DICE_Manual

COLLABORATORS			
	TITLE : DICE_Manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		October 9, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

DICE_Manual

1.1 The DICE 3.xx Command Referance

```
                                                Command Reference


As you may well have noticed by the size of this install DICE is not just a
standalone compiler infact Dillions/Drummonds Intuition C Enviroment
is a major collection of programming tools and here are the complete set of
commands to use them.. MF


DICE Command Referance

Command    : Purpose
===========+============================================================

               bintohex
                  : Convert binary files to Motorola S-records or Intel Hex
           : Dumps.  Used for programming ROM chips.
-----------+------------------------------------------------------------

               cat
                       : Shows contents of text files (takes wild cards)
-----------+------------------------------------------------------------

               ci
                        : Check in RCS Source (RCS).
-----------+------------------------------------------------------------

               co
                        : Check out RCS Source (RCS).
-----------+------------------------------------------------------------

               das
                       : DICE Assembler.
-----------+------------------------------------------------------------

               dc1
                       : DICE Compiler.
-----------+------------------------------------------------------------
```

```
               dcc
                        : DICE Compiler Front End.
-----------+-------------------------------------------------------

               dcpp
                       : DICE Preprocessor.
-----------+-------------------------------------------------------

               dd
                        : DICE Debugger.
-----------+-------------------------------------------------------

               dicehelp
                  : Fast online help utility.  Integrates with any editor.
-----------+-------------------------------------------------------

               diff
                     : File compare utility, three-way file compare utility
diff3      : (RCS).
-----------+-------------------------------------------------------

               dlink
                     : DICE Linker.
-----------+-------------------------------------------------------

               dmake
                     : DICE Make Utility. Automates compiles.
-----------+-------------------------------------------------------

               dme
                        : Text editor.
-----------+-------------------------------------------------------

               dobj
                       : Disassembles object & executable files.
-----------+-------------------------------------------------------

               dprof
                     : Code profiler.  Helps optimize code for speed.
-----------+-------------------------------------------------------

               dsearch
                   : Search for a string in a File.
-----------+-------------------------------------------------------

               du
                        : Determine Disk space Usage.
-----------+-------------------------------------------------------

               dupdate
                   : Distribution Maker.
-----------+-------------------------------------------------------

               expand
                    : Expand Wild cards to stdout with formatting.
-----------+-------------------------------------------------------
```

```
               fdtolib
                   : Creates Link Libraries from standard .FD files.
-----------+-------------------------------------------------------

               fdtopragma
                : Converts standard .FD files into
-----------+-------------------------------------------------------

               flush
                     : Flush libraries, etc. out of memory.
-----------+-------------------------------------------------------

               head
                      : Display first few lines of a file.
-----------+-------------------------------------------------------
hunks      : Show internal structure of object or executable files.
-----------+-------------------------------------------------------

               ident
                     : Identify files (RCS).
-----------+-------------------------------------------------------

               istrip
                    : Strip Comments from include files.
-----------+-------------------------------------------------------

               lbmake
                    : Link Library Creation Utility.
-----------+-------------------------------------------------------

               libtos
                    : Convert Large-Data Amiga.lib to Small-Data version.
-----------+-------------------------------------------------------

               loadabs
                   : For creating ROM images located at a specific address.
-----------+-------------------------------------------------------

               loadfile
                  : Load & hold a file in memory.
-----------+-------------------------------------------------------

               makeindex
                 : Create an index file for use by DICEHelp.
-----------+-------------------------------------------------------

               makeproto
                 : Create a file containing function prototypes for your code.
-----------+-------------------------------------------------------

               merge
                     : Three Way File Merge (RCS).
-----------+-------------------------------------------------------

               rcs
                       : Change RCS File Attributes (RCS).
```

```
-----------+-------------------------------------------------------

               rcsclean
                  : Clean up RCS Working Files (RCS).
-----------+-------------------------------------------------------

               rcsdiff
                   : Compare RCS Revisions (RCS).
-----------+-------------------------------------------------------

               rcsmerge
                  : Merge RCS Revisions (RCS).
-----------+-------------------------------------------------------

               rlong
                      : Display RCS History (RCS).
-----------+-------------------------------------------------------

               romable
                   : Generate romable image.
-----------+-------------------------------------------------------

               touch
                     : Update a file's datestamp without changing the file.
-----------+-------------------------------------------------------

               ttxsame
                   : Used by integrated error scripts to start TurboText.
-----------+-------------------------------------------------------

               Vmake
                     : Visual Make program.  A front-end for the compiler.
-----------+-------------------------------------------------------

               VOpts
                     : Visual Options.  Usually invoked from within VMake.
-----------+-------------------------------------------------------

               wbrun
                     : Used to simulate starting a program from Workbench.
-----------+-------------------------------------------------------

               wc
                        : Count words, lines, etc. in a file.
-----------+-------------------------------------------------------
```

1.2 bintohex Commands

```
FUNCTION
Generate Motorola S-Records

SYNOPSIS
bintohex inFile [-o outFile] -s[1,2,3] [-i] [-O offset]
```

```
DESCRIPTION
Bintohex converts a binary file into Motorola S-Record or Intel  Hex
format files.  Hex format files are used by many brands of EPROM or
Flash EPROM programming devices, and are accepted by by most vendors
of mask ROM.

inFile  Binary input file.

-o outFile
    Output file.  If no output file is specified, bintohex writes to
    the console.

-s[1,2,3]  Specify Motorola S-Record format.  Records are output in the
    form:

    SXnn[addr]DD..DDcc

   SX     : X is the file type, S1, S2 or S3.
   -------+-----------------------------------------------------
   nn     : Number of bytes, not including the two nn characters.
   -------+-----------------------------------------------------
   [addr] : Address.  S1=4 bytes, 64K limit.
   -------+-----------------------------------------------------
          : S2=6 bytes, 16MB limit.
   -------+-----------------------------------------------------
          : S3=8 bytes, 4GB limit.
   -------+-----------------------------------------------------
   DD..DD : Data, in hexadecimal pairs.
   -------+-----------------------------------------------------
   cc     : Line checksum. 0xff minus the sum of bytes on line.
   -------+-----------------------------------------------------

-i  Specify Intel Hex format.  64K bytes maximum.  Records are
    formatted:

    :xxaaaarrDD..DDcc

xx   : Number of bytes on line.
-----+-------------------------------------------------------------
aaaa : Address. Intel files are limited to 64K bytes of addressing.
-----+-------------------------------------------------------------
rr   : Record type: 00=normal, 01=end.
-----+-------------------------------------------------------------
DD   : Data (xx bytes worth).
-----+-------------------------------------------------------------
cc   : Checksum: 0 minus the sum of bytes on line.
-----+-------------------------------------------------------------

-O offset  Set initial address offset for hex file.
```

1.3 cat Commands

```
FUNCTION
Display file contents
```

```
SYNOPSIS
cat [files...]

DESCRIPTION
Like the AmigaDOS type command, cat displays one or more files on the
standard output.  Wildcards and multiple file names are accepted.
```

1.4 ci Commands

```
FUNCTION
Check in RCS Source

SYNOPSIS
ci [ options ] file ...

DESCRIPTION
Ci stores new revisions into RCS files.  Each file name ending in ,v
is taken to be an RCS file, all others are assumed to be working
files containing new revisions.  Ci deposits the contents of each
working file into the corresponding RCS file.  If only a working file
is given, ci tries to find the corresponding RCS file in the
directory RCS and then in the current directory.  For more details,
see the file naming section below.

For ci to work, the caller's login must be on the access list, except
if the access list is empty or the caller is the superuser or the
owner of the file.  To append a new revision to an existing branch,
the tip revision on that branch must be locked by the caller.
Otherwise, only a new branch can be created.  This restriction is not
enforced for the owner of the file, unless locking is set to strict
(see rcs).  A lock held by someone else may be broken with the rcs
command.

Normally, ci checks whether the revision to be deposited is different
from the preceding one.  If it is not different, ci either aborts the
deposit (if -q is given) or asks whether to abort (if -q is omitted).
A deposit can be forced with the -f option.

For each revision deposited, ci prompts for a log message.  The log
message should summarize the change and must be terminated with a
line containing a single . or a CTRL-\.  If several files are checked
in, ci asks whether to reuse the previous log message.  If the
standard input is not a terminal, ci suppresses the prompt and uses
the same message for all files.  See also -m.

The number of the deposited revision can be given by any of the
options -r, -f, -k, -l, -u, or -q.

If the RCS file does not exist, ci creates it and deposits the
contents of the working file as the initial revision (default number:
1.1).  The access list is initialized to empty.  Instead of the log
message, ci requests descriptive text (see -t below).

-r[rev]  assigns the revision number rev to the checked-in revision,
```

releases the corresponding lock, and deletes the working file. This is the default. Rev may be symbolic, numeric, or mixed.

If rev is a revision number, it must be higher than the latest one on the branch to which rev belongs, or must start a new branch.

If rev is a branch rather than a revision number, the new revision is appended to that branch. The level number is obtained by incrementing the tip revision number of that branch. If rev indicates a non-existing branch, that branch is created with the initial revision numbered rev.1.

If rev is omitted, ci tries to derive the new revision number from the caller's last lock. If the caller has locked the tip revision of a branch, the new revision is appended to that branch. The new revision number is obtained by incrementing the tip revision number. If the caller locked a non-tip revision, a new branch is started at that revision by incrementing the highest branch number at that revision. The default initial branch and level numbers are 1.

If rev is omitted and the caller has no lock, but he is the owner of the file and locking is not set to strict, then the revision is appended to the default branch (normally the trunk; see the -b option of rcs).

|| NOTE: On the trunk, revisions can be appended to the end, but
|| not inserted.

-f[rev] forces a deposit; the new revision is deposited even it is not different from the preceding one.

-k[rev] searches the working file for keyword values to determine its revision number, creation date, state, and author (see co), and assigns these values to the deposited revision, rather than computing them locally. It also generates a default login message noting the login of the caller and the actual checkin date. This option is useful for software distribution. A revision that is sent to several sites should be checked in with the -k option at these sites to preserve the original number, date, author, and state. The extracted keyword values and the default log message may be overridden with the options -r, -d, -s, -w, and -m.

-l[rev] works like -r, except it performs an additional co -l for the deposited revision. Thus, the deposited revision is immediately checked out again and locked. This is useful for saving a revision although one wants to continue editing it after the checkin.

-u[rev] works like -l, except that the deposited revision is not locked. This is useful if one wants to process (e.g., compile) the revision immediately after checkin.

-q[rev] quiet mode; diagnostic output is not printed. A revision that is not different from the preceding one is not deposited, unless -f

```
    is given.

-ddate  uses date for the checkin date and time.  Date may be specified
    in free format as explained in co.  Useful for lying about the
    checkin date, and for -k if no date is available.

 -mmsg  uses the string msg as the log message for all revisions checked
    in.

 -nname  assigns the symbolic name name to the number of the checked-in
    revision.  Ci prints an error message if name is already assigned
    to another number.

 -Nname  same as -n, except that it overrides a previous assignment of
    name.

 -sstate  sets the state of the checked-in revision to the identifier
    state.  The default is Exp.

 -t[txtfile]
    writes descriptive text into the RCS file (deletes the existing
    text).  If txtfile is omitted, ci prompts the user for text
    supplied from the standard input, terminated with a line
    containing a single . or C\.  Otherwise, the descriptive text is
    copied from the file txtfile.  During initialization, descriptive
    text is requested even if -t is not given.  The prompt is
    suppressed if standard input is not a terminal.

 -wlogin  uses login for the author field of the deposited revision.
    Useful for lying about the author, and for -k if no author is
    available.

  FILE NAMING
Pairs of RCS files and working files may be specified in 3 ways (see
also the example section of co).

1) Both the RCS file and the working file are given.  The RCS file
   name is of the form path1/workfile,v and the working file name is
   of the form path2/workfile, where path1/ and path2/ are (possibly
   different or empty) paths and workfile is a file name.

2) Only the RCS file is given.  Then the working file is assumed to
   be in the current directory and its name is derived from the name
   of the RCS file by removing path1/ and the suffix ,v.

3) Only the working file is given.  Then ci looks for an RCS file of
   the form path2/RCS/workfile,v or path2/workfile,v (in this order).

If the RCS file is specified without a path in 1) and 2), then co
looks for the RCS file first in the directory RCS, then in the
directory contained in the file RCS_LINK, followed by the current
directory.

DIAGNOSTICS
For each revision, ci prints the RCS file, the working file, and the
number of both the deposited and the preceding revision.  The exit
status always refers to the last file checked in, and is 0 if the
```

```
operation was successful, 1 otherwise.

SEE ALSO
co, ident, rcs, rcsdiff, rcsintro, rcsmerge, rlog, section .
```

1.5 co Commands

```
FUNCTION
Check out RCS Source;

SYNOPSIS
co [ options ] file ...

DESCRIPTION
Co retrieves a revision from each RCS file and stores it into the
corresponding working file.  Each file name ending in ,v is taken to
be an RCS file; all other files are assumed to be working files.  If
only a working file is given, co tries to find the corresponding file
in the RCS directory and then in the current directory.  For more
details, see the file naming section below.

Revisions of an RCS file may be checked out locked or unlocked.
Locking a revision prevents overlapping updates.  A revision checked
out for reading or processing (e.g., compiling) need not be locked.
A revision checked out for editing and later checkin must normally be
locked.  Co with locking fails if the revision to be checked out is
currently locked by another user.  (A lock may be broken with the rcs
command.)  Co with locking also requires the caller to be on the
access list of the RCS file, unless he is the owner of the file or
the superuser, or the access list is empty.  Co without locking is
not subject to accesslist restrictions, and is not affected by the
presence of locks.

A revision is selected by options for revision or branch number,
checkin date/time, author, or state.  When the selection options are
applied in combination, co retrieves the latest revision that
satisfies all of them.  If none of the selection options is
specified, co retrieves the latest revision on the default branch
(normally the trunk, see the -b option of rcs).  A revision or branch
number may be attached to any of the options -f, -l, -p, -q, -r, or
-u. The options -d (date), -s (state), and -w (author) retrieve from
a single branch, the selected branch, which is either specified by
one of -f,..., -u, or the default branch.

A co command applied to an RCS file with no revisions creates a
zero-length working file.  co always performs keyword substitution
(see below).

-r[rev]  retrieves the latest revision whose number is less than or equal
     to rev.  If rev indicates a branch rather than a revision,
     the latest revision on that branch is retrieved.  If rev is
     omitted, the latest revision on the default branch (see the -b
     option of rcs) is retrieved. rev is composed of one or more
     numeric or symbolic fields separated by ..  The numeric
```

equivalent of a symbolic field is specified with the -n option of the commands ci and rcs.

