentation about all features present in this editor.

     A help displayer
       . enables anyone to write help files with its prefered editor
       . supports any number of languages than required (which can be selected
         dynamically and through locale preferences.)
       . possesses a search command over all files in specified language.
       . comes with a help stripper to transform help files into normal text
         files and a help to amiga guide to transform help files into one guide
         file.

1.103   lk V1.04 - (Overlay)

```
                   OVERLAY <filename> ... END
                   Default: OBJECT (No overlay)
```

  Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

  All object files defined between OVERLAY and END keywords will be linked into
the overlay part of the file. Each unit of each object file will represent an
overlay unit into the destination file. Memory attributes keywords are valid
within that définition. You may also write:

```
  OVERLAY
    FAST prog1.o
    CHIP pic1.o
    PUBLIC port_n_msg.o
  END
```

  Note: The sharp sign (#) is a synonym of END keyword to keep the compatibility
with Blink and Slink. The star sign (*) is simply taken as a space character (If
you use a star as part of a DOS pattern, you will have to enclose your file name
within cotes.)

```
  See also:
      AUTOOVERLAY
      <filename>
      NODATAOVERLAID
      NOOVLDEBUG
      OVERLAYOBJECT
      SHORTRELOCOVERLAY
      The Overlays
      Become Registred
```

1.104   lk V1.04 - (Overlayobject)

```
            OVERLAYOBJECT or OO or OVRLMRG <filename>
                        Default: OBJECT
```

  Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

  lk overlay handler program is given with this product to enable programmers to
make mofications and eventually transform it completly. If you create an
external handler, you will have to specify it with the OVERLAYOBJECT
instruction.
  You may put that object file into one of your libraries. In this case you must
give a specific name to the hunk section of CODE which will be the program
startup:

```
                  OVERLAYENTRY or NTRYHUNK or ENTRYHUNK
```

  lk also offer another possibility. You can save your object file as

'LK:LIB/overlay.o'. lk will automatically try to load that file and it will use
it rather than the internal one. That file will never be used when the
OVERLAYOBJECT instruction was used.
   The usage of OVERLAYOBJECT instruction will enable the effect of memory
instructions to be effective. The following command line will link the overlay
handler to be loaded in fast memory:
        FAST OVERLAYOBJECT <filename>

   That object has to contain:
      . a JSR/JMP to the '_ovl_root' which is the startup point of the main
        program.
      . a function called '_ovl_call' to ensure that the object in overlays can be
        called (It was the function named '_ovlyMrg' when you used Blink and
        Slink.)
      . a starting overlay table with the magic long word $0000ABCD followed by
        four null long words.
      . a hunk of type CODE at start (It may contain some other code or data and
        bss hunks.)

   Note:
      If you define a label or a function named '_ovl_root', lk will use it as the
   startup program instead of the STARTUP label or the first hunk of code.

   See also:
      AUTOOVERLAY
      <filename>
      NODATAOVERLAID
      NOOVLDEBUG
      OVERLAY
      SHORTRELOCOVERLAY
      The Overlays
      Become Registred

1.105   lk V1.04 - (Ovl)

Page 15
                              The overlays

   Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

   The purpose of overlays is to have really big programs not entirely in memory
at a time. The overlay manager will take care of memory allocations and
dispositions.
   To make the main code the smaller than posible, all library hunks which are
used only by a specific overlaid unit will be linked into that unit. This will
save execution time and space.

   To correctly show you the difference between an overlay hunk and an overlay
unit there is a diagram:

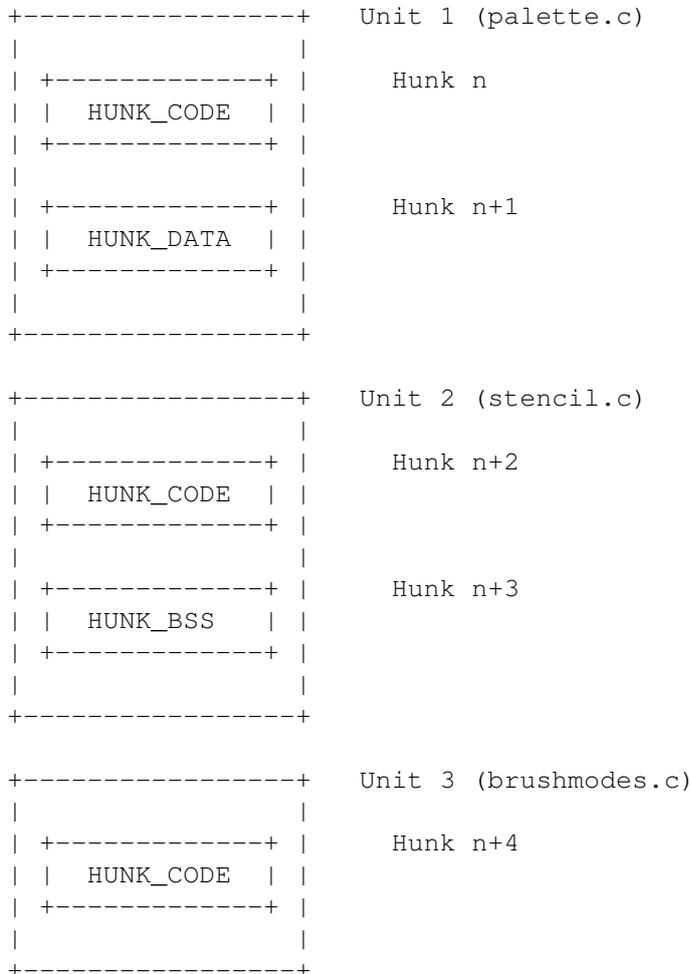   let's say that we have a WITH file which contains:

        OVERLAY
          palette.o

```
        stencil.o
        brushmodes.o
      END
```

   knowing that each object file contains a hunk of code and a hunk of DATA or
BSS, the following diagram show you how each file is saved inside the final
overlaid executable:

```
  +----------------+   Unit 1 (palette.c)
  |                |
  | +------------+ |      Hunk n
  | |  HUNK_CODE  | |
  | +------------+ |
  |                |
  | +------------+ |      Hunk n+1
  | |  HUNK_DATA  | |
  | +------------+ |
  |                |
  +----------------+


  +----------------+   Unit 2 (stencil.c)
  |                |
  | +------------+ |      Hunk n+2
  | |  HUNK_CODE  | |
  | +------------+ |
  |                |
  | +------------+ |      Hunk n+3
  | |  HUNK_BSS   | |
  | +------------+ |
  |                |
  +----------------+


  +----------------+   Unit 3 (brushmodes.c)
  |                |
  | +------------+ |      Hunk n+4
  | |  HUNK_CODE  | |
  | +------------+ |
  |                |
  +----------------+
```

   (*) where n is the number of hunk is the root part.

   Of course, each hunk of CODE, DATA and BSS may have some attached hunks of
DEBUG, SYMBOL and RELOCation. If you need to refer a hunk in overlay, you just
need its unit number.

