fd2pragma

COLLABORATORS

	TITLE : fd2pragma	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 11, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

fd2pragma

1.1 fd2pragma - the programmers file generator

Calling the program seems to be (is) very difficult, but it offers you a large set of functions. A lot of options need a lot of abilities to turn them on/off! The program has an internal help, which can be accessed by the standard method: First call "fd2pragma ?" to get the synopsis and afterwards enter ? again to get the argument description. Nobody can remember all the SPECIAL options without that!

Read this documentation very carefully, because there are some notes you may not see on fast reading, but which will help you a lot. (for example HEADER option and "" filename)

1.2 about

This is a utility to create:
- the following pragma statements for certain C compilers: amicall, libcall, flibcall, tagcall, and syscall
- proto files for C compilers
- offset (LVO) files for assembler programs
- stub functions for either tag-functions or all library functions
- stub functions as assembler text
- stub functions as usable link library file
- FD files out of pragma files
- stubs for C++ compilers (SPECIAL 11, 12, and CLIB)
- the files with your own headers inserted
- files for using local pointers for shared library bases in compilers which do not normally support this
- stub functions for Pascal compilers
- inline files for GCC
- inline, pragma, and stub libraries using FPU registers
- files needed to develop for Phase5's PowerUP boards
- unit texts for FPC Pascal compiler
- BMAP files for AmigaBASIC and MaxonBASIC
- VBCC assembler inline files
- VBCC WOS stub texts and link library
- VBCC PowerUP stub texts and link library
- VBCC MorphOS stub texts and link library
- Modules for AmigaE
- FD files
- CLIB files
- SFD files
- auto library open files for VBCC

Therefor only the FD file giving the library information is needed. For some SPECIAL options you may additionally supply the CLIB keyword (or you need to supply it) giving fd2pragma the prototypes file in clib directory. Better is to supply the all-in-one SFD format as input.

Special option 200 does the opposite of the usual: it converts pragma to FD!

Please tell me if something is missing. I do not know all the possibilities
of different compiler systems that I do not have, so maybe I disabled some
stuff (e.g. entries for PPC), which could be produced normally.

Commodore had a special program called sfd, which created FD files, pragmas,
stub libs, clib includes, and fd files out of a special sfd file format.
Because this format was never released to the public we need to use a
program like fd2pragma which uses FD and clib files as input. It is able to
create all needed files as well. As fd2pragma is a lot newer than the
Commodore tool it can do some more stuff (inline files, Storm files, VBCC
files, GCC files, link libraries, C++ names). As I got finally to rights to
include and release the SFD format, fd2pragma now also supports this
definition files. Thanks a lot to Olaf Barthel for making this possible.

I suggest using SFD instead of FD in the future, as it is much better
usable and covers all required information in one file. And it means you
files are equal to the system development files.

1.3 docs

This documentation is an collection of nearly all the necessary information
about development files related to library creation.

It describes the program fd2pragma, its options, the resulting files, the
source files, and all the data formats. It also covers a lot of tips and
suggestions for designing your own libraries and for designing the
development material.

If something is missing or unclear, contact me and I will fix the stuff.

Greetings go to Gary Goldberg for reading the whole document and fixing lots
of errors.

1.4 beginner

To make using fd2pragma much easier I collected some example calls, which
maybe useful when suing fd2pragma seldom.

All the example are based on dos.library files. Replace intuition with
the library you want to create files for. Also use the directories which
are used on you system.

If you have the newer SFD files installed, replace all
"fd/dos_lib.fd CLIB clib/dos_protos.h" with "sfd/dos_lib.sfd".

If these examples do not fit your needs try different numbers for SPECIAL
or start using the additional options.