-l[rev] same as -r, except that it also locks the retrieved revision for the caller. See option -r for handling of the revision number rev.

-u[rev] same as -r, except that it unlocks the retrieved revision (if it was locked by the caller). If rev is omitted, -u retrieves the latest revision locked by the caller; if no such lock exists, it retrieves the latest revision on the default branch.

-f[rev] forces the overwriting of the working file; useful in connection with -q. See also the section on file modes below.

-p[rev] prints the retrieved revision on the standard output rather than storing it in the working file. This option is useful when co is part of a pipe.

-q[rev] quiet mode; diagnostics are not printed.

-ddate retrieves the latest revision on the selected branch whose checkin date/time is less than or equal to date. The date and time may be given in free format and are converted to local time. Examples of formats for date:

22-April-1982
17:20-CDT
2:25 AM
Dec. 29, 1983
Tue-PDT, 1981
4pm Jul 21 (free format)
Fri, April 16 15:52:25 EST 1982 (output of ctime).

Most fields in the date and time may be defaulted. co determines the defaults in the order year, month, day, hour, minute, and second (most to least significant). At least one of these fields must be provided. For omitted fields that are of higher significance than the highest provided field, the current values are assumed. For all other omitted fields, the lowest possible values are assumed. For example, the date "20, 10:30" defaults to 10:30:00 of the 20th of the current month and current year. The date/time must be quoted if it contains spaces.

-sstate retrieves the latest revision on the selected branch whose state is set to state.

[login] retrieves the latest revision on the selected branch which was checked in by the user with login name login. If the argument login is omitted, the caller’s login is assumed.

joinlist
generates a new revision which is the join of the revisions on joinlist. Joinlist is a comma-separated list of pairs of the form rev2:rev3, where rev2 and rev3 are (symbolic or numeric) revision numbers. For the initial such pair, rev1 denotes the revision selected by the above options -r, ..., -w. For all other pairs,

```
    rev1 denotes the revision generated by the previous pair.  (Thus,
    the output of one join becomes the input to the next.)

    For each pair, co joins revisions rev1 and rev3 with respect to
    rev2.  This means that all changes that transform rev2 into rev1
    are applied to a copy of rev3.  This is particularly useful if
    rev1 and rev3 are the ends of two branches that have rev2 as a
    common ancestor.  If rev1 < rev2 < rev3 on the same branch,
    joining generates a new revision which is like rev3, but with all
    changes that lead from rev1 to rev2 undone.  If changes from rev2
    to rev1 overlap with changes from rev2 to rev3, co prints a
    warning and includes the overlapping sections, delimited by the
    lines

    <<<<<<<
    rev1
    =======
    rev3
    >>>>>>>

    For the initial pair, rev2 may be omitted.  The default is the
    common ancestor.  If any of the arguments indicate branches, the
    latest revisions on those branches are assumed.  The options -l
    and -u lock or unlock rev1.

KEYWORD SUBSTITUTION
Strings of the form $keyword$ and $keyword:...$ embedded in the text
are replaced with strings of the form $keyword: value $, where
keyword and value are pairs listed below.  Keywords may be embedded
in literal strings or comments to identify a revision.

Initially, the user enters strings of the form $keyword$.  On
checkout, co replaces these strings with strings of the form
$keyword: value$. If a revision containing strings of the latter form
is checked back in, the value fields will be replaced during the next
checkout.  Thus, the keyword values are automatically updated on
checkout.

Keyword    : Value
===========+============================================================
$Author: dice $   : The login name of the user who checked in the revision.
-----------+------------------------------------------------------------
$Date: 1994/08/18 05:39:56 $       : The date and time the revision was  ↩
   checked in.
-----------+------------------------------------------------------------
$Header: /home/dice/com/master/Doc/RCS/dice_commands.doc,v 30.8 1994/08/18
05:39:56 dice Exp dice $   : A standard header containing the full pathname
       : of the RCS file, the revision number, the date, the author,
       : the state, and the locker (if locked).
-----------+------------------------------------------------------------
$Id: dice_commands.doc,v 30.8 1994/08/18 05:39:56 dice Exp dice $
: Same as $Header: /home/dice/com/master/Doc/RCS/dice_commands.doc,
:v 30.8 1994/08/18 05:39:56 dice Exp dice $, except more useful.
:Rather than the full path name as, this leaves just the file name.
-----------+------------------------------------------------------------
$Locker: dice $   : The login name of the user who locked the revision
       : (empty if not locked).
```

```
    -----------+-------------------------------------------------------------
    $Log: dice_commands.doc,v $
# Revision 30.8  1994/08/18  05:39:56  dice
# .
#
# Revision 30.0  1994/06/10  17:57:04  dice
# .
#
# Revision 30.0  1994/06/10  17:57:04  dice
# .
#       : The log message supplied during checkin, preceded by a
            : header containing the RCS file name, the revision
            : number, the author, and the date.  Existing log
            : messages are NOT replaced.  Instead, the new log
            : message is inserted after $Log: dice_commands.doc,v $
# Revision 30.8  1994/08/18  05:39:56  dice
# .
#
# Revision 30.0  1994/06/10  17:57:04  dice
# .
#
# Revision 30.0  1994/06/10  17:57:04  dice
# .
#.  This is useful for
            : accumulating a complete change log in a source file.
 -----------+-------------------------------------------------------------
 $RCSfile: dice_commands.doc,v $  : The name of the RCS file without path.
 -----------+-------------------------------------------------------------
 $Revision: 30.8 $ : The revision number assigned to the revision.
 -----------+-------------------------------------------------------------
 $Source: /home/dice/com/master/Doc/RCS/dice_commands.doc,v $   : The full  ↩
    pathname of the RCS file.
 -----------+-------------------------------------------------------------
 $State: Exp $    : State of the revision as set by the -s option of rcs or
        : ci.
 -----------+-------------------------------------------------------------

 FILE NAMING
 Pairs of RCS files and working files may be specified in 3 ways (see
 also the example section).

 1) Both the RCS file and the working file are given.  The RCS file
    name is of the form path1/workfile,v and the working file name is
    of the form path2/workfile, where path1/ and path2/ are (possibly
    different or empty) paths and workfile is a file name.

 2) Only the RCS file is given.  Then the working file is created in
    the current directory and its name is derived from the name of the
    RCS file by removing path1/ and the suffix ,v.

 3) Only the working file is given.  Then co looks for an RCS file of
    the form path2/RCS/workfile,v or path2/workfile,v (in this order).

 If the RCS file is specified without a path in 1) and 2), then co
 looks for the RCS file first in the directory RCS, then in the
 directory contained in the file RCS_LINK, followed by the current
 directory.
```

```
EXAMPLES
Suppose the current directory contains a subdirectory RCS with an RCS
file io.c,v.  Then all of the following commands retrieve the latest
revision from RCS/io.c,v and store it into io.c.

co io.c
co RCS/io.c,v
co io.c,v
co io.c RCS/io.c,v
co io.c io.c,v
co RCS/io.c,v io.c
co io.c,v io.c

FILE MODES
If a file with the name of the working file exists already and has
write permission, co aborts the checkout if -q is given, or asks
whether to abort if -q is not given.  If the existing working file is
not writable or -f is given, the working file is deleted without
asking.

DIAGNOSTICS
The RCS file name, the working file name, and the revision number
retrieved are written to the diagnostic output.  The exit status
always refers to the last file checked out, and is 0 if the operation
was successful, 1 otherwise.

SEE ALSO
ci, ident, rcs, rcsdiff, rcsintro, rcsmerge, rlog, section .

LIMITATIONS
The option -d gets confused in some circumstances, and accepts no
date before 1970.  There is no way to suppress the expansion of
keywords, except by writing them differently.

BUGS
The option -j does not work for files that contain lines with a
single ..
```

1.6 das Commands

```
FUNCTION
DICE Assembler

SYNOPSIS
DAS asmfile [-o objectfile] [-E errorfile] [-nu]

DESCRIPTION
Das is a minimal 68000 assembler designed to assemble the output of
dc1.

|| NOTE: Das should not be used for assembly projects, it is meant
|| solely to deal with the output from the compiler.  Das supports
|| only a minimal subset of features.
```

```
-o objfile
    Specify object file, else writes to asmfile.o.

-E errorfile
    Specify file for errors, else diagnostics are sent to stderr.

    -nu  Specify that HUNK_UNIT hunks have no name.  This option is used
    for creating link libraries, to make the library smaller)

SEE ALSO
Chapter .
```

1.7 dc1 Commands

```
FUNCTION
DICE Compiler

SYNOPSIS
DC1 cppd_src_file [-o outfile] options

DESCRIPTION
DC1 is the compiler itself.  As input it requires a file preprocessed
by dcpp, and as output it provides assembly code ready for the das
assembler.  Normally one uses either dcc or VMake as a front end,
never directly invoking dc1.

The compiler generates absolute-data references and absolute code
references by default.  Do not confuse this with DCC's default, which
is small-data and small-code.

The compiler will put argument and auto variables into registers
according to register availability and usage.  It will use
A0-A1/D0-D1 for register variables whenever possible.  Consequently,
the most heavily used variables will be in registers even for very
large subroutines.

You should get into the habit of declaring automatic variables within
sub blocks rather than declaring all your autos at the top of the
procedure. Apart from making the code more modular, this will enable
the compiler to make better decisions when allocating register
variables.

DCC does not do any major contents tracking and redundant
instructions will be generated.  DAS will handle properly optimizing
branches and DAS has a peephole optimizer built in it to handle other
obvious redundancies. The compiler does some optimizations itself,
such as using bit instructions to handle special cases of &, |, and
^, include using BTST.

|| NOTE: volatile forces a data item NOT to be placed in a register.
|| register is treated as a hint only by the compiler.  const is
|| ignored by default but will force objects into the code section
|| given the -ms or -mS options (see below). Other type and storage
|| qualifiers are described in chapter .
```

```
 -S
-S0  Set alternate section names "libdata" and "libbss".

-Sd name  Set section name for data sections

-Sb name  Set section name for bss sections

-Sc name  Set section name for code sections

-SD name  Set section name for __far data sections

-SB name  Set section name for __far bss sections
```

The -S option allows you to modify the default section naming conventions. DICE uses data, text, and bss as defaults for the data, code, and bss sections.

The DICE c.lib is compiled with -S and the startup code (c.o) references these first to force c.lib's data to come before program data. The data ordering is then as follows:

1) Library Initialized Data

2) Program Initialized Data

3) Library BSS Space

4) Program BSS Space

As long as the program does not declare more than 64KBytes of initialized data it can be linked with the small-data model c.lib. Thus, large-data-model programs that declare more than 64KBytes of BSS space will still link with the small-data-model c.lib

This may be of no consequence because any __far declared data will be placed in a different data segment entirely. Simply declare your large arrays as __far and the rest may remain small-data

-d[#] Set debug mode. This isn't pretty, it is primarily used for diagnosing potential compiler problems.

-E file specify stderr file, any errors are appended to the file instead of to stdout. Useful for batch compiles

-R Tells the compile to remove (delete) the input file when it no longer needs it. The input file is usually a temporary preprocessor file and DCC will use this option to get DC1 to delete it as soon as possible.

-proto The main compiler will generate errors for any unprototyped function call.