   The overlay section of lk does not follow the way it was designed by
Commodore. No tree has to be respected, every unit can call every other unit, lk
will take care of that. Calls to an external module can be done with a JSR/JMP
and BSR/BRA instructions as well. lk will automatically generates the overlay
tables needed to permit those calls (If you used BSR/BRA assembly instructions,
you will probably need the use of the ALV command.)
   Note that calls from an overlaid hunk to any of the root hunks is just part of
the relocation informations. Only calls from the root to the overlaid units or
calls from overlaid units to other overlaid units generate overlay tables.
   An overlay call is very long and you MUST avoid them in loops. It actually
includes around 20 assembly instructions. To ensure faster calls, you should use

the SMALLCODE command. This will avoid the research of the CODE hunk to call.
  An overlay unit may consist of DATA only. In that case, an access will be
granted only after a call to the overlay handler loader.
  After a call returns from an external function, the called funtion is no
longer residing in memory. No direct access can be realised (Even if that
function is keept in memory by the default overlay handler.) This means no
pointer on static data can be returned from an overlaid hunk. Note that the
special NODATAOVERLAID command can be used to enable this kind of programming.
Have a look to the instruction NODATAOVERLAID to see some examples.
  You may force all overlaid hunks to be loaded at a time with a call to the
'ovl_load' function.

  If you are linking C objects which access their data in an unknown way, you
may take the risk to have problems with those data because they are no more
available after the last return from an overlaid function. To prevent that
problem the NODATAOVERLAID instruction will move all DATA and BSS hunks into the
main program.
  Note: this is required if you use a small memory model (?!?) But lk will
automatically handle the data displacement and NODATAOVERLAID use not necessary.

  When loaded with the default overlay handler, hunks are NOT linked like they
are with LoadSeg() of AmigaDOS. The segments have height bytes at the beginning
with the size and the 68000 pointer to the next segment.
  Each unit is a new linked list (Units are never linked together.)

  The available functions from the default overlay handler are listed and
explained below. Those are available into the 'overlay.h' header.

    hunk_pointer _ovl_load(unit_number)

    void *ovl_load(long);

    The parameter is the number of the unit to be loaded. This call will
  increase the access counter by one. This function might be called to ensure
  the physical presence of a unit in memory, especially if that unit is only
  composed of data.
    The returned pointer is the startup to the first hunk of that overlay
  unit. This is a normal 68000 pointer, not a BCPL.
    If you call this function with -1, all units will be loaded in memory and
  then available at any time. In this case this function returns DOSTRUE when
  no error occured, otherwise DOSFALSE. To know a unit pointer you will have
  to call 'ovl_load' again with a specific unit number. A call   to
  'ovl_release' enable the system to suppress the lock.

    success _ovl_release(unit_number)

    long ovl_release(long);

    Release the specified unit from its lock. This function reduce the access
  counter by one, but does not free the corresponding units memory, even the
  counter drop to zero. The free is realised with the 'ovl_dispatch' function.
    If you need a data hunk temporary in memory, you may call the 'ovl_load'
  function and then the 'ovl_release' and 'ovl_dispatch' to free it when you
  do not need it any longer.

    success _ovl_dispatch(unit_number)

```
    long ovl_dispatch(long);
```

   Try to free the specified unit from memory. It only works when the access
counter is null. You might pass -1 as parameter to discard all units which
can actually be freed.

```
    success _ovl_freehook(free_func)

    long ovl_freehook(void (*)(long));
```

   When the overlay handler needs to load a new hunk, it may fails with the
memory allocation. Then it will free all memory buffers reserved for other
unused units. If the allocation fails again a call to a user function is
processed.
   If that function is able to free some memory, it will return and the
overlay handler will try again to load the needed unit. If it does not work
a second call will be processed to the user function... and so on. Each new
call to the user function will be done with an incremented parameter
starting from null. On zero the user should free unused data, on one it
should free every thing it can and on 2 it should quit on an 'out of memory'
error (except if the unit is not necessary...)
   The overlay handler will call that function with -1 if it has a problem
loading the unit (AmigaDOS error.)
   If no user function is defined and the new hunk can not be loaded, the
function 'ovl_call' will return with all registers unchanged (including CCR)
giving the impression of an empty call.

   Note: the parameter is given in D0 and on the stack.

   The overlay handler program is given with this product to enable programmers
to make mofications and eventually transform it completly. See also the
instruction OVERLAYOBJECT for more informations.

   See also:
      ALV
      AUTOOVERLAY
      <filename>
      NODATAOVERLAID
      NOOVLDEBUG
      OVERLAY
      OVERLAYOBJECT
      SHORTRELOCOVERLAY
      Become Registred

1.106   lk V1.04 - (Paths)

                         FDPATH "<...>" ...
                       ICONSPATH "<...>" ...
                        LIBPATH "<...>" ...
                        OBJPATH "<...>" ...
                       WITHPATH "<...>" ...
                        Default: see below

Defines the list of paths to search the given type of file. Any number of
directories and assigns can appear into your paths. Devices which does not
exists into your system are not checked. For instance, if 'LIB:' does not exist,
the path for library search will not create the usual DOS requester.) Multiple
paths may appear into one string, in this case you must separate each path with
one of the following sign (only the first one is conventional):
    . '|' a pipe (Logical OR in C)
    . '+' a plus
    . ' ' a space character
    . ',' a coma
    . '\\' a backslash
If you have some spaces into the name of your directory, you will have to type
its name between cotes:
    "'My Path As Spaces'|NormalPath"
become:
    1. My Path As Spaces
    2. NormalPath
(Note: the opposite cotes was used within the string definition.)

    To indicate the current directory you can use a single dot (.), like:
    ".|LK:LIB/|LIB:"
for the default library paths present in lk. If you need to use only the current
directory an empty string is valid, for instance:
    WITHPATH ""

    Paths are checked in the given order. If you prefer to check 'LIB:' before
'LK:LIB/', you could change the default path with:
    LIBPATH ".|LIB:|LK:LIB/"

    The fact you use one of those instructions, destroy the previous list of paths
of the given type. If you want to suppress all default paths, use the
instruction NOPATH.

    At start, if you have the V36 or more of AmigaDOS, lk checks global variables
(Present in 'ENV:' assign.) All paths can be defined that way and also available
to all of you links. The variables have the same name plus the prefix 'LK/'. To
redefine the path for the description files, you may type into your CLI:
    setenv LK/FDPATH ".|INCLUDE:FD.FILES/"
This definition is temporary until the next RESET. If you want it to become
permanent, you must copy each variable into the 'ENVARC:' assign.
    Note: the usage of one of the instructions will erase the external variable
paths.

    The available paths and their usage are listed below. If you use a specific
preference file ('slink.prefs' for instance) the starting default will differ.
The default given here is the defaults which are found into the original file
(except for those of lk which are internal.) The 'c.prefs' file uses the same
defaults than lk.