****
Users of MaxonC++ or StormC++ up to Version V3:

1) Create the correct pragma files:

```
   fd2pragma fd/dos_lib.fd CLIB clib/dos_protos.h TO pragma SPECIAL 6

2) Create the correct proto files:
   fd2pragma fd/dos_lib.fd CLIB clib/dos_protos.h TO proto SPECIAL 35

********************
Users of SAS-C

1) Create the correct pragma files:
   fd2pragma fd/dos_lib.fd CLIB clib/dos_protos.h TO pragma SPECIAL 6

2) Create the correct proto files:
   fd2pragma fd/dos_lib.fd CLIB clib/dos_protos.h TO proto SPECIAL 35

********************
Users of Storm V4

1) Create the correct inline files:
   fd2pragma fd/dos_lib.fd CLIB clib/dos_protos.h TO inline SPECIAL 43

2) Create the correct proto files:
   fd2pragma fd/dos_lib.fd CLIB clib/dos_protos.h TO proto SPECIAL 39

********************
Users of GCC

1) Create the correct inline files:
   fd2pragma fd/dos_lib.fd CLIB clib/dos_protos.h TO inline SPECIAL 40

2) Create the correct proto files:
   fd2pragma fd/dos_lib.fd CLIB clib/dos_protos.h TO proto SPECIAL 35

********************
Users of VBCC (when using inlines)

1) Create the correct inline files:
   fd2pragma fd/dos_lib.fd CLIB clib/dos_protos.h TO inline SPECIAL 70

2) Create the correct proto files:
   fd2pragma fd/dos_lib.fd CLIB clib/dos_protos.h TO proto SPECIAL 38

********************
All systems

1) Create pragmas to proto redirect:
   fd2pragma fd/dos_lib.fd CLIB clib/dos_protos.h TO pragmas SPECIAL 80
```