-r Resident option. The main compiler will generate special autoinit code to initialize data-data relocations. This simplifies the work that DLink and the startup module must do to support residentable programs.

```
-v  Verbose

-o outfile
    Specify assembly output file name

   -mc  Small-code model (DCC default)

   -mC  Large-code model (DC1 default)

   -md  Small-data model (DCC default)

   -mD  Large-data model (DC1 default)

   -mw  Absolute-word addressing (overrides -md/-mD)

   -ma  Absolute addressing (no effect on DC1 operation)

    These options specify the memory model.  The small-code model
    uses PC-relative addressing and the small-data model uses
    A4-relative addressing

    -mw is used when making ROMable code and specifies that the
    ABSOLUTE WORD addressing mode be used instead of either absolute
    long or A4-relative.  Absolute word addresses are resolved at
    link time.

    || NOTE: This option should not be used when generating
    || executables meant to run on the Amiga.

  -ms0  (default) const is ignored

   -ms  string constants and const objs placed in code section

   -mS  string constants and const objs placed in code section

    These options control how const data items are handled, including
    string constants such as char *ptr = "abcd"; The default is to
    ignore the const type qualifier.

    If -ms is specified string constants and const data items are
    placed in the code section.  Local references to const data items
    use PC-RELATIVE addressing.  Remote references (from other
    modules) to const data items use ABSOLUTE LONG addressing.

    -mS works the same as -ms but remote references are forced to use
    PC-RELATIVE addressing.

    || NOTE: This can be dangerous and the final CODE size MUST BE
    || LESS THAN 32KBYTES!

    Usually it is safe to use -ms and, in fact, can save a lot of
    memory when combined with -r residentable programs because the
    string constants will not be duplicated for each running instance
    of the program.

SEE ALSO
```

dcc, dcpp, dlink

1.8 dcc Commands

```
FUNCTION
DICE Compiler Front End

SYNOPSIS
dcc options file file ...

DESCRIPTION
Dcc is the normal method of using the DICE system from a CLI window.
Dcc automatically invokes all the other parts of DICE, relieving you
of learning the grimy details.  Dcc is similar to the UNIX cc
command.  Many users will prefer to use VMake, the visual interface
to DICE. See chapter for full documentation.

Dcc options may occur anywhere on the command line but MUST occur
singly. That is, -c -a instead of -ca.  file arguments to options may
occur with or without an intervening space.  -oFILE and -o FILE are
both legal.

Files ending in .a or .asm are assumed to be assembly files.  Files
ending in .l or .lib are assumed to be library files.  Files ending
in .o or .obj are assumed to be object files.  All other files are
assumed to be C source files.

Normally DCC compiles all C source files, assembles all asm files,
and links the resulting object files with any specified .o files
together to produce an executable.  The output file may optionally be
specified with the -o option.  If not specified, a default output
filename based on the name of the input file is generated.  This
general action is modified by two options:

-c  DCC does NOT link, -o specifies the output object file

-a  DCC does NOT assemble (i.e.  leaves the .a file resulting from a
    compile).  -o specifies the output assembly file

    If neither option is given -o specifies the name of the resulting
    executable.

    The default object directory is T: and may be changed with the -O
    option.  The default temporary directory is also T: and may be
    changed with the -T option.  IF YOU HAVE LIMITED MEMORY you may
    have to specify that temporary files not be placed in T: either
    by re-assigning T: or using the -T option.  DICE goes much slower
    if temporary files must be written to a floppy or even a hard
    disk.

    ## WARNING: asm files are assembled with DAS, See the assembler
    ## reference if you intend to assemble non-DC1 generated assembly

  file  File to compile, assemble (.a), and/or link (.o, .lib)
```

```
@@file  containing further list of files, one per line.  (blank lines and
    lines beginning with ';' or '#' are ignored.  File may NOT
    contain further options).

E file  specify stderr file, any errors are appended to file instead of
    to stdout.  Useful for batch compiles

-c  Compile C source files and assemble into OBJECT files only (do
    not link).

-a  Compile C source files into ASSEMBLY files (do not assemble or
    link).

    Keep in mind the DAS will do further optimizations on the
    assembly file that you see.

   -l0  Skip linking default libraries (dlib:c.lib dlib:amigas.lib
    dlib:auto.lib), or standard startup (dlib:c.o and dlib:x.o).

    :: Beginner's Note: Do not use this option

    This option is used in special circumstances, such as when
    generating .libraries or .devices.

    ## WARNING: DICE is much more dependent on its startup code (c.o
    ## and x.o) than other compilers, do not link without the startup
    ## unless you know what you are doing.

-l lib  Include this library when linking.  (space is optional)

    :: Beginner's Note: Use -lm to link with the math library.  The
    :: math library is required before functions such as printf will
    :: work with floating point.

    See chapter   for more information on linking in custom
    libraries.

0  -2.0  -1.3
    Set the compiler to look for libraries and includes in the proper
    place.  Libraries and includes are different for each operating
    system release.  DICE eases compiling for, or using, different OS
    versions.  DICE inserts the revision number into library names
    ("amigas30.lib") and the include file path ("dinclude:amiga30/").

x  -2.x  -1.x
    Like the above options, except x specifies a specific minor OS
    revision.

   -L0  remove default library search path, including all explicitly
    specified (-L dir) directories up to this point.

-L dir  add the specified directory to the library search path.  If the
    object module or library can not be found in the current
    directory, directories specified with -L are searched.  -L
    directories are searched before the default library directory
    (DLIB:), assuming it was not removed with -L0 .
```

```
     Note that the directory path specified by -L is used to search
     for libraries AND object modules.

     A trailing '/' is optional

    -I0  Remove default include path from search list.  The default
     include path is dinclude: and dinclude:amiga/ (unless modified by
     -1.x and -2.x options)

 -I dir  When compiling scan this include directory (space is optional)
     The specified path takes precedence over defaults but defaults
     are NOT removed.

 -D define[=value]
     Pre-define a symbol

 -U  Undefine __STDC__, mc68000, _DCC, and AMIGA.

     :: Beginner's Note: Do not use any of these options

 -Houtfile=headerfile
     This option enables precompiled header file generation and
     operation. You may specify any number of -H options.  Example
     usage:

     -Ht:defs.m=defs.h

     When DICE encounters an #include <defs.h> this will cause it to
     first check for the existence of T:DEFS.M ...  if T:DEFS.M does
     not exist DICE will generate it from <defs.h>.  if T:DEFS.M does
     exist then DICE will use it directly and ignore <defs.h>

     You must specify the -H option both to have DICE create the
     precompiled header file and to have DICE use the precompiled
     header file.  Normally one makes operation as transparent as
     possible so as not depend on the option existing when porting to
     other environments.

     ## WARNING: A precompiled header file contains the preprocessed
     ## header and preprocessor macros.  These are set in stone!

     If you modify a #define that would normally effect preprocessing
     of a header file which is precompiled THE EFFECT WILL NOT OCCUR.
     It is strongly suggested you use precompiled headers ONLY with
     includes that are pretty much unchanging.  For example, the
     commodore includes or otherwise have an appropriate dependency in
     your DMakefile or make script to delete the precompiled header
     file whenever any of your headers are modified.

     Normally one has a single -H option that enables precompiling of
     a local header file, say defs.h, which contains #include's of all
     other header files.  Source modules would then #include <defs.h>

     :: Beginner's Note: Do not use this option

-o file  Specify output executable, object, or assembly file name
     depending on what you are producing.  The space is optional
```

```
-020  Generate code for the 68020 and later microprocessors

-030  Generate code for the 68030 and later microprocessors

-881  Generate inline FFP code for the 68881

-882  Generate inline FFP code for the 68882

  :: Beginner's Note: Do not use any of these options

  These options exist to produce 020 and 030 opcodes, and 881/882
  inline assembly for floating point operations.

 -md  small data model (default) uses A4-relative

 -mD  large data model uses absolute-long

 -mc  small code model (default) uses PC-relative

 -mC  large code model uses absolute-long

  :: Beginner's Note: Use only -mD if you declare more than
  :: 64KBytes of data.

  These options specify the default data and code model to use.
  The model may be overridden by use of the __near and __far type
  qualifiers on a variable by variable basis.

  DICE defaults to the small data and small code model, and is able
  to generate >32KBytes of code using the small code model so you
  should never have to use -mC.  Note that the DICE libraries have
  all been compiled with the small-data model, and certain
  applications may disrupt the base register, A4...  in this case
  use of the __geta4 type qualifier should be of use.  If worse
  comes to worse you can recompile a large-data model c.lib, but I
  suggest you try other solutions first.

-ms0  (default), Only const objects are put into a CODE hunk

 -ms  String constants are put into the read-only code hunk

 -mS  String constants are put into the read-only code hunk AND all
  external const references use NEAR addressing

  :: Beginner's Note: Use only -ms

  -ms0 turns off -ms/-mS in case you have it in your DCCOPTS
  environment variable and want to turn it off.

  Default operation (no -ms or -mS) puts const items into a
  read-only CODE hunk.  Locally declared objects are referenced
  using PC-REL while external objects (declared in some other
  module) are referenced using 32-BIT ABSOLUTE addressing.

  -ms will additionally make all string constants, such as "fubar",
  const and referenced via PC-REL.  -ms is an extremely useful
```

```
   option when you will never modify any of your string constants
   because the strings are not copied for multiple running instances
   of the program (if resident).

   -mS works like -ms, but in addition forces all external const
   references to use PC-REL addressing INSTEAD of 32-bit absolute
   addressing.

   || NOTE: This is a very dangerous option, do not use unless the
   || final code size is less than 32 kbytes.

   Using -ms along with -r can result in huge savings of memory due
   to the string constants being moved out of the data segment
   (which must be duplicated for each running instance of the
   program).

   ## WARNING: In all cases if you declare an object as const it
   ## must be extern'd as const in other modules or incorrect code
   ## will be generated.  This is true whether you use -ms/S or not.

 -mRR  registered arguments, strict

   This option controls the automatic registerization of procedure
   arguments.  Only those prototyped procedures declaring 4 or fewer
   arguments will be registered.  Values are passed in D0/D1/A0/A1
   according to the type of variable and availability of registers.

   -mRR generates a single registerized entry point and extends
   registerization to indirect function calls (that must be fully
   prototyped).

   -mRR assigns either the registered or normal entry point to
   function pointers depending on whether they are prototyped or not
   (and any calls made through these function pointers will use the
   registered args method).

   ## WARNING: -mR cannot be used if you make c.lib calls that take
   ## call-back functions as arguments.

   -mr and -mRR CAN be used, however with -mRR you must be careful
   to supply the registered entry point.

   ## WARNING: AMIGA.LIB routines that take call-back functions as
   ## arguments must be given non-registered entry points.

   Thus if you use -mRR you MUST qualify the procedure or function
   pointer type specification with __stkargs to entire it has a
   normal entry point.

-mw addr  Used for making romable executables, Do not use to create AMIGA
   executables

   :: Beginner's Note: Do not use this option

   This option is another data model, called the ABSOLUTE-WORD data
   model.  Source files compiled with this option generate
   absolute-word data references to access data objects instead of
```

A4-relative or absolute-long. The base of the data segment must be specified as decimal, 0octal, or 0xHEX.

Since absolute-word is used exclusive of A4-relative, the compiler will now use A4 for register variables. You may NOT mix -mw modules with small-data models.

The ROMABLE program is usually run on the executable generated by DLink to generate a ROM.

-ma addr Used for making romable executables, do not use to create Amiga executables

:: Beginner's Note: Do not use this option

This option specifies to the compiler and linker that the resulting code is intended to be relocated to a permanent data address, that specified by addr in decimal, 0octal, of 0xHEX.

Unlike -mw, -ma assumes that the data segment can be placed anywhere. The ROMABLE program is usually run on the executable generated by DLink to generate a ROM.

You may still specify a data model, -md or -mD, to Be with this option. Unlike -mw, -ma does NOT touch the A4 register and thus may be mixed with the small-data model. See the section on generating Romable code.

-rom Set up options for generating romable code

:: Beginner's Note: Do not use this option

Like -l0, -rom disables automatic inclusion of a startup file (you must specify your own) and libraries. However, x.o is still included to sink any autoinit code. Your startup code must handle the appropriate model and call autoinit code before calling your program main

This option is used to support ROMed firmware, i.e. non-Amiga executables. You should never link with c.lib. Instead, a new library, rom.lib, is available.

rom.lib contains no static or global references and thus works with any data model, and only completely self-contained routines are included. The only data rom.lib uses is stack-based. All rom.lib routines are completely reentrant, including [v]sprintf() !