        Command: FDPATH
           Type: Description Files
     lk Default: Current Directory
                 INCLUDE:FD.FILES/
     Slink Def.: Current Directory
     Blink Def.: Current Directory
      Dice Def.: Current Directory
    Instruction: FD

```
            FDLIB
            LIBFD

   Command:  ICONSPATH
      Type: Icon Files
 lk Default: Current Directory
            LK:ICONS/
            SC:ICONS/
            ENV:SYS/
 Slink Def.: Current Directory
            SC:ICONS/
            ENV:SYS/
 Blink Def.: Current Directory
  Dice Def.: Current Directory
            ENV:SYS/
Instruction: DEFICON
            ICON

   Command: LIBPATH
      Type: User libraries
            Default objects (*)
 lk Default: Current Directory
            LK:LIB/
            LIB:
 Slink Def.: Current Directory
            LIB:
 Blink Def.: Current Directory
  Dice Def.: Current Directory
Instruction: LIBRARY
            OVERLAYOBJECT

   Command: OBJPATH
      Type: User objects
 lk Default: Current Directory
            LK:OBJECT
            OBJECT/
 Slink Def.: Current Directory
 Blink Def.: Current Directory
  Dice Def.: Current Directory
Instruction: <default object files>
            OBJECT
            FROM/ROOT

   Command: WITHPATH
      Type: Script Files
 lk Default: Current Directory
            LK:PREFS/
            S:
            SLINKWITH:
 Slink Def.: Current Directory
            SLINKWITH:
 Blink Def.: Current Directory
  Dice Def.: Current Directory
Instruction: WITH
```

(*) Default objects are any automatically loaded object file. Those are:
    . 'autorun.o' (AUTORUN)

```
    . 'overlay.o'  (OVERLAY)
    . 'library.o'  (AMIGALIBRARY)
    . 'stub.o'     (WARUNDEF)
```

   Note: patterns (or wild cards) are searched only into the current directory,
event the current directory does not exist into the path.

```
   See also:
       AMIGALIBRARY
       AUTORUN
       DEFICON
       FD
       FDLIB
       <filename>
       FROM/ROOT
       ICON
       LIBFD
       LIBRARY
       OVERLAY
       OVERLAYOBJECT
       WARUNDEF
       WITH
       Become Registred
```

1.107   lk V1.04 - (Plain)

```
                                    PLAIN
                              Default: FANCY
```

   Forbid any printer/console character (CSI) to be included in XREF file. To
reset to default use FANCY keyword.

```
   See also:
       FANCY
       HEIGHT
       MAP
       MARGIN
       SWAPFH
       WIDTH
       XREF
       Become Registred
```

1.108   lk V1.04 - (Priority)

```
                          PRIORITY or PRI <value>
                            LIBPRIORITY <value>
                        Default: no priority symbol
```

   This instruction defines the symbol PRIORITY. This symbol can be used for any
purpose. lk has a special header named 'priority.o' which will use this symbol

to automatically change the priority of the created command.

   The LIBPRIORITY instruction will defines the priority of an Amiga library,
also the __LIBRARYPRIORITY symbol. The default value of that symbol is zero.
That symbol is always defined.

   See also:
      SETLKPRI
      STACKSIZE
      Become Registred

1.109 lk V1.04 - (Prompt)

                                                                Page 11-85
                              PROMPT or -P
                            Default: no prompt

   Ask to user to types its own informations. This will ask new lk instructions
until the 'QUIT' command is typed or an empty line is given.

   See also:
      HELP
      Become Registred

1.110 lk V1.04 - (Public)

                                                                Page 11-86
                                 PUBLIC
                            Default: DEFAULT

   Link all hunks of following object files to be automatically loaded in public
memory. This is done regarless of any information present in those object file
hunks. It will last with the next memory instruction (ATTRIBUTES/CHIP/DEFAULT or
FAST.)

   Because this is a user specification, lk will not send any warning.

   This instruction can appear into the OVERLAY/END block.

   See also:
      ATTRIBUTES
      CHIP
      DEFAULT
      FAST
      HUNKMEMORY
      <filename>
      Become Registred

1.111 lk V1.04 - (Pure)

                          PURE or -R
                        Default: impure

   This instruction will turn the pure bit to ON into your destination file. This
is useless when you create a library.
   If the instruction SLINK has been used, then a table of relocation will be
created at the end of the near data hunk (if that hunk exists.) This table is
also required when you use the startup object file: 'cres.o', the   lk
automatically set the PURE flag when 'cres.o' or 'catchres.o' are used.

   When an executable file is read by lk, the current flag setting is kept for
the result. PURE will just force that bit to be set.

   See also:
     Become Registred

1.112   lk V1.04 - (Purpose)

                       The purpose of lk

   lk is a linker editor. It may be used to link any object all together. Objects
can be created with any language and compiler or assembler which generate normal
Amiga object files (Also not AztecC from manx.)

   Four ways to use lk:
     1. Link object files to generate an executable.
     2. Link object files to generate a library (.Lib file.)
     3. "Link" an executable to remove symbols and debugs and ask for small code,
        data and bss hunks.
        This may eventually be used to force all the program to be loaded in a
        specific memory type, reshoot relocation table from long to short, and
        some more.
     4. Fix code, data and bss hunks to a specific address.
        This is essential for people which are building games or ROMs.

   The first way is the first purpose of lk. When you need to create very large
programs, it will be more convenient to make each function in a single file, and
then link all object files to generate the program.

   The second function was created to enable anyone to generate libraries with a
more convenient way than with a JOIN function of the AmigaDOS shell. When files
pass trough lk for this purpose, lk may already link some hunks together (This
is done this way to ensure a better linking time. Of course, only hunks which
needs each other will be linked in a single one.)

   The third way can be useful to eliminate debug and symbol hunks from
executable files. And this is possible whenever you do not have the source
object files (The future AmigaDOS loadings will automatically be faster!)

   The forth option will be used only by people which does not want to use
AmigaDOS. This will create plain binary files without the reloc tables because
it already is positionned. Game and ROMs are only two examples, this features

might be used for other purpose like Unix codes (Which usuly are loaded at a
fixed address.)
  The debug and symbol tables are freed too.
  See also the following keyword for more informations:
                                 ADDRESS

  Whenever possible lk will prompt you about a bizarre object(s). To eliminate
most or all of lk warnings, see:
                           QUIET (Default)
                              WARNINGLEVEL
                              NOWARNING
To disable the reloc32 and ref32 to generate overflow (which should never
happen, but...) use the flag:
                              NOOVERFLOW32

  All source hunks with the same name will be linked together, creating one
hunk. Some special keywords may be used to force other hunks to be linked
together. Note that there is no way to avoid hunks with the same name to be
appended one after another. The given order of the source files will always be
keept inside the resulting file.
  Note: most of linkers, which support a function to link hunks together, will
not check memory requirements correctly. But lk does that for you. Anyway you
may not use these functions.