-proto Prototype checking and optimizations

When this option is used, an ERROR message will be generated for any call that is not prototyped. This option is useful to ensure that you have properly prototyped routines (when you use prototypes), especially when floats and doubles are passed and indirect function pointers are used (they must be prototyped as well!).

```
   In the future this will enable stack-argument optimization.
   Currently, chars and shorts are extended to long's when pushed
   onto the stack for a subroutine call.  In the future if the
   -proto option is used these objects will be pushed as shorts and
   not extended.

-prof  enable profiling for source modules

-prof1  same as -prof

-prof2  enable profiling for source modules and c*p.lib

-prof3  enable profiling for source mods, c*p.lib, and amiga*p.lib

   Enable profiling.  You may compile some or all your source
   modules with profiling enabled.  Any -prof* option will enable
   profiling for compiled source modules.  -prof2 will cause DCC to
   link with a profiled c*p.lib while -prof3 will cause DCC to link
   with a profiled c*p.lib and amiga*p.lib (the ultimate).

   To profile c*.lib and/or amiga*.lib functions the equivalent
   c*p.lib and amiga*p.lib must exist.  These libraries are most
   likely lharc'd in DCC2:dlib/ or DCC3:dlib/ but if not, registered
   users may create any link library from the library source.

-r  Make executable residentable with separate CODE & DATA hunks

  -pr  Make executable residentable w/ no relocation hunks

  -pi  Make executable NON-residentable w/ no relocation hunks

   :: Beginner's Note: Just use -r to get residentable executables
   :: and do not worry about these other options.

   -pr/-pi generate 'position independent' code also useful for
   ROMed applications.  NOTE that -pi and -pr force const items to
   be referenced pc-relative as well, causing -ms and -mS to do the
   same thing (when combined with -pr/-pi)

   Code size is limited to 32k bytes when you use -pr or -pi

  Refer to the RESIDENTABILITY section in Chapter 5 for a
   discussion of these options

   || NOTE: You may not make data references within const declared
   || objects when using the -r/-pr options.

   This is because the CODE hunk is shared between running instances
   of the program and these address references would be different
   between the instances.

   However, if you are using the -ms option, string constants will
   be in the code section and thus no problem.

  -O outdir  Specify directory that is to contain output executable, object,
   or assembly files (used when specifying multiple source files)
```

```
   -O is useful to tell the compiler where to put the objects when
   you use dcc to compile and link a whole bunch of files at once.
   In this case, the -o option can still be used to specify where to
   put the final executable.

   || NOTE: The -O name is used as a prefix so if you are specifying
   || a directory be sure it has a ':' or '/' on the end.

 -R  If the compile resulted in errors or warnings, execute the ARexx
   script specified in dcc:config/dcc.config.  This activates the
   integrated error tracking features of DICE.

 -T tmpdir  Specify the temporary directory used to hold preprocessed source
   files and temporary assembly files...  files that will be deleted
   after use.

   || NOTE: The -T name is used as a prefix so if you are specifying
   || a directory be sure it has a ':' or '/' on the end.

   The default is T:.  This option is useful in low-memory
   situations where you may decide to put intermediate files
   elsewhere.  Putting intermediate files on your hard disc or
   floppy slows down compilation by an order of magnitude, but if
   you are running on a system with little memory you may not have a
   choice.

-s  Include symbolic debugging information in the executable (dlink
   option).

   This option includes the symbol table in the resulting executable
   and is passed to dlink.  When using DOBJ to disassemble an
   executable, DOBJ will use the symbol table to generate a more
   symbolic dump.

-S  Alternate section naming op for libraries

   When making libraries: uses alternate section naming conventions
   so that all initialized data in the module will be placed before
   any initialized data in non -S modules (i.e.  normal linked
   object files).  Any library BSS will be placed before non-library
   BSS.  Thus, the final ordering in the final executable will be:

   LIBDATA
   PROGRAMDATA
   LIBBSS
   PROGRAMBSS

   Thus, if your program contains >64K Bytes of BSS you can still
   link with a library that tries to reference its own BSS using the
   small-data model.  If your library declares only initialized data
   (i.e.  int x = 0; ), then you can link with the library even if
   your program declares >64KBytes of *initialized* data !

-v  Display commands as DCC executes them.

   -new Checks timestamps for source/destination and only
   compiles/assembles if object is outdated or does not exist.  Used
```

```
   to make DCC a standalone make.

-f  Fast  CTRL-C handling for 1.3

   This option is used for 1.3 only.  You MUST be using the
   Commodore shell (NewShell) and if you make programs resident you
   MUST use the Commodore C:Resident command.

   This option will probably not work if you use WShell or ARPShell
   under 1.3.  This option allows DICE to take short cuts to run
   sub-programs and allows CTRL-C to be propagated to said programs.
   This option is useful to set permanently in your DCCOPTS ENV:
   variable if you run under 1.3.  DICE under 2.0 has no such
   problems and will run sub programs optimally, including
   propagation of ^C.

-frag  FRAGment (linker option).Quake People.. DICE was here first

   Tell linker not to combine all code hunks together or combine all
   data hunks together.  Cannot be used if the -r or -mw options are
   used. Generally only useful if the large-data model is used.  Not
   entirely supported yet.

 -ffp  Set fp library for floats

   :: Beginner's Note: When using single precision floating point
   :: this option, use of the original ffp libraries, will make the
   :: program portable across all Amigas.

   Otherwise only amigas that have the Commodore
   MathIeeeSing*.library libraries will be able to run the program.

   If not specified, mathieeesingtrans.library and
   mathieeesingbas.library are used.  These are new 2.0 libraries
   that may not exist on all machines yet.

   If specified, mathtrans.library is used ..  Motorola's FFP float
   library.

   || NOTE: IF -ffp is used, be warned that conversion from floats
   || to doubles and back again is not entirely reliable.

  -d#  Set debugging level (# = a number), used for compiler diagnostics
   only.

 -d opts  Specify any combination of debugging options.  These options may
   be combined in one -d option.

   Currently no options are defined.

  -gs  Generate Dynamic Stack Code.  This generates code on every
   subroutine call to check available stack.  If available stack
   falls below 2K a new stack frame is allocated which will be
   deallocated when the subroutine returns.

   If the allocation fails, stack_abort() is called.  If this
   routine is not defined by you, the library stack_abort() will
```

```
    call abort().

    This option is extremely useful when compiling UNIX code that
    expects infinite stack.

 -chip  CHIP force (linker option).

    Tell linker to force all hunks into CHIP memory.  You should
    generally not use this option.  Instead, use the __chip keyword
    for those specific data items that need to be in CHIP memory.

    || NOTE: CHIP data items are accessed using the large-data model,
    || thus you cannot create residentable executables that contain
    || __chip declarations Unless they are also const objects --
    || read-only.

 -unix  Causes DICE to use DLIB:uc*.lib instead of DLIB:c*.lib ...  the
    uc*.lib is exactly the same as the normal c*.lib except that all
    filenames are assumed to be UNIX names ..  that is, a beginning
    slash is converted to ':' (root of the current volume), "./" is
    ignored, and "../" is converted to "/" for all file accesses.

    This makes porting and usage of UNIX programs easier.

 -aztec  The front end attempts to run Aztec executables

 -sas  -lattice
    Identical.  The front end attempts to run SAS/Lattice executables

    These options allow one to write a single DMakefile able to
    handle compilation under any compiler, assuming the source is
    compilable under any compiler.  These are very limited options
    and may not work across new versions of Aztec or SAS/C

   -//  This option enables C++ style // comments.  This form of
    commenting begins with a // causing it and the remainder of the
    line to be considered a comment.

 -no-env  This option disables DCCOPTS.  DCC will not process options in
    the DCCOPTS environment variable.

    The ENV:DCCOPTS environment variable may contain additional
    options.

    ENV:must exist for DCC to run, even if you do not have a DCCOPTS
    environment variable.  If you do not use ENV:,assign it to RAM:

    1> assign env: ram:

 EXAMPLES:
 Example #1.  Compile hello.c to create executable "hello.":

1> dcc hello.c
1> hello

Example #2.  Compile hello.c to executable "fish" and put the object
file in X:
```

```
1> dcc hello.c -o ram:fish -TX:

Example #3.  Compile hello.c to and object file  in RAM, then link
with symbols:

1> dcc -c hello.c -o ram:hello.o
1> dcc ram:hello.o -o ram:hello -s

Example #4.  Compile foo.c and link with an already compiled object
file gar.o to produce an executable.  foo.o is placed in T:

1> dcc foo.c gar.o -o ram:foogar

SEE ALSO
das, dc1, dcpp, dlink
```

1.9 dcpp Commands

```
FUNCTION
DICE Preprocessor

SYNOPSIS
dcpp sourcefile [-o outfile] [-I includedir ...] options

DESCRIPTION
DCPP is a C preprocessor.  C code is first preprocessed, then
compiled.  The preprocessing step resolves all # operators, like
#define and #include, and generally prepares the C code for
compilation.  Most programmers use dcc or VMake, and do not invoke
dcpp directly.

Dcpp automatically scans DINCLUDE:, DINCLUDE:PD/, and DINCLUDE:AMIGA/
.  Any -I option directories are searched in sequence BEFORE dcpp's
default search path. The last default directory, DINCLUDE:AMIGA/, may
be modified with the -1.3, -2.0 and -3.0 options.

Note that DINCLUDE:PD/  is meant to be a place to put public domain
header files so as not to clutter the top level DINCLUDE: directory.

As with all DCC commands, the space between the option and an
associated file/dir argument is optional.

The following symbols are defined by default

Symbol        : Type        : Usage
==============+=============+======================================
__LINE__      : integer     : Current line number.
          : constant    :
--------------+-------------+--------------------------------------
__DATE__      : string      : Current date.
--------------+-------------+--------------------------------------
__TIME__      : string      : Current time.
--------------+-------------+--------------------------------------
__FILE__      : string      : Current file.
```

```
-------------+-------------+---------------------------------------
__BASE_FILE__ : string      : Base source file.  Allows an include
          :         : file to know which C file included it
-------------+-------------+---------------------------------------
__STDC__      : boolean     : Indicates ANSI compiler.
-------------+-------------+---------------------------------------
mc68000       : integer     : Indicates Motorola CPU.
          : constant    :
-------------+-------------+---------------------------------------
_DCC          : integer     : Indicates the DICE system.
          : constant    :
-------------+-------------+---------------------------------------
AMIGA         : integer     : Everyone's favorite computer.
          : constant    :
-------------+-------------+---------------------------------------
_FFP_FLOAT    : boolean     : Set if single precision floats are in
          :         : Fast Floating Point format
-------------+-------------+---------------------------------------
_SP_FLOAT     : boolean     : Set if single precision floats are in
          :         : IEEE-SING format (default).
-------------+-------------+---------------------------------------
```

-1.x -2.x -3.x
    Selects operating system revision compatibility for the Preprocessor. If not specified, DCPP searches dinclude:amiga for amiga includes. If specified, DCPP searches dinclude:amigaNN for the includes.

    DCC supports this option and passes it along to dcpp. This allows developers to compile under any OS revision with the flick of an option. (Note: DCC also sets a different amiga.lib based on these options).

 -d[#] This option turns on DCPP debugging

-ofile This option sets the output file, otherwise stdout is used.

  -ffp Passed from DCC, tells preprocessor to define _FFP_FLOAT. If not specified, preprocessor defines _SP_FLOAT. This exists to better support alternate floating point models in header files.

-Dvar[=val]
    This option predefines a symbol or macro.

-E file specify stderr file, any errors are appended to file instead of to stdout. Useful for batch compiles

-U This option undefines the following symbols:

    __STDC__ mc68000 _DCC AMIGA

-Hprecomp=header
    Enable use/creation of precompiled header files. See chapter for more information.

    -Ht:defs.m=defs.h

```
   -I0  This option causes DCPP to *NOT* include any default directories
    in the include search list.

-I dir  This option adds the specified directory to the include search
    list.  A hanging slash on the end of the path is not required.
    The space is optional.

   -//  Enable C++ style // comments.  The remainder of the line after //
    is encountered is interpreted as a comment.  This differs from /*
    style commenting in that no explicit comment-terminator is
    required.

-notri  Disable tri-graph scan pass.  Note that the tri-graph pass is
    implemented in assembly and does not slow down preprocessing in
    any noticeable fashion, you should not disable tri-graphs unless
    you need to.

   SEE ALSO
   dcc, dc1
```

1.10 dd Commands

```
FUNCTION
DICE debugger

DESCRIPTION
dd is a simple symbolic debugger.  dd allows you to single step and
display much internal information about the operation of your code.
Documentation for dd is in the dcc:doc directory of your DICE
installation.
```

1.11 dicehelp Commands

```
FUNCTION
Fast Online Help Utility

SYNOPSIS
DICEHelp searchitem

DESCRIPTION
DICEHelp can quickly retrieve help on any DICE topic.  From a CLI,
simply type your request as above.  From the Workbench, just click on
the DICEHelp icon.

Within many text editors, "hotkeys" have been established to link
with DICEHelp.  A key will either search for information on the word
under the cursor, or bring up a box for your selection.  DICEHelp
uses a "fuzzy" search, so your never have to worry about getting the
correct case or suffix.

SEE ALSO
Chapter .
```

1.12 diff Commands

```
FUNCTION
File Compare Utilities

SYNOPSIS
diff options fileA fileB

DESCRIPTION
diff is used to compare the contents of two text files. Diff3
compares three files.  Lines that are the same in all files are not
printed.  Lines that are different are shown with arrows.  Note that
the arrow "points" at the file from which the line came:

< Lines found in fileA, but not in fileB.
> Lines found in fileB, but not in fileA.
```

1.13 dlink Commands

```
FUNCTION
DICE Linker

SYNOPSIS
dlink options files libraries

DESCRIPTION
The final step in creating an Amiga program is linking.  Normally the
linker is invoked as needed by dcc or VMake.

Options may occur anywhere on the command lines.  Any file ending in
.o or .obj is assumed to be an object file.  Any file ending in .l or
.lib is assumed to be a library.  Any file name beginning with @@
specifies a text file containing a further list of files.

File ordering is maintained.  Section ordering is maintained.  All
sections of the same name are coagulated together with ordering
maintained.

|| NOTE: Inter-section ordering is not maintained within a library
|| since library modules are random included.  However, ordering is
|| maintained *between* libraries.

All object files specified are included in the final executable.  All
libraries specified are searched at the point they are specified
(that is, specifying an object file that references a symbol defined
in a library specified BEFORE the object file will cause an undefined
symbol error).  Normally an object file is specified after a library
to terminate an autoinit or autoexit section.

You do not have to order object files within a library, DLink will
automatically make as many passes as required to handle all internal
library references.  However, ordering object files will make DLink
go faster.
```

```
Symbols defined in object files override symbols defined in
libraries. Symbols defined in libraries specified before later
libraries override symbols defined in later libraries.  Symbols
defined in a library and also defined in a later specified object
module causes an error. -o execname name of executable

-s  Include symbolic information.

    || NOTE: if -r is used symbolic info for the data sections will
    || point to the statically init'd stuff, NOT The actual data
    || space (in BSS) referenced by the code.  This is a bug.

 -frag  Fragment output file (default is to coagulate all hunks of the
    same type regardless of name).  If frag is specified then only
    hunks of the same type AND name are coagulated.

    see fragmentation note at bottom

-r[es]  Resident link.

   -pi  Position independent non-residentable (i.e.  only one copy of the
    data but also no relocation hunks)

    -pr residentable position independent

-Ppostfix  specify library name postfix.  If DLink cannot find the library
      as specified it will append the postfix and try again.  Used by
    DCC to specify the memory model.

-mw addr  specify absolute data base

-ma addr  specify absolute data base

    Both options do exactly the same thing and are in duplicate to
    conform to DCC's options.

-mw
-ma   specify the base of data as a decimal, 0octal, or 0xHEX address.
    You must use the -r option in conjunction with these options.

    DLink will resolve all Absolute-Word addresses but not all
    Absolute-Long addresses.  This is left up to the ROMABLE program
    which generates a raw binary image of the program that can then
    be transferred to an EPROM.

    || NOTE: Do not use this option when generating Amiga
    || executables.

 -d[#]  debug mode (spews lots of debugging junk out)

 -E file  specify stderr file, any errors are appended to the file instead
     of to stdout.  Useful for batch compiles

 -chip  chip-only - forces all hunks into CHIP memory

   -L0  remove default library search path, including all explicitly
    specified (-L dir) directories up to this point.
```

```
 -L dir  add the specified directory to the library search path.  If the
    object module or library can not be found in the current
    directory, directories specified with -L are searched.  -L
    directories are searched before the default library directory
    (DLIB:), assuming it was not removed with -L0 .

    Note that the directory path specified by -L is used to search
    for libraries AND object modules.

    A trailing '/' is optional

  -Ppostfix  This allows you to specify -lc -Ps and DLink will automatically
    look for cs.lib ...  you can specify a postfix that occurs before
    the .lib in the library name here.  If DLink cannot find the
    library as it is named by default it will try it with the
    postfix.

    DCC uses this to supply the memory model to DLINK also allowing
    the user to say -lm in DCC and have it find MS.LIB if you are
    using the small-data model.

CREATING A LIBRARY
DLink libraries are standard Amiga libraries...  simply join one or
more object modules together and rename the result with a .lib
extension.

LINKER SYMBOLS
DLink generates the following special symbols to aid in program
startup:

Symbol         : Meaning
===============+=========================================================
__ABSOLUTE_BAS : Base of data in volatile space.  This symbol is NOT
           : defined for normal residentable programs since the
           : base address is not known (must be allocated
           : run-time)
---------------+---------------------------------------------------------
__DATA_BAS     : Base of data in non-volatile space.  This symbol
           : points to a read-only copy of the initialized data
           : for a program.  For Non-residentable programs this
           : is the same as __ABSOLUTE_BAS. For residentable
           : programs this points to a read-only copy of the
           : initialized data that the program can duplicate on
           : startup. For programs linked with an absolute base
           : address for data this points to the end of the CODE
           : section.  The ROMABLE program always generates a
           : ROM copy of the initialized data just after the
           : CODE section (which startup code must copy into
           : RAM)
---------------+---------------------------------------------------------
__DATA_LEN     : Length of data space is longwords.  i.e.
           : __DATA_LEN*4 yields the number of bytes of
           : initialized data.  This is used by startup code to
           : copy read-only initialized data to volatile space
           : (residentable and data-absolute programs)
---------------+---------------------------------------------------------
```

```
__BSS_LEN      : Length of bss space in longwords.  i.e.
           : __BSS_LEN*4 yields the number of bytes of
           : uninitialized (BSS) data.  This is used in
           : combination with __DATA_LEN to allocate the
           : DATA+BSS space for residentable programs, and clear
           : the BSS space for non-residentable and
           : absolute-data-base programs.  The BSS space occurs
           : after the DATA space unless the -frag option is
           : used.
--------------+-----------------------------------------------------
__RESIDENT5D   : This symbol is set to 0 if the -r option was used
           : and 1 if the -r option was not used.  If set to 1
           : (-r option)
--------------+-----------------------------------------------------
```

Programs linked with the -mw or -ma options obviously do not 'allocate' their data space since it is predefined. Most Amiga programmers will never use the -mw or -ma options, by the way.