  Because some compilers or assemblers does not support a way to specify
specific memory requirements, 'lk' have special keywords for that purpose. See
also:
                              CHIP
                              FAST
                             PUBLIC
                           ATTRIBUTES
                            DEFAULT

  Some symbols may be specified on the command line or into the WITH files. This
creates an internal file. Also you may receive some errors and comments which
occur with the file named: 'Quick Defines'. See also:
                              DEFINE

  If you always have an error because some branchement and JSR/JMP with PC
generetes an overflow, you can use the instruction ALV to enable lk to create
ALV tables (Thus are table of JMP instructions.)
  In the future versions, this will support more instructions, but this will
probably generate some bugs at times.
  See also:
                              ALV

  lk supports relative definitions. This is complex to properly use those
informations and be able to link objects most of the time. lk will always try to
order hunks to fit the relative requirements. Except for tiny programs, a
relative of 8 bits should be avoid or used a very small amount of times. If you
use your own offset (See OFFSET instruction,) hunks may not be well ordered,
generating overflow errors. If no error occur, the resulting code will always be
working.
  The fact that lk keeps the BSS hunks at the end is important for users which
want to use XDATA, CLEARXDATA or XRELATIVE instructions.
  lk will try to let BSS hunks at the end of the resulting hunk, this may not be
possible when you use short and long relatives (16 and 32 bits.)

lk keep a BSS hunk when all relative informations point to a BSS hunk,
otherwise the hunk become a DATA hunk.
   lk will link a program with relative informations pointing to a hunk of CODE
type. In this case only hunks of type code can be pointed.
   In the future versions, lk will be able to handle large d16(An) addressing
mode and transform the instruction to accept 32 bits (Similarly the d8(An,Xn.S)
will be transformed.)
   See also:

                              OFFSET
                              XCODE
                              XDATA
                           CLEARXCODE
                           CLEARXDATA
                           KEEPXDATA
                         XRELATIVEDATA

   A safer utilisation of lk, is to run it with a destination file in your RAM:
disk. Because this is the first version, it may remains a big bug (Even I tryed
it in a lot of ways...)

1.113 lk V1.04 - (Qddiscard)

                                                            Page 11-88
                                QDDISCARD
                     Default: keep all pre-loaded files

   Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

   lk uses QuickDOS.library when that library is installed into your 'LIBS:'
assign. QuickDOS will also read and keep in memory any file used by lk. After a
while you will have a lot of memory used only by those files. If you need that
memory, just type:

        lk qddiscard

lk will display the name of the discarded files or not.

   See also:
     QDLIST
     QDPREPARE
     QDREMOVE
     QDSTART
     QDSTOP
     QuickDOS
     Become Registred

1.114 lk V1.04 - (Qdlist)

                                                            Page 11-89
                                 QDLIST
                  Default: do not display QuickDOS directory

Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

  lk uses QuickDOS.library when that library is installed into your 'LIBS:'
assign. QuickDOS will also read and keep in memory any file used by lk.
  At any time, you can know how much memory is used by typing:

        lk qdlist

lk displays the name of each file with their protection flags and respective
size, and at the end the number of file and the total size in use.

  See also:
      QDDISCARD
      QDPREPARE
      QDREMOVE
      QDSTART
      QDSTOP
      QuickDOS
      Become Registred

1.115   lk V1.04 - (Qdprepare)

                                                                   Page 11-90
                                 QDPREPARE
                              Default: link

  Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

  This instruction is useful to ask lk to load files through QuickDOS but not
link them. This way files will be ready for the first link.

  For instance:

      lk qdprepare lib:mylib.lib

  See also:
      QDDISCARD
      QDLIST
      QDREMOVE
      QDSTART
      QDSTOP
      QuickDOS
      Become Registred

1.116   lk V1.04 - (Qdremove)

                                                                   Page 11-91
                                 QDREMOVE
                       Default: keep the library in memory

Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

lk uses QuickDOS.library when that library is installed into your 'LIBS:'
assign. QuickDOS will always stay in memory. This takes about 6Kb (The library
code and the necessary memory buffers.) If you want to suppress QuickDOS from
your memory because you are finished with lk, type:

         lk qdremove

lk tries to free the library and display a message saying so when it succeed.

    See also:
       QDDISCARD
       QDLIST
       QDPREPARE
       QDSTART
       QDSTOP
       QuickDOS
       Become Registred

1.117   lk V1.04 - (Qdstart)

                              QDSTART
                         Default: QDSTART

  Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

  lk, by default, uses QuickDOS.library when that library is installed into your
'LIBS:' assign. You can forbid its usage by typing QDSTOP. Files loaded after
that instruction will be loaded as usuly. Now if you need to recover the default
usage you can use QDSTART, then files will be loaded through QuickDOS again.

    See also:
       QDDISCARD
       QDLIST
       QDPREPARE
       QDREMOVE
       QDSTOP
       QuickDOS
       Become Registred

1.118   lk V1.04 - (Qdstop)

                              QDSTOP
                         Default: QDSTART

  Only the commercial version will have this instruction fully supported.

Registred people will receive that version.

   lk, by default, uses QuickDOS.library when that library is installed into your
'LIBS:' assign. You can forbid its usage by typing QDSTOP. Note that only files
loaded after that instruction will be loaded as usuly. Then if you need to
recover the default usage you can use QDSTART.

   See also:
      QDDISCARD
      QDLIST
      QDPREPARE
      QDREMOVE
      QDSTART
      QuickDOS
      Become Registred

1.119   lk V1.04 - (Quickdos)

Page 17
                                QuickDOS© V1.00

                      QuickDOS was written by Alexis WILKE (c) 1994
                                All rights reserved.

   lk is offered with a library named QuickDOS. That library can be installed
with other usual libraries into your LIBS: directory.

   QuickDOS was created to increase by two or three the loading speed (I check
this on a product linking 751Kb of data!)
   To increase the speed at its maximum you will have to add buffers to the
device you use to stock your object files. One buffer is needed for each object
file plus one per directory (only one is necessary for a directory even that
directory is used for several objects.) If you use assigns, do not forget that
every file name is extended. QuickDOS uses only full paths. To know about thoses
paths, you may use the instruction QDLIST after the first usage of lk. An
overhead of 50 buffers is not a lux. To add thoses buffers use ADDBUFFERS
instruction from you CLI.

   QuickDOS will automatically be invoked by lk. Some useful keywords are
available to check QuickDOS:

      QDDISCARD
      QDLIST
      QDPREPARE
      QDREMOVE
      QDSTART
      QDSTOP

   QuickDOS can be used by any programmer. All necessary informations are given
into the QuickDOS archive file. This one is given with the registred version and
is available on your prefered BBS.

1.120   lk V1.04 - (Quickfatal)

QUICKFATAL or QF

lk always stops on a fatal error. It has a few number of fatal errors which
remains (like "Out of Memory",) Most of the errors are just overridden by
changing some data in a convenient way.

lk is composed of four general levels:

    1. command line parse
    2. builder
    3. cheker
    4. saver
   (5. external reference file creator)

each level stops lk when an error occurs during its process. Note that is the
way I used at start to develop lk. To restore that state use QUICKFATAL.
Otherwise lk processes the most of the things it can before to stop.

Note: when a source file is not founded, lk will continue and you should have a
lot of missing references appearing. This is the reason I though it could be
cool to keep QUICKFATAL.

If you find a fatal error which, you think, should not be endless, let me
know. Or eventually an error which should become a warning or vice versa.

    See also:
       WARUNDEF
       Become Registred

1.121   lk V1.04 - (Quiet)

QUIET

Intend to suppress all warnings and commentaries of any kind. However come
commentaries may appear when an error occur.

    See also:
       CALM
       NOWARNING
       WARNING
       WARNINGLEVEL
       Become Registred

1.122   lk V1.04 - (Release)

What about the next releases ?

If I have a lot of request about one point, it will also quickly be handled.
This will also depend upon the amount of documentation I have about the specific
things I will have to do.

   The following options will be included for my own pleasure and will also be
available for you.

   1. More listing with more details.
   2. More references check, verifying really everything.
   3. Eliminates eventual empty code/data/bss hunks in libraries, which may not
      be fully supported by all of existing linkers.

1.123   lk V1.04 - (Reloc)

                                                                Page 11-97
                              RELOC <value>

   Defines the number of reloc structures needed. The default is 1000. Like
SYMBOL, the number of reloc used by a link, may be known using VERBOSE keyword.

   See also:
      HUNK
      SYMBOL
      VERBOSE
      Become Registred

1.124   lk V1.04 - (Script)

                                                                Page 11-98
                          Scripts supplied with lk

   Rather than always think on how to do this and that, lk development system is
given with some useful scripts. You can modify then as required and create you
own scripts.
   The scripts given with lk have the 's' flag set. You set that flag with your
CLI instruction:
         PROTECT <filename> +S
This flag prevent the usage of the EXECUTE intruction every time you call a
script.

   There is a list of the available scripts:

      strip
      fd2lib
      fd2offset

 . strip <source> <destination>

   This script will ask to lk to load a file (which has to be an executable) and
save that file without any kind of debug. All debugs, symbols and advisory hunks
are suppressed. The organization of the hunks will remains the same.

. fd2lib

Creates a library or a header file with the function names defined within a
description ('.fd') file.

1. ASM

You can ask for an assembler include file. This will create a text file
with a list of reference. Usuly assembler programmers uses the 'fd2offset'
script.

XREF   _<funcname>

2. C

You can ask for a C header file. This will create a text file with a
list of function reference. Because the '.fd' files does not describe
suffisiantly the parameter types, only 'long' and 'void *' are used. You
may use VERBOSE and edit the resulting file to provide the correct types.

extern long <funcname>(type,type,...);

3. PASCAL

You can ask for a Pascal header file. This will create a text file with
a list of function references. Because the '.fd' files does not describe
suffisiantly the parameter types, only 'long' and '^long' are used. You
may use VERBOSE and edit the resulting file to provide each function of
the correct types.

extern Function <funcname>(a:type,b:type,...): long;

4. OBJECT

You can ask for an object file. In this case lk generates a library
(like Amiga.Lib) with each function which can be called with a C like
call, which means that parameters are stacked. You can use the option 'C'
to have the corresponding function calls.

. fd2offset

Creates a library or a header file with the offset generated from a
description ('.fd') file.

1. ASM

You can ask for an assembler include file. This will create a text file
with a list of offset.

_LVO<funcname>   =   <offset>

2. C

You can ask for a C header file. This will create a text file with a
list of #define.

#define _LVO<funcname>   <offset>

3. PASCAL

   You can ask for a PASCAL header file. This will create a text file with
   a list of constants.

   const _LVO<funcname> = <offset>;

4. OBJECT

   You can ask for an object file. In this case lk generates a library (not
   like Amiga.Lib) with each function offset defined as an XDEF. The
   correspong assembler programme would be:

        XDEF _LVO<funcname>
   _LVO<funcname>   =   <offset>

See also:
   WITH
   WITHPATH

1.125   lk V1.04 - (Setlkpri)

                         SETLKPRI <value>
                        Default: CLI priority

  This instruction defines the priority of lk. lk usuly takes the priority of
your CLI. This instruction enable you to fix the priority at a given value.

  The last call to SETLKPRI will be effective for all the duration of the link.

  See also:
     PRIORITY
     Become Registred

1.126   lk V1.04 - (Shortreloc)

                        SHORTRELOC or SR
                      SHORTRELOCOVERLAY or SRO

  Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

  The instruction SHORTRELOC can be used for version 37 and over of DOS. This
will transform any reloc hunk into a short 32 reloc hunk. It should enable you
to save up to 32Kb...

  You must also consider that the usage of this instruction disables the ability

of the earlier DOS versions to load your executable.

   The usage of this keyword moves all small relocs into the short reloc table.
This might enlarge your file in a so few cases that no internal test is done.
   You can use lk on all of your executable (even thoses of Commodore) to make
them smaller. Anyway, ensure that you have a copy of the original file.

   When you do not use this instruction, the short reloc tables are deleted and
replaced by normal reloc 32.

   If you want to generate (or load) files for the V37 of DOS, you will have to
use the following instructions:
     READSRV37
     WRITESRV37
which is a counter pass to the Commodore bugs.

   The lk overlays support short relocs at anytime while you use the default lk
overlay handler. This does not check the system version. Then the
SHORTRELOCOVERLAY instruction should be used at anytime when an overlay is
created. The short reloc of the overlay takes in account the version V37 bug.

   See also:
     MAKERELATIVE
     OVERLAY
     SMALL
     SMALLBSS
     SMALLCODE
     SMALLDATA
     SINGLE
     SINGLEBSS
     SINGLECODE
     SINGLEDATA
     READSRV37
     WRITESRV37
     Become Registred

1.127   lk V1.04 - (Shortv37)

                                                              Page 11-101
                               READSRV37
                               WRITESRV37
                 Default: warning when a HUNK_DREL32 is read
                   and writes HUNK_RELOC32SHORT has it should be

   Only the commercial version will have this instruction fully supported.
Registred people will receive that version.

   Because Commodore made a big mistake under the V37 of DOS loader, there is a
way to create dirty codes.

   The specification of READSRV37 will enable the usage of the value 1015
(HUNK_DREL32) as an equivalent to the value 1020 (HUNK_RELOC32SHORT.)
   This flag is not necessary, but it forbids the generation of a warning when a
1015 is founded into an executable.

The usage of WRITESRV37 should be prohibited. Do not use short reloc in V37,
to avoid this bug, that's all! Otherwise it will use the value 1015 instead of
1020 to save small relocation tables.

    See also:
        SHORTRELOC
        SHORTRELOCOVERLAY
        Become Registred

1.128   lk V1.04 - (Single)

                                  SINGLE
                           SINGLEBSS or SGB
                          SINGLECODE or SGC
                          SINGLEDATA or SGD
                               SINGLEHUNK
                          SINGLEUNIT or SGU
                            Default: NORM

   Note: I take the right to tell you that the SMALL???? instructions are more
valuable to produce clean executable than those ones.

   Ask for one hunk of specified type. While different type of memory may have
been defined for each hunk, a way of priority was defined. If only one type of
hunk exist, the single hunk will be of that type. If any of the hunk is for CHIP
memory the resulting hunk will be in chip memory. If any of the hunk is for FAST
memory, and no CHIP hunk exists, the resulting hunk will be in fast memory.
   The SINGLE instruction is an equivalent to all SINGLE??? instructions, except
SINGLEHUNK.
   The SINGLEHUNK instruction transforms all hunks into one single hunk of type
CODE, the BSS's will be transformed into large areas of zeroes. This instruction
should be used in conjunction with all other SINGLE??? instructions if you want
to have an x-hunk.

   This function generates a lot of warning if a lot of hunk have different
memory requirements. You may use the NOWARNING command to avoid all of them.