SMALL DATA MODEL
The small data model uses A4 relative addressing. The linker sets up all relative offsets such that A4 must be initialized by startup code the BaseOfInitializedData + 32766 for A4-relative references to access the appropriate address.

RESIDENT
If the -r options is given then NO BSS SPACE is allocated after the data space... the startup code MUST allocate a data+bss space as shown above. DLink will give error messages for any absolute data references that occur (except the __DATA_BAS symbol which must be used to copy the static data to the newly allocated data+bss memory on program startup).

DLink will give an error message if any data-data reloco32s exist when you specify the -r option as such relocations would be incorrect when copied to the newly allocated data+bss space. DC1 understands this and will produce autoinit code to handle any such static data relocations that occur in your C code when the -r option is given to compile a C program.

However, DLink does allow data-data relocations to occur if an absolute data base is specified along with the -r option. This is used only when making ROMABLE code.

PC-RELATIVE
Because the linker will insert a jump table for PC-RELATIVE references to different hunks (after coagulation) or where the range is larger than +/-32K, data should not be placed into a code segment and be referenced via an external label(pc) unless you are positive the reference is within +/-32K. This can only happen when referencing between like-named code hunks. NOTE that the jump table is tagged onto the end of the section the jump(s) occur in and thus you do not want to have any autoinit/autoexit code that might possibly generate a jump table (since the whole idea with autoinit is that the code falls through to other autoinit code until the terminating section in x.o's RTS).

```
Currently dlink cannot handle inter-module PC-RELATIVE references
beyond +/-32K (i.e.  when one object file has more than 32K of code).
An error will occur.

Note that if -frag is used you cannot make PC-RELATIVE calls between
sections of differing names ever, or make a program resident.  The
-frag option is almost never used on untested.

|| NOTE: When -frag is specified, the linker will not create a
|| special combined data+bss hunk (so data and bss can both be
|| referenced with one base variable).

However, when -frag is NOT specified, the linker will stil not
necessarily combine ALL data hunks into one big hunk and ALL bss
hunks into one big hunk.  Any data or bss hunk with special upper
bits set (e.g. to force it into chip) is not combined into these
special hunks, and any data or bss hunk whos NAME begins with 'far'
(upper or lower case) will also not be considered.

EXAMPLE
This is what DCC gives the linker to link the program foo.c:

dlink dlib:c.o @tmp dlib:x.o -o ram:foo

Where tmp contains:

foo.o
dlib:c.lib
dlib:amiga.lib
dlib:auto.lib

Basically it tells dlink to link the startup code, c.o, then the
program object module(s) (foo.o), then c.lib, amiga.lib, and
auto.lib, then finally x.o.

DCC handles all this for you

auto.lib contains autoinit code for certain selected libraries,
including the dos.library.  Autoinit code is brought in whenever a
given library base symbol has been referenced BUT NOT DEFINED.
auto.lib defines the symbol and generates autoinit code to open the
library and autoexit code to close the library.  To maintain
portability you probably do not want to use this automatic
library-opening feature yourself, it is really meant to support
certain actions of the DICE library (such as floating point support).

x.o terminates the autoinit and autoexit sections with an RTS
instruction.  The autoinit and autoexit sections are called from the
startup code c.o.
```

1.14 dmake Commands

```
FUNCTION
Make Utility
```

```
SYNOPSIS
dmake [file]

DESCRIPTION
Make utilities automate complex compiles.  Makefiles can be
considered recipes for complex programs.  Makefiles contain
"dependencies," which are rules that say things like "if this header
file changes, recompile this C file."

The idea with DMake is to provide a powerful make utility through
general features rather than specialized hacks.  DMake is governed by
a few simple rules that can be combined into incredibly powerful
operations.

Generally you simply run DMake and have a list of dependencies in
your DMakefile which DMake then executes.  The DMakefile may contain
three different kinds of lines:

1) COMMENTS -- Any line beginning with a '#' is a comment and ignored

# This DMakefile generates an executable for fubar
# The compiler options are as follows ...

1) ASSIGNMENTS -- Any line of the form SYMBOL = ...  is considered an
   assignment.  Any variable references from within the assignment
   will be resolved immediately.

CFLAGS= -r SRCS= x.c y.c z.c

1) DEPENDENCIES: -- A line containing a list of symbols, a colon, and
   more symbols is assumed to be a dependency.  Note that you cannot
   have a raw filename with a colon in it as that confuses DMake.
   Instead, use an ASSIGNMENT variable.

Following the dependency line is zero or more command lines --
commands to run to resolve the dependency, terminated with a blank
line.

|| NOTE: Not only is a zero-command dependency allowed, it is
|| sometimes necessary.

A particular destination may have only ONE command list so if you
have something like

a.o : a.c

with a command list to compile the source into an object you can also
have another dependency such as 'a.o : defs.h' which would NOT have
any associated command list.

dst1 ...  dstN : src1 ...  srcN command1 command2 ...
dst1 ...  dstN : src1 ...  srcN command1 command2 ...

Finally, note that a dst* or src* symbol does not need to be a
filename.  It is perfectly valid to make up dummy names which are
then used as the lhs of a dependency that collects other dependencies
together.
```

DEPENDENCIES
When declaring dependencies you may use four different forms. The first form is to have a single destination and several sources. This is interpreted to mean that ALL the sources must be resolved before the single destination can be resolved via the command list for the dependency. The special variable, %(left), is set to the dst symbol and the special variable %(right) is set to ALL of the src symbols

For example, this form would be used to indicate that an executable depends on all the objects being resolved before you can run the link.

```
dst : src1 src2 src3 ...  srcN
```

The second form is the most useful in that it allows you to specify multiple 1 : 1 dependencies. Thus, you can specify, for example, that each object file depends on its source file counterpart for ALL the files in your project on a single line and have a single command list representing what to do (to compile a source file into an object, say).

In this case %(left) and %(right) are set to each dst* : src* pair individually and the command list is run for any individual pair that is out of date.

```
dst1 dst2 dst3 ...  dstN : src1 src2 src3 ...  srcN
```

The next form may be used to specify that many files depend on one file being resolved. An example of usage would be to make all the object files depend on one header file. The command list, if any, is run for each dst* : src pair with %(left) set to the current dst* and %(right) set to the single source.

```
dst1 dst2 dst3 ...  dstN : src
```

The last form is esoteric but sometimes useful. EACH dst* on the left hand side depends on the entire right hand side. You can have an arbitrary number of symbols on either side. %(left) will be set to a particular DST while %(right) will be set to all of the SRCs.

for example, you could specify $(OBJS) :: $(HDRS) -- make all objects depend on all headers causing a recompile to occur if any header is modified.

```
dst1 dst2 dst3 ...  dstN :: src1 src2 ...  srcI
```

WILDCARDS
DMake's most powerful feature is an easy to use file list replacement through options in a variable specification. You may insert the contents of any variable using the form:

```
$(SYMBOL)
```

Furthermore, you can make modifications to the contents of the variable on the fly using:

```
$(SYMBOL:wildcard)

only those files which match wildcard

$(SYMBOL:wildcard:wildcard)

matching files and also do a conversion
```

Simple */? wildcarding is used. A wildcard may contain a colon or other punctuation but if it does you MUST surround it with quotes. Here is a quick example:

```
SRCS= a.c b.c c.c d.c xx.a
OBJS= $(SRCS:*.c:"dtmp:%1.o")

all: echo $(OBJS)
```

Will Produce

```
dtmp:a.o dtmp:b.o dtmp:c.o dtmp:d.o
```

The first wildcard specification restricts which files from the list are to be taken -- 'xx.a' was ignored, as you can see. Each '*' or '?' in the first wildcard specification corresponds to %N specifications in the second wildcard specification. You can prepend, insert, or append text and freely mix or ignore items matched to create your destination file list.

This capability allows you to specify your source files EXACTLY ONCE in the DMakefile and then use the file munging capability to convert them to the object file list, etc...

You can embed variables within variables as with the following example (note that this time xx.a is included):

```
OD= dtmp:fubar/
SRCS= a.c b.c c.c d.c xx.a
OBJS= $(SRCS:*.?:"$(OD)%1.o")

all: echo $(OBJS)
```

Will produce

```
dtmp:fubar/a.o dtmp:fubar/b.o dtmp:fubar/c.o
dtmp:fubar/d.o dtmp:fubar/xx.o
```

As a side note, you may also specify '?' and '*' in the destination wildcard. These are considered dummies and are equivalent to %N where N is incremented from 1..9 for each '?' or '*' encountered.

You can use the capability anywhere in the DMakefile. Another common thing to do is restrict your link line to include only the object files and skip the headers:

```
$(EXE) : $(PROTOS) $(OBJS) $(HDRS)
dcc %(right:*.o) -o %(left)
```

```
ENVIRONMENT VARIABLES
2.0 local variables and 1.3/2.0 ENV: variables are fully accessible.
Under 2.0 you can also modify local variables on the fly.
DMake-specific variables override 2.0 local variables override ENV:
variables.

Under 2.0, any command containing <, >, `, or |, or is an alias, will
be run with System().  Thus, such commands may not be used to modify
local variables or the local environment.  Also, such commands cannot
be ^C'd due to the way AmigaDOS works.

EXAMPLE
The following is an example dmakefile.  The variable $(FILES) is set
to "main input output".  The next two variables are constructed from
$(FILES).  The command $(FILES:*:"*.c") tells dmake to take $(FILES),
look up each item ":*", and append two characters ":*.c".  The
results of the conversions are listed above.  This unique feature of
dmake makes for very elegant DMakeFiles.

The first rule is the default rule, executed if you just type dmake.
The rule, "sample:", states that the resulting program, "sample", is
made up from the files listed in $(FILE_OBJECTS). If the date on
"sample" is older than any dates in $(FILE_OBJECTS), the rule will
execute.

In turn, the next rule states that $(FILE_OBJECTS) ("main.o input.o
output.o") are made from $(FILE_SOURCES) ("main.c intput.c
output.c").  If any of the .o files are older than the corresponding
.c file, the rule executes.

In short, the makefile is a description of how source files inter
depend.  When any file changes, dmake figures the minimum number of
steps to regenerate the final result.  If you change just "input.c",
only "input.c" will recompile.

The last rule, "clean:", has no dependencies (nothing on the right
side of the :).  When executed, this rule deletes all the
compiler-generated files, but not the source code. To execute this
rule, type "dmake clean".  Any number of rules may exist in a single
DMakeFile.

                  :     DMakeFile - generic
------------------------------+-------------------------------------
FILES = main input output     : Equals "main input output" Set to
FILE_SOURCES =            : "main.c input.c output.c" Set to
$(FILES:*:*.c") FILE_OBJECTS  : "t:main.o t:input.o t:output.o"
= $(FILES:*:"t:*.o)           :
------------------------------+-------------------------------------
sample: $(FILE_OBJECTS)     dcc  : Rule: sample is made from
$(FILE_OBJECTS) -o sample     : FILE_OBJECTS (defined above as
                  : "t:main.o t:input.o t:output.o")
------------------------------+-------------------------------------
$(FILE_OBJECTS) :         : Rule: FILE_OBJECTS are made from
$(FILE_SOURCES)     dcc -c       : FILE_SOURCES (see above).
%(right) -o %(left)       :
------------------------------+-------------------------------------
clean:            delete  : Rule: "dmake clean" executes this
```

```
$(FILES) $(FILE_OBJECTS)      : delete command.
------------------------------+-------------------------------------

LINE CONTINUATION AND ESCAPES
Any line may be continued by terminating it with a backslash '\'.  It
is possible to escape the special characters '$' and '%' by doubling
them though this is only necessary if an open-parenthesis follows the
'$' or '%' and you do not want it interpreted as a variable.

It is possible to escape ':' and other special characters by
assigning them (or a string containing them) to a variable

COMMAND SHELL
Under 2.0 commands that do not contain any sort of redirection are
run with RunCommand().  If a command is an alias or there is some
sort of redirection in the arguments it will be run with System().

Under 1.3 everything is run with Execute()

ADVANCED CAPABILITIES
Now, you may have noted earlier that I said you could not have any
given left-hand-side with more then one command list.  Take, for
example:

a.o : a.c dcc %(right) -o %(left)

a.o : defs.h <--- illegal to put command list here

Actually, it isn't illegal.  When DMake encounters a dependency
without a command list it will automatically 'force' the next higher
level dependency of the same left-hand-side.  Therefore if you do not
have a command list for the lower level left-hand-side things work as
you expect.  Note that this requires all such null dependencies to
exist AFTER the one that has the command list.

If you do have two or more command lists for the same left-hand-side
they will run independent of each other according to their individual
right hand sides.  If several command lists apply then their order of
execution will be bottom-up

T FOR EXISTENCE
ther advanced feature quite useful in fully automating the
pilation process is the ability to create a directory tree on the
.  That is, if you have a projects called 'fubar' and want the
ects to go into the directory DTMP:fubar/ you might want to have a
endency that creates DTMP:fubar if it does not already exist.

 dtmp:fubar
X) : $(XX) makedir %(left)
```