   In the case of a library (Note SINGLEUNIT only is valid) this instruction will
create a single unit. Most of the time this has the effect of destroying a C
library.

   Note: this instruction does not replace the ONEDATA one.

    See also:
        CREATELIBRARY
        FOR
        FRAGBSS
        FRAGCODE
        FRAGDATA
        NOWARNING
        NORM
        NORMBSS
        NORMCODE
        NORMDATA

```
        NORMHUNK
        NORMUNIT
        ONEDATA
        SMALL
        SMALLBSS
        SMALLCODE
        SMALLDATA
        SMALLHUNK
        SMALLUNIT
        WARNINGLEVEL
        Become Registred
```

1.129   lk V1.04 - (Slink)

                                    SLINK

   This instruction enables the creation of the special creators and destructors
tables. This will ask to lk to create the '___ctors' and '___dtors' arrays.

   This option automatically calls CC and BLOCKUNIT. Now the following
instructions are needed to ensure that lk will work correctly (as Slink):

```
        SLINK   ALV
        DEFINE  __BSSBAS=__BSS_BASE
                __BSSLEN=__BSS_LENGTH
                _LinkerDB=__DATA_START
                __BackGroundIO=___BackGroundIO
```

 or for resident:

```
        DEFINE  _LinkerDB=__DATA_BASE
                __RESBASE=__DATA_OFFSET
                __RESLEN=__DATA_SIZE
                _NEWDATAL=__DATA_LENGTH
                __BackGroundIO=___BackGroundIO
```

   I added an option for those symbols, even a very simple test would suffice for
two reasons. Not slow down the speed of lk, and enable anyone to use the special
symbols '___ctors' and '___dtors' like any other symbol.

```
  See also:
     Auto-Init/Exit
     BLOCKHUNK
     BLOCKUNIT
     CC
     DICE
     Become Registred
```

1.130   lk V1.04 - (Small)

                                  SMALL
                            SMALLBSS or SB
                           SMALLCODE or SC
                           SMALLDATA or SD
                              SMALLHUNK
                           SMALLUNIT or SU
                           Default: NORM

   Ask for the smallest distribution than possible. All hunks of the specified
type which have the same memory requirements will be linked together.
   If you really want all hunks put all togheter regardless of whatever memory it
should be loaded in, you may use SINGLE???? keyword instead.
   The instruction SMALL is a short hand to execute all small instruction at the
same time.
   The instruction SMALLHUNK will change DATA and BSS hunks in CODE hunks and all
hunks with the same memory requierement will be linked together. This
instruction should be used with all other SMALL??? instructions, if you want to
have a x-hunk.

   A common mistake is to use the SMALLBSS instruction to enable small data
hunks. A C program like always will have a hunk of data which have a smaller
size than the size specified into the HUNK_HEADER. To keep (and eventually
reduce) those small data, you have to use XDATA instruction.

   C programs (Like do Slink/Blink) usuly needs the option ONEDATA.

   In the case of a library (Note that SMALLUNIT only is valid) this instruction
will make hunks which needs each other to be linked at the library creation time
(See FOR for more information about how it goes.) This instruction should not be
used when you still develop your library, because it is really slow (It uses a
recursive algorithm which cannot make the stack crash!)

   See also:
      CREATELIBRARY
      FOR
      FRAGBSS
      FRAGCODE
      FRAGDATA
      FRAGHUNK
      NORM
      NORMBSS
      NORMCODE
      NORMDATA
      NORMHUNK
      NORMUNIT
      ONEDATA
      SINGLE
      SINGLEBSS
      SINGLECODE
      SINGLEDATA
      SINGLEHUNK
      SINGLEUNIT
      XDATA
      XCODE
      Become Registred

1.131   lk V1.04 - (Sort)

```
                                    SORT
                           Default: unsorted lists
```

   When this instruction is specified on the command line a sort is intented on
tables and symbols.

   The sorted informations:
      . number of referenced hunks (0 to max)
      . relocation tables (Smaller to bigger)
      . hunk types (CODE, DATA, BSS)
      . symbols (Sort with their values)

   This function is useless for AmigaDOS, this might be used only for some
debuggers which may not understand unsorted tables.

   The use of this instruction will ask lk to make a check on relocation tables.

   Note: lk does not sort, this enables it to be quicker.

   See also:
      Become Registred

1.132   lk V1.04 - (Stacksize)

```
                           STACKSIZE <value>
                             Default: 4 kb
```

   This instruction defines the value of the symbol named __STACKSIZE. That
symbol is always defined, the default value being 4096. The default is also the
minimum stack required.

   The object file 'stacksize.o' given with lk enables your executable(s) to have
an allocated stack of the defined size at start. This object needs to be present
into your command line to work.
   This is not necessary if you are creating an executable for the Workbench.
Note that the eventual icon associated with the executable will receive that
stack size.

   See also:
      LIBREVISION
      LIBVERSION
      VERSION
      Become Registred

1.133   lk V1.04 - (Start)

```
                           Start with lk®
```

lk® is a utility which works from CLI or from your Workbench. In that second
case you must have a prepared WITH file. In any case you should have a
preference file. You may use 'lkopts' utility to create those files.
   In order to run lk at its full power, you should assign the directory of lk as
'LK:'. That directory should include 'HELP/', 'PREFS/', 'GUIDE/', 'SOURCES/',
'LIB/', 'ICONS/', 'TOOLS/' directories.

   In any case, when you start lk on V36+, it checks global variables of the
AmigaDOS. You may defined a global variable with the instruction 'setenv' of
your CLI (you will have to copy variables from 'ENV:' to 'ENVARC:' to keep them
after a reset.) Those variables are not necessary. The available variables are:

   . LK/PREFS

   Gives the preference file name. If this file does not exist, the default
preference file will be searched as usuly. For more informations, please refer
to the following documentation:
                           The preferences file

   . LK/OBJPATH
   . LK/LIBPATH
   . LK/FDPATH
   . LK/WITHPATH
   . LK/ICONPATH

   Give the path of the given type. Those are unused when you use the preference
files: 'slink.prefs', 'dice.prefs' or 'blink.prefs'. For more information,
please refer to the following documentation:
                                 PATHS

   . LHAOPTS

   This variable is used by the archiver. If you have archived libraries and you
use 'lha' tool, then the LHAOPTS variable can be set to add some option flags to
the default ones. The actual command line to extract files is:

         lha -q e <library name> <file name>

A great option is also: '-w t:' to have the temporary files created in memory
instead of your hard drive.

   Start from CLI:

   . type lk followed by a list of instruction
   . create a WITH file and type 'lk with <with file>'
   . create a WITH file named lk.with and type lk alone

   Start from your Workbench:

   . double-click on lkopts to create the WITH file
   . click on lk icon and then double-click on a WITH file
     (You may select several WITH files)
       OUTPUT
         that type defines the output window; all lk messages will be sent to
         that window.
       PREFS

        that type defines the preferences file name. That name will overwrite
        the global variable LK/PREFS definition and it will be used instead of
        the usual default 'lk.prefs' file.
     OBJPATH
     LIBPATH
     FDPATH
     WITHPATH
     ICONPATH
        those types define the given acces paths. Those definitions will erase
        the one given in global variables.
        Note: if you define WITHPATH before or after PREFS it will be used to
        serach the preference file.

1.134   lk V1.04 - (Startup)

                            STARTUP <label name>
                        Default: use root (first) file

   Defines the startup label name. lk will search for that label and try to use
its hunk as the first hunk. The label should be defined in a hunk of CODE type
and defined with an offset of zero. In any case lk will be able to use that
label as the startup, but it has to be the beginning of the program.
   An alteration is effective over STARTUP when the instruction OVERLAY or
AUTORUN is used.

   The normal usage of lk uses three steps to determine the hunk of code to use
at start:

   . the very first hunk when it is of type CODE.
   . searchs the first hunk of code named "TEXT" (The name is case insensitive.)
   . the first hunk of code will be used instead.
   . if no hunk of code is available, no executable will be created.

   See also:
      AUTORUN
      OVERLAY
      Become Registred

1.135   lk V1.04 - (Stripdebug)

                            STRIPDEBUG or STRIP
                     Default: keep the known debugs and symbols

   Eliminates all debug and symbol tables. This is also an equivalent to NODEBUG
and NOSYMBOL used together.

   See also:
      ADDSYM
      CLEARADVISORY
      CREATEDEBUG

```
        CREATESYMBOL
        KEEPDEBUG
        HUNKADVISORY
        LEFTADVISORY
        NOBSSDEBUG
        NOCODEDEBUG
        NODATADEBUG
        NODEBUG
        NOLIBDEBUG
        NOLOCALSYMBOL
        NOOVLDEBUG
        NOSYMBOL
        SETADVISORY
        Become Registred
```

1.136   lk V1.04 - (Swapfh)

```
                            SWAPFH
                      Default: header/foot
```

If you prefer to have the foot replacing header line and vice versa in XREF
files, use this function.

```
  See also:
      FANCY
      HEIGHT
      MAP
      MARGIN
      PLAIN
      WIDTH
      XREF
      Become Registred
```

1.137   lk V1.04 - (Symbol)

```
                        SYMBOL <value>
                        Default: 16Kb
```

Defines the symbol buffer size. The default is 16384 bytes (16Kb). For a
faster system it may be enlarged while huge programs need more.

Use VERBOSE keyword to know about the current amount of memory used. Note that
there is no need to change this value and it is really useless to write a size
just under the amount of memory used.

In reality the lk system allocates a new table, each time that a table is
full.

```
  See also:
      HUNK
```

```
RELOC
VERBOSE
Become Registred
```

1.138   lk V1.04 - (Time)

```
                                 TIME
                          Default: no time
```

  Automatically adds the time into the copyright string. This is not fully
supported by the VERSION command but a lot of people use it.
  The time will appear only when a version and a copyright string are defined.

```
  See also:
    COPYRIGHT
    VERSION
    Become Registred
```

1.139   lk V1.04 - (To)

```
                          TO or -O <filename>
            Default: take the name of the first (root) object file
```

  Defines the destination file name. This may be unused and also an automatic
file name will be generated. Anyway this is safer while you develop and always
modify your lk instructions.

  The automatic file name creator will use the root file name (Or the very first
object file name when no FROM/ROOT was specified) deleting the eventual
extension '.o' or '.exe'. If such a name exist in the list of your object file
names then '.exe' extension will be appended. No check is provided if the option
NOEXE has been used.

  If the given file name is a directory name, the usual auto creator file name
will be used and the result will be saved into the given directory. The source
file name will have its path suppressed before the concatenation to the source
file name.
```
  Example:
    FROM prg:object/myprog.o
    TO ram:
  generates the destination file name:
    ram:myprog
```

```
  See also:
    <filename>
    FOR
    FROM/ROOT
    NOEXE
    Become Registred
```

1.140   lk V1.04 - (Tobss)

```
                    CODE2BSS
                 DATA2BSS or D2B
             Default: no transformation
```

  By testing each hunk of code and/or data, you may have the possibility to find
out some of them which may be a BSS. lk automatically transforms and saves them
into a BSS hunk.

  While your CODE (or DATA) hunk may have the same name than another CODE (or
DATA) hunk, this function may produce a warning when lk try to link hunks of the
same name together.

  This transformation occurs before the relative ordering and also is consequent
for it.

```
  See also:
     CLEARXCODE
     CLEARXDATA
     XCODE
     XDATA
     OFFSET
     Become Registred
```

1.141   lk V1.04 - (Unsnapshot)

```
                   UNSNAPSHOT
            Default: do not touch positions
```

  When an icon is defined, you may want to ensure the positions are undefined.
The use of this instruction will automatically undefine positions.

```
  See also:
     DEFICON
     ICON
     NOICON
     Become Registred
```

1.142   lk V1.04 - (Uselastdefine)

```
                USELASTDEFINE or ULD
               Default: use first define
```

  Uses the last definition found in libraries. This instruction is valid only if
CC or DICE has been used. Usuly only the first definition is used, the following
being forgotten. This is also another mode to use C libraries, which is not
common.

See also:
    CC
    DICE
    Become Registred

1.143   lk V1.04 - (Validnop)

VALIDNOP
Default: do not check NOP instruction

   Some times lk needs to append some data at the end of a hunk of code (Usuly
when you have ALVs.) lk checks if that hunk was finished by a NOP instruction
when VALIDNOP has been specified, and if so lk uses the space of that NOP.
Otherwise no check is done to be sure that lk is not taking a value for that
purpose.

   See also:
    ALV
    NOALV
    WARUNDEF
    Become Registred

1.144   lk V1.04 - (Verbose)

VERBOSE
Default: no verbose

   Make the linker speaking as much as it can. You will have the name of the
currently processed files, and some more info!

   See also:
    ECHO
    HELP
    Become Registred

1.145   lk V1.04 - (Version)

VERSION <value 1>[.[<value 2>]]
LIBVERSION <value>
LIBREVISION <value>
Default: no version symbols

   Defines the version and release numbers. This defines two symbols (XDEF or
extern.) The symbol VERSION is set to the first value and the symbol REVISION is
set to the second one.
   The fact that the version number is defined enables lk to create the copyright
string automatically including the system current date and time. Please, refer

to the COPYRIGHT command for more informations.

  When the period (.) does not appear, the REVISION symbol is not created. When
the period is not followed by anything, REVISION receives the value 0.

  Values may be written in signed decimal or hexadecimal when the value start
with a dollar sign ($).
  Example: 'Version -5.$A7'.

  The LIBVERSION and   LIBREVISION   instructions   were   added   for   Slink
compatibility. Slink always creates two symbols _LibVersion and _LibRevision,
those receive the default value of zero. If you want to define them you can use
the VERSION instruction or LIBVERSION and LIBREVISION respectivly. lk creates
two internal symbols __LIBRARYVERSION and __LIBRARYREVISION which will be
transformed via the DEFINE instruction like defined in the 'slink.prefs' file.
The copyright ID generator accept those two symbols as well as VERSION and
REVISION.
  The fact one of the LIB instructions is used set the flag AMIGALIBRARY. Please
refer you to that instruction to have all necessary informations.
  Note that the VERSION instruction will defines __LIBRARYVERSION   and
__LIBRARYREVISION equal to the symbols VERSION and REVISION respectivly.