1.15 dme Commands

```
FUNCTION
Editor
```

```
SYNOPSIS
Dme file

DESCRIPTION
Dme is a full screen programmable editor.  See chapter for complete
documentation.
```

1.16 dobj Commands

```
FUNCTION
Disassemble objects, executables, or Libraries

SYNOPSIS
DOBJ object_files [-o outfile] [-nd] [-nc] [-d[#]]

DESCRIPTION
DOBJ disassembles object modules and libraries into assembly.  DOBJ
is useful for, say, finding bugs in an assembler.  Most DICE users
will use DOBJ to browse through libraries and object modules, and
perhaps to check DAS optimizations...  for example, branch
optimizations will show up in disassembled object modules that are
not otherwise apparent by looking at assembly output (DCC -a).

DOBJ generates output to the console unless the -o option is used.
The -d option is for debugging the DOBJ program itself and not
normally used.

filename
 redirect output

-d[#]  Set debug level

-nd  Do not show actual data, only display symbol names

-nc  Do not disassemble actual code, only display symbol names

 DOBJ will replace hunk/offset references with symbol names when
 possible to yield a more readable output, and interprets each
 hunk according to its type (CODE, DATA, or BSS).

 There is NO limit to the size of the object file that may be
 disassembled, but it should be noted that DOBJ can take a while
 to resolve a large object file's symbols so be patient.  DOBJ
 does not take up much memory run-time, even when disassembling
 large object modules.

 ## WARNING: DOBJ does not does not understand any 68020/030
 ## instructions yet.
```

1.17 dprof Commands

```
FUNCTION
Code Profiler

SYNOPSIS
DPROF proffile [-call]

DESCRIPTION
This utility allows you to discover where time is used in your
program.  Careful analysis of the output can help you focus on areas
of your code that would be most valuable to optimize.

DPROF generates profiling output from the binary data file generated
by an executable which was compiled with profiling enabled.

In order to use DPROF you must compile your program with the -prof
option.  There are three levels of profiling:

Dcc Option : Effect
===========+============================================================
-prof1     : Profile only your code
-----------+------------------------------------------------------------
-prof2     : Profile your code and the standard C library
-----------+------------------------------------------------------------
-prof3     : Profile your code, the C library, and the Amiga library
       : tags
-----------+------------------------------------------------------------

To use -prof2 you must have installed DLIB:CSP.LIB (small data
profiled c.lib) or DLIB:CSRP.LIB (small data profiled c.lib for
registered arguments).

To use -prof3 you must have installed DLIB:AMIGASP20.LIB (small data
profiled amiga.lib) or DLIB:AMIGASRP20.LIB (small data profiled
amiga.lib for registered arguments).

USAGE

## WARNING: The profiling code is accurate to 20 microseconds under
## 2.0, 1/60 second under 1.3.  The profiling code itself will slow
## down a program by quite a bit but, in general, the system makes
## every attempt to filter out its profiling overhead in the
## statistics file (so the grand total time will be less then the
## actual amount of time the program took to run).

Note, however, that the results will be skewed somewhat anyway, not
only due to the overhead of the profiling code, but also due to task
switches and other system overhead.  To get accurate results you
should only run the executable that is to generate a .dprof file on
an unloaded system (i.e.  don't do anything else while the executable
is running).  Many calls to very short, quick routines will suffer
the most and numbers should be taken more in a qualitative fashion
than a quantitative fashion.

Keep in mind that it is not necessary to profile everything,
particularly for large projects.  You may want most of the system to
run at full speed while only profiling a small part of it at a time.
```

```
EXAMPLE
Given a program called example.c (you can clip this from the online
help and compile it):

void fubar1(void);
void fubar2(void);
void loop(long);

main(ac, av)
char *av[];
{
   short i;
   for (i = 0; i < 100; ++i)
   {
      fubar1();
      fubar2();
   }
   loop(10);
   fubar1();
   fubar2();
}

void fubar1()
{
   short j;
   for (j = 0; j < 10000; ++j);
   fubar2();
}

void fubar2()
{
   short j;
   for (j = 0; j < 100; ++j);
}

void loop(n)
{
   if (n)
      loop(n - 1);
}

Compile and the run the program, then dump the profile.  the DPROF
program automatically appends  '.dprof' onto the filename you
specify.

1> dcc test.c -o test -prof1
1> test
1> dprof test


@@($)DPROF V2.06.01  Sep 30 1991     test.dprof

GrandTotal:   539.53 mS

**** BY ROUTINE ****
_main   calls=1      total=  539.53 mS (100.00%) local=   10.37 mS (
```

```
 1.92%)
_fubar1 calls=101    total=  517.45 mS ( 95.90%) local=  507.75 mS (
 94.10%)
_fubar2 calls=202    total=   20.44 mS (  3.79%) local=   20.44 mS (
 3.79%)
_loop   calls=11     total=    0.96 mS (  0.17%) local=    0.96 mS (
0.17%)

The total numbers are time spent in the function.  The local numbers
are the same, except time spent calling other profiled subfunctions
have been subtracted out.

**** BY PARENT ****
_fubar2 calls=202    total=   20.44 mS
    From _fubar1 calls=101    total=    9.69 mS ( 47.43%)
From _main   calls=101    total=   10.75 mS ( 52.56%)

This section shows who called the function, how many times, and how
long that took.

**** COMBINED CALL TREE ****
_main   calls=1     tot=  539.53 (100.00)  loc=   10.37 (  1.92)
    _fubar1 calls=101   tot=  517.45 ( 95.90)  loc=  507.75 ( 94.10)
    _fubar2 calls=101   tot=   10.75 (  1.99)  loc=   10.75 (  1.99)
_loop   calls=1     tot=    0.96 (  0.17)  loc=    0.08 (  0.01)

The top line contains the same information from table 1.  Here main()
calls fubar1() 101 times, fubar1() takes 517 mS total time over these
calls. Also,  main() calls fubar2() directly 101 times and fubar2()
takes 10 mS over these calls.  Note that fubar2()'s time is not the
same as in table 1 because only those calls made from main() are
counted here.  Percentages are relative to main().  The profiled data
includes the entire call tree but for simplicity, recursive calls are
simply shown with <SELF>.

-call  You can request DPROF to print out the entire call tree.  This is
       done by adding the -call option to dprof.  Note, however, that
       this option may result in a huge amount of data printed out.  On
       the other hand, sometimes the data is quite useful especially
       when tracing subroutine stacking and other things.
```

1.18 dsearch Commands

```
FUNCTION
Search for string in a file

SYNOPSIS
Dsearch string files

DESCRIPTION
Dsearch is used for searching for a string in a series of files.
Wildcards are accepted.
```

1.19 du Commands

```
FUNCTION
Show Disk usage

SYNOPSIS
du path

DESCRIPTION
Du stands for "disk-usage".  This program returns disk space used by
a directory or volume.  It attempts to account for all blocks used by
a file, but the numbers are only estimates.

## WARNING: The results of the DU command will vary depending on the
## filesystem installed on the device.  This is especially noticable
## when DU is used on files in RAM:, as under 2.0 and abover the
## ram-handler packs multiple file headers and datablocks together.
```

1.20 dupdate Commands

```
FUNCTION
Distribution Maker

SYNOPSIS
DUPDATE dist-file dest-dir [options] [DISTFILE distfilename]

DESCRIPTION
DUPDATE is a program that creates distributions.  It creates an exact
duplicate of the source directory tree in the destination with
modifications according to control files in the tree.  DUPDATE
deletes files in the destination tree that do not exist in the source
and updates files from the source into the destination tree that have
been modified since the last dupdate (or copies them fresh if they do
not exist).

 FORCE  DUPDATE will not ask permission to copy a fresh file

 QUIET  DUPDATE will not display verbose output

 NODEL  DUPDATE will not delete files in the destination that do not
    exist in the source.

 DISTFILE file
 Specify alternate control file that 'modifies' the dist update,
 default is .DistFiles

 If a file ".DistFiles" exists in any directory of the source
 tree, updating of the destination is modified according to the
 file.  This is a text file which may specify additional
 files/directories to add to the destination directory (pulled
 from other random places), files and directories NOT to include
 in the destination tree, or a list of specific files to include
 (where files not listed are not included).
```

```
By using the DISTFILE file option you can generate different
distributions for different purposes all based in the same source
tree.  For example, I have a DISTFILE set to create the
registered and non-registered DICE distributions and other
DISTFILE files (using different names) to create the three
floppies in the registered distribution.

In the first format, if the ONLY keyword is specified after the
first file name only these files / sub-directories will be
included from this directory.  No other files will be copied

file_or_dir_name ONLY
file_or_dir_name
file_or_dir_name
file_or_dir_name
file_or_dir_name

The second format allows files/directories to be made part of the
destination tree that do not necessarily exist in the current
directory. Additionally, specific files/directories that do exist
in the current directory can be excluded.  Any file/dir not
explicitly unincluded using the 'no' keyword will be copied.

file_or_dir_path
file_or_dir_path
file_or_dir_path
no file_or_dir_path
no file_or_dir_path
file_or_dir_path
```

1.21 expand Commands

```
FUNCTION
expand wildcards

SYNOPSIS
expand [format] wildcards

DESCRIPTION
Expand functions like the AmigaDOS 2.0 "LIST LFORMAT" command, Expand
generates a list of files, one per line, using the specified format
string (the default is "%s").

1 expand "type %s" #?

The above would create one "type" command for every file in the
current directory.
```

1.22 fdtolib Commands

```
FUNCTION
Create Link Libraries from .FD files

SYNOPSIS
FDTOLIB files/wildcard.fd [-h hdrfile] -o libname [-mr] [-mD]

DESCRIPTION
FDTOLIB will create an amiga standard link library out of specified
.FD files (for example, you can generate most of amiga.lib by using
the .FD files on your 1.3 Extras disk).  .FD files are a standard
format file that describe the function names and offsets of shared
Amiga (Exec) libraries.  See section for a description of the format.
fdtolib creates the interface stubs and the AutoInit code used by
DICE to automatically open and close the library.

Basically, FDTOLIB will generate one of four types of libraries:

 Option  : Library Type
 ========+=============================================================
 default : small-data model
 --------+-------------------------------------------------------------
 -mD     : large-data model
 --------+-------------------------------------------------------------
 -mr     : small-data model + DICE registered parameters entry pts
 --------+-------------------------------------------------------------
 -mr -mD : large-data model + DICE registered parameters entry pts
 --------+-------------------------------------------------------------

If -mr is used suitable prototypes must be specified with the -h
option. In this case, FDTOLIB will run DCC with a special option to
have it generate a register-specification file for it to match up
again the .FD files.

FDTOLIB then proceeds to scan the .FD files, creating temporary
assembly files in T: and assembling them with DAS, then appending
them to

the output library and deleting the scratch files.  This step occurs
for each function in each .FD files.

(For faster operation, you will want to make DAS resident for the
duration)

If -mr was specified, FDTOLIB only generates library entries for
those routines for which a prototype exists.  At the end of the run
FDTOLIB will report any routines which existed in the .FD files but
did not have a prototype.

files/wildcard.fd specifies one or more files and/or AmigaDOS
wildcarding that represents the .FD files that are to be processed
into a library

-h hdrfile
    hdrfile is a .H files that #include's all prototypes associated
    with the .FD files.  It is only used if the -mr option is
    specified
```

```
-o libname
    specify output library name

   -mr  specify that a REGISTERED call interface library is to be
    generated (for DICE -m[r,R,RR] options), else generates a normal
    stack-args interface library.

   -mD  specify large-data model, else small-data model

-I include-dir
    passed to DCC

-p prefix  Set prefix (currently only for standard generation, doesn't work
    with -mr).  The default is a single underscore _.

 -prof  Generate profiling code for the tags.  This will cause all
    library calls to be profiled when the program that links with
    this library is run.

-auto library
    Generate auto-init code for library after the tags.  library is
    the name of the shared library.  For example, -auto fubar.library

-AUTO library
    Generate ONLY auto-init code for library (do not generate tags)

SEE ALSO
fdtopraga for a description of .fd file format.
```

1.23 fdtopragma Commands

```
FUNCTION
Create #pragma statements from .FD files

SYNOPSIS
fdtopragma source/ [-o dest]

fdtopragma
This program generates header files containing #pragma libcall lines
for use with inline library calls.  #pragma is an ANSI C mechanism to
allow compiler extensions.  #pragma libcall is an Amiga standard for
describing shared library entries, allowing the compiler to generate
calls directly, rather than requiring linking in cumbersome interface
functions.  .FD files are a standard format file that describe the
function names and offsets of shared runtime libraries. See section
for a description of the format.

The source may be specified as a single file, or as a directory (with
trailing /).  Under DICE, these header files are stored in
DINCLUDE:CLIB/ and mimic the Commodore standard headers in
DINCLUDE:AMIGA20/CLIB/.  The purpose of the mimicry is to have a
single standard for specification of prototypes, the Commodore
standard:
```

```
#include <clib/exec_protos.h>
```

If no pragma header exists, the Commodore standard header will be included. If a pragma header does exist, it will be included. Rhe pragma header file explicitly #include's the original Commodore header file and then conditionally generate the #pragma lines based on whether you specified the -mi option to DCC. The -mi option to DCC simply defines the preprocessor symbol __DICE_INLINE which causes the #pragma lines to be conditionally included. It is important to note that great pains have been taken to allow you to turn on and off inline library calls for your program without having to modify the source in any way, shape, or form.

.fd files are formatted very simply. For the Commodore libraries, these files are stored in dinclude:amigaxx/fd/". Comments start with "*", commands start with "##", and everything else is assumed to be a function entry. The commands are:

Command	Usage		
		base	Base pointer name for this library
		_DOSBase	(DOSBase)
		bias 30	New negative function offset
			(Negative 30)
		public	Until next
		private	Until next
		end	End marker

The following function entry defines four parameters, which must be passed in registers A0,D0,A1 and D1 respectively. A "/" separator is a hint to some programs that registers are in proper order to move from the stack with a single 68000 "MOVEM" instruction:

```
OpenDevice(devName,unit,ioRequest,flags)(a0,d0/a1,d1)
```

Library functions exist as negative offsets from the library base; ##bias sets a new negative offset. Each function entry decrements the offset by six.

1.24 flush Commands

```
FUNCTION
Flush Memory, Libraries, and Devices

SYNOPSIS
flush

DESCRIPTION
Flush causes a "memory panic," forcing all currently unused fonts,
```

```
libraries, etc. to be removed from memory.  This is useful to free up
large chunks of memory or force an old version of a library out of
memory in order to test a new one.


@ENDNODE
```

1.25 head Commands

```
FUNCTION
Display start of a file

SYNOPSIS
head file

DESCRIPTION
Head prints the first ten lines of the specified file.
```

1.26 ident Commands

```
FUNCTION
Identify Files

SYNOPSIS
ident [ -q ] [ file ...  ]

DESCRIPTION
Ident searches the named files or, if no file name appears, the
standard input for all occurrences of the pattern $keyword:...$,
where keyword is one of Author, Date, Header, Id, Locker, Log,
Revision, RCSfile, Source, or State.  This command works much like
the AmigaDOS version command.

These patterns are normally inserted automatically by the RCS command
co, but can also be inserted manually.  The option -q suppresses the
warning given if there are no patterns in a file.

Ident works on text files as well as object files and dumps.  For
example, if the C program in file f.c contains

char rcsid[] = "$Header: /home/dice/com/master/Doc/RCS/dice_commands.doc,v
 30.8 1994/08/18 05:39:56 dice Exp dice $";

and f.c is compiled into f.o, then the command will print:

1> ident f.c f.o
f.c: $Header: /home/dice/com/master/Doc/RCS/dice_commands.doc,v 30.8 1994/08/18  ↩
   05:39:56 dice Exp dice $
f.o: $Header: /home/dice/com/master/Doc/RCS/dice_commands.doc,v 30.8 1994/08/18  ↩
   05:39:56 dice Exp dice $

SEE ALSO
```

```
ci, co, rcs, rcsdiff, rcsintro, rcsmerge, rlog
```

1.27 istrip Commands

```
FUNCTION
Strip Comments From Include Files

SYNOPSIS
ISTRIP destprefix wildcards

DESCRIPTION
ISTRIP will strip comments and extraneous whitespace from all files
specified by wildcards and create an output file under the same name
prefixed by destprefix. ISTRIP preserves the copyright notice, and
replaces comments with blank lines to avoid changing any line
numbering.

ISTRIP is very stupid in that it will not create the destination
directory hierarchy.  The COPY command in the example below basically
does that for us, the copied files are extraneous and overwritten
when ISTRIP is run.

ISTRIP is useful mainly for developers who obtain later versions of
the commented Amiga includes and want to create an uncommented
version (The uncommented includes are much smaller, yielding faster
compilation).

EXAMPLE
1> copy dinclude:amiga13  ram:amiga13  ALL  QUIET
1> cd dinclude:
1> istrip  ram:  amiga13/#?/#?
```

1.28 libmake Commands

```
FUNCTION
Create Link Library

SYNOPSIS
libmake file options

DESCRIPTION
Libmake is a utility that will scan a file listings sources files for
a library, determine what is out of date, compile the out of date
modules (compile .c modules, assemble .a modules), and JOIN the whole
thing together in the end to create a library.  Libmake is useful for
creating large libraries that would otherwise overflow the command
line length limit in DMakefile.

Libmake takes several arguments, some optional:

  file  specify the control file that contains a list of source modules,
```

```
    see below.

-v  verbose operation

-n  dry run (do not actually compile/assemble/join anything)

-Dmacro[=def]
    specify DCPP macro, i.e.  #define equivalent to be passed to all
    compiles.

-o object_dir
    specify object directory prefix, if a directory must end in '/'
    or ':', allowing both file prefixes and directory paths.

-l library
    specify library output file, usually something.lib

-clean  instead of compiling/assembling/join'ing the library, delete ALL
    object modules from object_dir relating to the library.

   -pr  pass -pr option to DCC

-proto  pass -proto option to DCC

  -mRR  specify reg-call opts to DCC.

   -mD  pass -mD to DCC, causes DCC to use the large-data model.  Default
    is to use the small-data model

   -mC  pass -mC to DCC, causes DCC to use the large-code model.  Default
    is to use the small-data model

 -prof  pass -prof to DCC, causes profiling code to be generated for all
    the routines in the library.

CONTROL FILE
The control file is named files.something by convention, for example,
'files.c3lib', which happens to be the control file used generate
C*.LIB.

A control file may contain blank lines, lines that begin with a
semi-colon (comments), and lines containing a file name optionally
preceded by a '*'.  Here is an example:

; Full C library
assert/assert.c
assert/abort.c
amiga/exit.c
amiga/main.c
amiga/wbmain.c
*amiga/c.a
*amiga/c_pi.a
*amiga/c_pr.a
*amiga/x.a
amiga/config.a

Lines beginning with a '*' tell LIBMAKE to compile/assemble the file
```

```
but NOT to include the object module in the generated output library.

Thus, in the above example amiga/c.a would be assembled but not made
part of the DLIB:C.LIB

Also note that the path specified for a given file is appended to the
-o (object directory) specification.  Thus, if you were to use the
following libmake line:

1> libmake files.c3lib -o dtmp:xx/ -l dlib:xx.lib -pr -proto

Then object modules would be created as follows:

DTMP:XX/assert/assert.o
DTMP:XX/assert/abort.o
DTMP:XX/amiga/exit.o
etc..

You probably want to pre-create the directory structure required.
Please refer to the library source archive for examples (no less than
DMakefile's calling libmake to regenerate every single DICE library
that exists!)

NAMING CONVENTIONS
In order to simplify the process, libmake makes assumptions about the
type of file based on the extension.

Extension : Libmake Action
==========+=================================================================
.a        : Assemble with DAS
----------+-----------------------------------------------------------------
.a68      : Assemble with external assembler A68K
----------+-----------------------------------------------------------------
.o        : Insert specified object into destination library (raw
          : copy)
----------+-----------------------------------------------------------------
.lib      : Insert specified library into destination library (raw
          : copy)
----------+-----------------------------------------------------------------
other     : Assumed to be a C source file to compile with DCC
----------+-----------------------------------------------------------------
```

1.29 libtos Commands

```
FUNCTION
Library Converter

SYNOPSIS
LIBTOS source dest

DESCRIPTION
This program converts the Commodore supplied amiga.lib from large
data model to small data model.  You must convert amiga.lib before
you can use it with the DICE system to generate residentable
programs.
```

```
Note that this isn't required, but a small-data amiga.lib will
generate faster code with fewer reloc32's (A reloc32 is a 32-bit
relocation, it uses up space in an executable and takes extra time to
load).

The small-data-model version of amiga.lib is called amigas.lib

1> cd DLIB:
1> LIBTOS amiga.lib amigas.lib
```

1.30 loadabs Commands

```
FUNCTION
Absolute Locator

SYNOPSIS
LoadAbs exefile -o outfile -A addr

DESCRIPTION
LoadAbs takes a standard Amiga executable and generates an image file
relocated to the absolute location specified.  The image file is
structured in the same order as the hunks appear in the Amiga
executable.  BSS hunks will generate 0's in the image file.

exefile  Executable to do the absolute relocation on

-O outfile
    Resulting image file

-A addr  0xHEX absolute relocation address

    || NOTE: This program will do 32 bit relocations only.  Generally
    || you only use LoadAbs with -mD -mC compiled programs.
```

1.31 loadfile Commands

```
FUNCTION
Load & Hold a File in Memory

SYNOPSIS
LoadFile filename

DESCRIPTION
Loadfile is very simple.  It just loads a binary file into memory,
and holds it there until CTRL-C is pressed. This lets you examine the
file with a debugger, or Metascope or some such tool.  Useful for ROM
work.
```

1.32 makeindex Commands

```
FUNCTION
Build Index File for Online Help System

SYNOPSIS
MakeIndex outfile pattern

DESCRIPTION
MakeIndex builds a lookup file for the DICEHelp online help system.
This file is normally called "s:dicehelp.index". Entries are always
appended to outfile to allow building the index file in steps.  Any
number of files may be specified with wildcards in pattern. MakeIndex
detects DICE documentation, AutoDoc files from Commodore, C include
files and Assembler include files. Documentation files are indexed by
the name of the function.  C include files are indexed by structures.
The names of assembler includes are recorded, but no additional
processing is done.

MakeIndex is normally run during the installation, or later by
selecting the installer option "refresh DICEHelp index file."  You
may append your own selections using MakeIndex.

SEE ALSO
Chapter , Online help.
```

1.33 makeproto Commands

```
FUNCTION
Easily Create Prototype File

SYNOPSIS
makeproto infile outfile

DESCRIPTION
Collects lines beginning with the word Prototype from all your source
files into a single header file.  Each source module in a project
normally includes a common header file, DEFS.H, which contains items
common to the project.  The idea is to add the following to your
DEFS.H file:

#define Prototype extern
#define Local static /* or as nothing at all */
#include "protos.h"   /* prototype file generated by MAKEPROTO

Each source would contain prototypes that look like this (shown with
example declarations):

Prototype int FuGlob;
Prototype void FuBar(int);
Prototype struct MyFu *FuBar2(short);

int FuGlob; /* etc... */
```

```
void FuBar(int x) {
...
}
```

You then create a PROTOS.H file by running MAKEPROTO on all source files. Among the tricks that are possible is the use of structure tags instead of typedefs in the prototypes themselves, allowing the prototype file to be #include'd during the normal course of compilation without necessarily requiring precursor includes to guarantee the validity of the types you use. Since a declaration containing a pointer to an undefined structure is valid as long as you do not try to access specific elements in the structure, this allows you to bring in prototypes for all functions in your entire project whether you use them in any specific source module or not.

MAKEPROTO has one additional feature which makes its usage all the more efficient... if the specified output file already exists MAKEPROTO will compare its output with the existing file and not modify the date stamp of the file unless the output diffs. This is especially useful when you use precompiled includes where you might want to include a dependency to force the precompiled include to be recomputed if any header file OR the prototype file changes. Without this feature you would have to force the precompiled include to be recomputed every time you modify a source file because you would not be able to determine whether that modification resulted in a change in the prototype file PROTOS.H or not.

1.34 merge Commands

```
FUNCTION
Three-Way File Merge

SYNOPSIS
merge

DESCRIPTION
Merge is used by rcsmerge to do three way file merges - integrating
changes from several revisions into a single complete file.

SEE ALSO
rcsmerge
```