  See also:
     AMIGALIBRARY
     CASEINSENSITIVE
     COPYRIGHT
     DEFINE
     LIBID
     TIME
     Become Registred

1.146   lk V1.04 - (Warning)

                              WARNING
                          Default: QUIET

  This instruction will ask to lk to display all warnings regardless of their
levels.

  See also:
     QUIET
     NOWARNING
     WARNINGLEVEL
     Become Registred

1.147   lk V1.04 - (Warninglevel)

                   WARNINGLEVEL or WL <value>
                          Default: QUIET

Suppresses absolutely all warnings having a level lower than the specified
value. A value over 99 will suppress all warnings and the value zero will enable
all warnings.

   The default warning level is 5. 10 is usuly a good enough value.

   The warning level appear when the given warning occur. That level is given as
the number after the period (.) character.

   See also:
      QUIET
      NOWARNING
      WARNING
      Become Registred

1.148   lk V1.04 - (Warundef)

                                     WARUNDEF
                          Default: error on undefined

   Transforms the error 'Undefined reference "<symbol name>".' into a warning.
This enables the creation of the resulting file, whenever unresolved symbols
existent. However, when there are unresolved symbol, lk creates a wrong (but not
automatically invalid) executable, with some long words, words and/or bytes
which remain undefined.

   If lk detects a function call with a missing define, a '_stub' function will
be defined to handle the call.

   You may define your own '_stub' function. It must be defined into a CODE hunk.
You may save its object file to a file named 'stub.o' in your current directory,
'LK:LIB/' or 'LIB:'. Otherwise you can use it as a library.
   The internal lk '_stub' function is a very simple code:

```
      XREF _stub
  _stub:
      MOVEQ   #$00,D0
      RTS
```

lk is also supplied with a nice '_stub' function. The declaration of the '_stub'
function is:

```
      extern void stub(char *function_name);
```

The variable 'function_name' is also defined into A0 register and is null when
no function name is available:

```
  if(function_name == (char *) 0) {
    function_name = "no function name";
  }
```

   Note: lk look for '_stub', '__stub' and '___stub' in this order. Any of those
may be defined.

See also:
    ASKUNDEF
    MAXREF
    VALIDNOP
    Error #003
    Become Registred

1.149   lk V1.04 - (Width)

                       FWIDTH or WIDTH OBJECTNAME <value>
                                   Default: 8
                       HWIDTH or WIDTH HUNKNAME <value>
                                   Default: 8
                       PWIDTH or WIDTH UNITNAME <value>
                                   Default: 8
                      SWIDTH or WIDTH SYMBOLNAME <value>
                                   Default: 16
                                WIDTH <value>
                           Default: 80 (30..255)

   Defines a width. WIDTH directly followed by a value defines the output line
width. This size does not include margins.

   Followed by a keyword width defines the size of that item in the tables:
    OBJECTNAME
        defines the source file length
        (Default is 8)
    UNITNAME
        defines the unit name length when exist
        (Default is 8)
    HUNKNAME
        defines the hunk name length
        (Default is 8)
    SYMBOLNAME
        defines the symbol name length
        (Default is 16)

   See also:
    FANCY
    HEIGHT
    MAP
    MARGIN
    PLAIN
    SWAPFH
    WIDTH
    XREF
    Become Registred

1.150   lk V1.04 - (With)

```
                         WITH [<filename> ...]
                          Default: OBJECT
```

   Specify a text file which contains some linker instructions. Before to execute
any WITH file, the current file (or the command line at start) will be entirely
parsed. Also WITH is not defined like an INCLUDE instruction from most of
compilers or assemblers.
   The WITH file is searched in each WITH path. If it is not found, the extension
'.lnk' (SAS link files) is append and lk checks again.

   Any number of WITH file may be specified, they will be proceeded in the given
order, one after another.

   WITH files may contain any instruction.

   An automatic WITH file may be created. Also lk will search for any file
called:

```
                              lkfile
                              lk.file
                              lk.with
```

in that order. If one of those files exist it will be used as a WITH file.   The
command line must be empty to enable one of those WITH file to be automatically
loaded. The current directory and then 'S:' will be scanned.

   On the top of that, at the very begining, lk scans the current directory and
then 'S:' for a file named 'lk.prefs'. If a such named file exists, it will be
loaded and executed as a WITH file before even the command line.

   See also:
     <filename>
     Become Registred

1.151   lk V1.04 - (Xhunk)

```
                       CLEARXCODE or CXC
                       CLEARXDATA or CXD
                        KEEPXDATA or KXD
                          XCODE or XC
                          XDATA or XD
                       XRELATIVEDATA or XRD
                Default: no X hunk (hunks are extended)
```

   By testing the end of each hunk of code and/or data, lk will try to delete
zeroes at that point. Also creating an X hunk. This means that, when loading the
hunk, the data at that position may or may not be zeroes; the hunk still has the
correct size in memory.
   To ensure those data to be cleared you must use the CLEARXCODE or CLEARXDATA.
But this create an executable which is valid only on machins having V37 or
higher (This is a mix of the ATTRIBUTES and the XHUNK commands.)

The XRELATIVEDATA sub-command will create an XDATA only with the relative data hunk. XDATA instruction override the use of XRELATIVEDATA. This instruction is usuly more powerful than the one of Slink because lk will re-order data hunks to search for the best configuration.

If an executable file does not run properly you may try to run lk on that executable, without the XCODE/XDATA options. Any hunk which was truncated will be restored.

The KEEPXDATA instruction enables a link of executable files with lk without a change to their near data hunk. This instruction is not necessary when you link object files.

All those instructions are only valid to produce executable files. However, all can be used on executable files to reduce some of their hunks.

```
See also:
    ATTRIBUTES
    CODE2BSS
    DATA2BSS
    FRAGBSS
    FRAGCODE
    FRAGDATA
    OFFSET
    SMALLBSS
    SMALLCODE
    SMALLDATA
    SINGLEBSS
    SINGLECODE
    SINGLEDATA
    Become Registred
```

1.152   lk V1.04 - (Xref)

                            XREF <filename>
                            Default: OBJECT

Defines a file where the resulting references tables will be saved. All symbols, files, references, etc... are saved. lk have some options to disable or enable some of those tables to appear in that file.
  Symbols in resulting table may be preceded by a sign to indicate their origin:
   <none> any object file
   *   library file(s)
   +   overlay file(s)
   ?   DEFINE instruction (See)

Some specific keywords are especially designed for XREF files and are listed here:
    MAP
      defines the contains of the external file
    PLAIN
      forbids printer characters
    FANCY
      enables printer characters

WIDTH
  defines the width of an item
HEIGHT
  defines the height of a page
MARGIN
  defines margins
SWAPFH
  swap foot and header lines

  Note: This function is executed even you are creating a file of type library.
But in that case you may not have really nice and useful informations.

  See also:
    FANCY
    <filename>
    HEIGHT
    MAP
    MARGIN
    PLAIN
    SWAPFH
    WIDTH
    Become Registred