1.35 rcs Commands

```
FUNCTION
Change RCS File Attributes

SYNOPSIS
rcs [ options ] file ...

DESCRIPTION
Rcs creates new RCS files or changes attributes of existing ones. An
```

```
RCS file contains multiple revisions of text, an access list, a
change log, descriptive text, and some control attributes.  For rcs
to work, the caller's login name must be on the access list, except
if the access list is empty, the caller is the owner of the file or
the superuser, or the -i option is present.

Files ending in ,v are RCS files, all others are working files.  If a
working file is given, rcs tries to find the corresponding RCS file
first in directory ./RCS and then in the current directory, as
explained in co.

-i  creates and initializes a new RCS file, but does not deposit any
    revision.  If the RCS file has no path prefix, rcs tries to place
    it first into the subdirectory ./RCS, and then into the current
    directory.  If the RCS file already exists, an error message is
    printed.

-alogins  appends the login names appearing in the comma-separated list
    logins to the access list of the RCS file.

-Aoldfile  appends the access list of oldfile to the access list of the RCS
    file.

-e[logins]
    erases the login names appearing in the comma-separated list
    logins from the access list of the RCS file.  If logins is
    omitted, the entire access list is erased.

-b[rev]  sets the default branch to rev.  If rev is omitted, the default
    branch is reset to the (dynamically) highest branch on the trunk.

-cstring  sets the comment leader to string.  The comment leader is printed
    before every log message line generated by the keyword $Log:
    dice_commands.doc,v $

# Revision 30.8  1994/08/18  05:39:56  dice
# .
#
# Revision 30.0  1994/06/10  17:57:04  dice
# .
#
# Revision 30.0  1994/06/10  17:57:04  dice
# .
#
    during checkout (see co).  This is useful for programming
    languages without multi-line comments.  During rcs -i or initial
    ci, the comment leader is guessed from the suffix of the working
    file.

-l[rev]  locks the revision with number rev.  If a branch is given, the
    latest revision on that branch is locked.  If rev is omitted, the
    latest revision on the default branch is locked.  Locking
    prevents overlapping changes.  A lock is removed with ci or rcs
    -u (see below).

-u[rev]  unlocks the revision with number rev.  If a branch is given, the
    latest revision on that branch is unlocked.  If rev is omitted,
```

```
    the latest lock held by the caller is removed.  Normally, only
    the locker of a revision may unlock it.  Somebody else unlocking
    a revision breaks the lock.

-L  Sets locking to strict.  Strict locking means that the owner of
    an RCS file is not exempt from locking for checkin.  This option
    should be used for files that are shared.

-U  Sets locking to non-strict.  Non-strict locking means that the
    owner of a file need not lock a revision for checkin.  This
    option should NOT be used for files that are shared.

-nname[:rev]
    Associates the symbolic name name with the branch or revision
    rev. Rcs prints an error message if name is already associated
    with another number. If rev is omitted, the symbolic name is
    deleted.

-Nname[:rev]
    Same as -n, except that it overrides a previous assignment of
    name.

-orange  Deletes ("outdates") the revisions given by range.  A range
    consisting of a single revision number means that revision.  A
    range consisting of a branch number means the latest revision on
    that branch. A range of the form rev1-rev2 means revisions rev1
    to rev2 on the same branch, -rev  means from the beginning of the
    branch containing rev up to and including rev, and rev means from
    revision rev to the end of the branch containing rev.  None of
    the outdated revisions may have branches or locks.

-q  Quiet mode; diagnostics are not printed.

-sstate[:rev]
    sets the state attribute of the revision rev to state.  If rev is
    a branch number, the latest revision on that branch is assumed.
    If rev is omitted, the latest revision on the default branch is
    assumed.  Any identifier is acceptable for state.  A useful set
    of states is Exp (for experimental), Stab (for stable), and Rel
    (for released).  By default, ci sets the state of a revision to
    Exp.

-t[txtfile]
    writes descriptive text into the RCS file (deletes the existing
    text).  If txtfile is omitted, rcs prompts the user for text
    supplied from the standard input, terminated with a line
    containing a single . or CTRL-\.  Otherwise, the descriptive text
    is copied from the file txtfile.  If the -i option is present,
    descriptive text is requested even if -t is not given.  The
    prompt is suppressed if the standard input is not a terminal.

DIAGNOSTICS
The RCS file name and the revisions outdated are written to the
diagnostic output.  The exit status always refers to the last RCS
file operated upon, and is 0 if the operation was successful, 1
otherwise.
```

```
FILES
rcs creates a semaphore file in the same directory as the RCS file to
prevent simultaneous update.  For changes, rcs always creates a new
file.  On successful completion, rcs deletes the old one and renames
the new one.

SEE ALSO
co, ci, ident, rcsdiff, rcsintro, rcsmerge, rlog
```

1.36 rcsclean Commands

```
FUNCTION
Clean up RCS Work Files

SYNOPSIS
rcsclean [ -rrev ] [ -qrev ] file...

DESCRIPTION
Rcsclean removes working files that were checked out and never
modified.  For each file given, rcsclean compares the working file
and a revision in the corresponding RCS file.  If it finds no
difference, it removes the working file, and, if the revision was
locked by the caller, unlocks the revision.

A file name ending in ',v' is an RCS file name, otherwise a working
file name.  Rcsclean derives the working file name from the RCS file
name and vice versa, as explained in co.  Pairs consisting of both an
RCS and a working file name may also be specified.

-r Rev  specifies with which revision the working file is compared.  If
    rev is omitted, rcsclean compares the working file with the
    latest revision on the default branch (normally the highest
    branch on the trunk).

-q  suppresses diagnostics.

    Rcsclean is useful for "clean" targets in Makefiles.  Note that
    rcsdiff prints out the differences.  Also, ci normally asks
    whether to check in a file if it was not changed.

EXAMPLES
rcsclean *.c *.h

The above command removes all working files ending in ".c" or ".h"
that were not changed since their checkout.

DIAGNOSTICS
The exit status is 0 if there were no differences during the last
comparison or if the last working file did not exist, 1 if there were
differences, and 2 if there were errors.

SEE ALSO
co, ci, ident, rcs, rcsdiff, rcsintro, rcsmerge, rlog
```

1.37 rcsdiff Commands

```
FUNCTION
Compare RCS Revisions

SYNOPSIS
rcsdiff [ -biwt ] [ -cefhn ] [ -q ] [ -rrev1 ] [ -rrev2 ] file ...

DESCRIPTION
Rcsdiff runs diff to compare two revisions of each RCS file given.  A
file name ending in ',v' is an RCS file name, otherwise a working
file name.  Rcsdiff derives the working file name from the RCS file
name and vice versa, as explained in co.  Pairs consisting of both an
RCS and a working file name may also be specified.

The options -b, -i, -w, -t, -c, -e, -f, and -h, have the same effect
as described in diff.

-n  generates an edit script of the format used by RCS

-q  Suppresses diagnostic output.

    If both rev1 and rev2 are omitted, rcsdiff compares the latest
    revision on the default branch (normally the highest branch on
    the trunk) with the contents of the corresponding working file.
    This is useful for determining what you changed since the last
    checkin.

    If rev1 is given, but rev2 is omitted, rcsdiff compares revision
    rev1 of the RCS file with the contents of the corresponding
    working file.

    If both rev1 and rev2 are given, rcsdiff compares revisions rev1
    and rev2 of the RCS file.

    Both rev1 and rev2 may be given numerically or symbolically, and
    may actually be attached to any of the options.

EXAMPLES
rcsdiff f.c

The above command runs diff, comparing the currently checked out
version with the latest revision stored on the current trunk.

DIAGNOSTICS
The exit status is 0 if there were no differences during the last
comparison, 1 if there were differences, and 2 if there were errors.

SEE ALSO
ci, co, diff, ident, rcs, rcsintro, rcsmerge, rlog
```

1.38 rcsmerge Commands

```
FUNCTION
```

Merge RCS Revisions

SYNOPSIS
rcsmerge -rrev1 [-rrev2] [-p] file

DESCRIPTION
Rcsmerge incorporates the changes between rev1 and rev2 of an RCS file into the corresponding working file. If -p is given, the result is printed on the standard output, otherwise the result overwrites the working file.

A file name ending in ',v' is an RCS file name, otherwise a working file name. Rcsmerge derives the working file name from the RCS file name and vice versa, as explained in co. A pair consisting of both an RCS and a working file name may also be specified.

Rev1 may not be omitted. If rev2 is omitted, the latest revision on the default branch (normally the highest branch on the trunk) is assumed. Both rev1 and rev2 may be given numerically or symbolically.

Rcsmerge prints a warning if there are overlaps, and delimits the overlapping regions as explained in co -j. The command is useful for incorporating changes into a checked-out revision.

EXAMPLES
Suppose you have released revision 2.8 of f.c. Assume furthermore that you just completed revision 3.4, when you receive updates to release 2.8 from someone else. To combine the updates to 2.8 and your changes between 2.8 and 3.4, put the updates to 2.8 into file f.c and execute

rcsmerge -p -r2.8 -r3.4 f.c >f.merged.c

Then examine f.merged.c. Alternatively, if you want to save the updates to 2.8 in the RCS file, check them in as revision 2.8.1.1 and execute co -j:

ci -r2.8.1.1 f.c
co -r3.4 -j2.8:2.8.1.1 f.c

As another example, the following command undoes the changes between revision 2.4 and 2.8 in your currently checked out revision in f.c.

rcsmerge -r2.8 -r2.4 f.c

Note the order of the arguments, and that f.c will be overwritten.

SEE ALSO
ci, co, merge, ident, rcs, rcsdiff, rlog

BUGS
Rcsmerge does not work on files that contain lines with a single ..

1.39 rlog Commands

```
FUNCTION
Display RCS History

SYNOPSIS
rlog [ options ] file ...

DESCRIPTION
Rlog prints information about RCS files.  Files ending in ,v are RCS
files, all others are working files.  If a working file is given,
rlog will locate the corresponding RCS file.

Rlog prints the following information for each RCS file: RCS file
name, working file name, head (i.e., the number of the latest
revision on the trunk), default branch, access list, locks, symbolic
names, suffix, total number of revisions, number of revisions
selected for printing, and descriptive text.  This is followed by
entries for the selected revisions in reverse chronological order for
each branch.  For each revision, rlog prints revision number, author,
date/time, state, number of lines added/deleted (with respect to the
previous revision), locker of the revision (if any), and log message.
Without options, rlog prints complete information.  The options below
restrict this output.

-L  ignores RCS files that have no locks set; convenient in
    combination with -R, -h, or -l.

-R  only prints the name of the RCS file; convenient for translating
    a working file name into an RCS file name.

-h  prints only RCS file name, working file name, head, default
    branch, access list, locks, symbolic names, and suffix.

-t  prints the same as -h, plus the descriptive text.

-b  prints information about the revisions on the default branch
    (normally the highest branch on the trunk).

-ddates  prints information about revisions with a checkin date/time in
    the ranges given by the semicolon- separated list of dates.  A
    range of the form d1<d2 or d2>d1 selects the revisions that were
    deposited between d1 and d2, (inclusive).  A range of the form <d
    or d> selects all revisions dated d or earlier.  A range of the
    form d< or >d selects all revisions dated d or later.  A range of
    the form d selects the single, latest revision dated d or
    earlier.  The date/time strings d, d1, and d2 are in the free
    format explained in co.  Quoting is sometimes necessary,
    especially for < and >.  Note that the separator is a semicolon.

-l[lockers]
    prints information about locked revisions.  If the
    comma-separated list lockers of login names is given, only the
    revisions locked by the given login names are printed.  If the
    list is omitted, all locked revisions are printed.
```

```
-rrevisions
    prints information about revisions given in the comma-separated
    list revisions of revisions and ranges.  A range rev1-rev2 means
    revisions rev1 to rev2 on the same branch, -rev means revisions
    from the beginning of the branch up to and including rev, and
    rev- means revisions starting with rev to the end of the branch
    containing rev.  An argument that is a branch means all revisions
    on that branch.  A range of branches means all revisions on the
    branches in that range.

-sstates  prints information about revisions whose state attributes match
    one of the states given in the comma-separated list states.

-w[logins]
    prints information about revisions checked in by users with login
    names appearing in the comma- separated list logins.  If logins
    is omitted, the user's login is assumed.

    Rlog prints the intersection of the revisions selected with the
    options -d, -l, -s, -w, intersected with the union of the
    revisions selected by -b and -r.

EXAMPLES
rlog -L -R RCS/*,v
rlog -L -h RCS/*,v
rlog -L -l RCS/*,v
rlog RCS/*,v

The first command prints the names of all RCS files in the
subdirectory RCS which have locks.  The second command prints the
headers of those files, and the third prints the headers plus the log
messages of the locked revisions.  The last command prints complete
information.

DIAGNOSTICS
The exit status always refers to the last RCS file operated upon, and
is 0 if the operation was successful, 1 otherwise.

SEE ALSO
ci, co, ident, rcs, rcsdiff, rcsintro, rcsmerge, section
```

1.40 romable Commands

```
FUNCTION
Generate Romable Image

SYNOPSIS
Romable exeFile -o outFile [-o out2] -C addr -D addr  -pi

DESCRIPTION
Romable takes an executable compiled by DICE, and generates a binary
image.  This is normally used to generate a file for programming into
ROM.

exeFile  input executable linked with dlink
```

```
-o outFile
    output binary (unformatted -- raw).  If TWO -o options are
    specified the two output files will have even bytes and odd bytes
    respectively, which is what you need when you must program two
    eproms (one on the LSB data lines and one on the MSB data lines).

-C addr  code start address, 0octal, decimal, or 0xHEX

-D addr  data start address, 0octal, decimal, or 0xHEX

   -DC  place actual data+bss just after code (i.e.  the result is
    intended to be downloaded into RAM, there is no duplicate data in
    this case). '-D addr' is not specified in this case

   -pi  generate a position independent module.  Neither -C or -D are
    specified in this case, and Romable will warn you have any
    absolute references.

    Note that your custom startup code determines how much of
    __autoinit and __autoexit is to be supported.  Note especially
    that __autoinit0 MUST BE SUPPORTED because DICE will generate
    __autoinit0 sections to handle 32 bit data relocations run-time.

    Romable generates a raw output file or files with the EPROM code
    first, and initialized data after the main code (still in EPROM)
    which, as has already been described, will be copied to RAM on
    reset by your startup routine.

    This startup-copying of initialized data and clearing of BSS
    makes it extremely easy to use DICE to generate ROMED
    applications without having to deal with major porting
    considerations.
```

1.41 touch Commands

```
FUNCTION
Update File Datestamp

SYNOPSIS
touch file

DESCRIPTION
Touch bumps the date of a file without changing the contents.  This
is useful to force utilities like VMake and DMake to recompile source
files.
```

1.42 ttxsame Commands

```
FUNCTION
Helper Program for Integrated Error Scripts
```

DESCRIPTION
ttxsame is a helper program used by the integrated error scripts to
start the TurboText editor.

1.43 vmake Commands

FUNCTION
Visual Interface to DICE

DESCRIPTION
VMake is a complete control center for DICE. From within VMake you
can manage a project, check files in or out of RCS, select files to
edit, and finally compile and run your program. VMake is an
alternate to the CLI-based dcc program. VMake is very flexible, and
can be configured to control programs other than DICE. See chapter
for a full description.

1.44 vopts Commands

FUNCTION
Visual Interface for Setting Options

DESCRIPTION
VOpts provides an easy and powerful visual interface for selecting
compiler options. Options may be specified explicitly, or simply
left to defaults. VOpts is very flexible, can can be configured to
set options for any program, not just DICE. See chapter for all the
details.

1.45 wbrun Commands

FUNCTION
Simulate Starting a Program From Workbench

SYNOPSIS
wbrun file

DESCRIPTION
Wbrun is used by VMake to simulate the method used by the Amiga
Workbench to start programs.

1.46 wc Commands

FUNCTION
Count Elements in a File

SYNOPSIS

```
wc file ...

DESCRIPTION
wc counts the number of characters, words, and lines in each
specified file and prints a total at the end.

This page is not blank.
```