1.5 options

```
You get the command template with fd2pragma ? .

FROM=INFILE/A,SPECIAL/N,MODE/N,TO/K,ABI/K,CLIB/K,COPYRIGHT/K,HEADER/K,
HUNKNAME/K,BASENAME/K,LIBTYPE/K,LIBNAME/K,PRIORITY/N/K,COMMENT/S,
EXTERNC/S,FPUONLY/S,NEWSYNTAX/S,NOFPU/S,NOPPC/S,NOPPCREGNAME/S,
NOSYMBOL/S,ONLYCNAMES/S,OPT040/S,PPCONLY/S,PRIVATE/S,SECTION/S,
```

```
SMALLCODE/S,SMALLDATA/S,SORTED/S,SYSTEMRELEASE/S,USESYSCALL/S:

In this position you may press <?> again and you get the following text!
Remember that! I myself need this whenever I call the program, as nobody
can remember all these options.
Be careful, because this text is longer than one normal high-resolution
screen, so it is useful to press a key in the middle of the text to stop
the output.

INFILE:  the input file which should be used
SPECIAL: 1 - Aztec compiler (xxx_lib.h,     MODE 2, AMICALL)
   2 - DICE compiler  (xxx_pragmas.h, MODE 3, LIBCALL)
   3 - SAS compiler   (xxx_pragmas.h, MODE 3, LIBCALL,LIBTAGS)
   4 - MAXON compiler (xxx_lib.h,     MODE 1, AMICALL)
   5 - STORM compiler (xxx_lib.h,     MODE 1, AMITAGS,AMICALL)
   6 - all compilers [default]
   7 - all compilers with pragma to inline redirect for GCC
  10 - stub-functions for C - C text
  11 - stub-functions for C - assembler text
  12 - stub-functions for C - link library
  13 - defines and link library for local library base (register call)
  14 - defines and link library for local library base (stack call)
  15 - stub-functions for Pascal - assembler text
  16 - stub-functions for Pascal - link library
  17 - BMAP file for AmigaBASIC and MaxonBASIC
  18 - module for AmigaE
  20 - assembler lvo _lvo.i file
  21 - assembler lvo _lib.i file
  22 - assembler lvo _lvo.i file no XDEF
  23 - assembler lvo _lib.i file no XDEF
  24 - assembler lvo link library
  30 - proto file with pragma/..._lib.h call
  31 - proto file with pragma/..._pragmas.h call
  32 - proto file with pragmas/..._lib.h call
  33 - proto file with pragmas/..._pragmas.h call
  34 - proto file with local/..._loc.h call
  35 - proto file for all compilers
  36 - proto file for GNU-C compiler only
  37 - proto file without lib definitions
  38 - proto file with VBCC inline support
  39 - proto file with special PPC related checks
  40 - GCC inline file (preprocessor based)
  41 - GCC inline file (old type - inline based)
  42 - GCC inline file (library stubs)
  43 - GCC inline file (new style - macro)
  44 - GCC inline file (new style - inline)
  45 - GCC inline file (new style - inline with include lines)
  50 - GCC inline files for PowerUP (preprocessor based)
  51 - GCC inline files for PowerUP (old type - inline based)
  52 - GCC inline files for PowerUP (library stubs)
  53 - SAS-C include file for PowerUP
  54 - Proto file for PowerUP
  60 - FPC pascal unit text
  70 - VBCC inline files
  71 - VBCC WOS stub-functions - assembler text
  72 - VBCC WOS stub-functions - assembler text (libbase)
  73 - VBCC WOS stub-functions - link library
```

```
  74 - VBCC WOS stub-functions - link library (libbase)
  75 - VBCC PowerUP stub-functions - assembler text
  76 - VBCC PowerUP stub-functions - link library
  77 - VBCC WOS inline files
  78 - VBCC MorphOS stub-functions - link library
  80 - pragma/proto redirect (xxx_pragmas.h, SAS/Dice)
  81 - pragma/proto redirect (xxx_lib.h, Aztec/Maxon/Storm)
  82 - pragma/proto redirect (xxx.h, GCC)
  83 - pragma/proto redirect (xxx_protos.h, VBCC)
  90 - stub-functions for C - assembler text (multiple files)
  91 - VBCC PowerUP stub-functions - assembler text (multiple files)
  92 - VBCC WOS stub-functions - assembler text (multiple files)
  93 - VBCC MorphOS stub-functions - assembler text (multiple files)
       100 - PPC assembler lvo file
       101 - PPC assembler lvo file no XDEF
       102 - PPC assembler lvo ELF link library
       103 - PPC assembler lvo EHF link library
       104 - PPC V.4-ABI assembler file
       105 - PPC V.4-ABI assembler file no XDEF
       106 - PPC V.4-ABI assembler lvo ELF link library
       107 - PPC V.4-ABI assembler lvo EHF link library
       110 - FD file
       111 - CLIB file
       112 - SFD file
       120 - VBCC auto libopen files (C source)
       121 - VBCC auto libopen files (m68k link library)
       200 - FD file (source is a pragma file!)
MODE:          SPECIAL 1-7:
                 1: _INCLUDE_PRAGMA_..._LIB_H definition method [default]
                 2: _PRAGMAS_..._LIB_H definition method
                 3: _PRAGMAS_..._PRAGMAS_H definition method
                 4: no definition
               SPECIAL 11-14,40-45,50-53,71-76,78,90-91,111-112:
                 1: all functions, normal interface
                 2: only tag-functions, tagcall interface
                 3: all functions, normal and tagcall interface [default]
TO:            the destination directory (self creation of filename)
ABI:           set ABI type (m68k|ppc|ppc0|ppc2)
CLIB:          name of the prototypes file in clib directory
COPYRIGHT:     the copyright text for CLIB files
HEADER:        inserts given file into header of created file ("" is scan)
HUNKNAME:      use this name for HUNK_NAME instead of default 'text'
BASENAME:      name of library base without '_'
LIBNAME:       name of the library (.e.g. dos.library)
LIBTYPE:       type of base library structure
PRIORITY:      priority for auto open files
Switches:
COMMENT:       copy comments found in FD file
EXTERNC:       add a #ifdef __cplusplus ... statement to pragma file
FPUONLY:       work only with functions using FPU register arguments
NEWSYNTAX:     uses new Motorola syntax for asm files
NOFPU:         disable usage of FPU register arguments
NOPPC:         disable usage of PPC-ABI functions
NOPPCREGNAME:  do not add 'r' to PPC register names
NOSYMBOL:      prevents creation of SYMBOL hunks for link libraries
ONLYCNAMES:    do not create C++ or ASM names
OPT040:        optimize for 68040, do not use MOVEM for stubs
```

```
PPCONLY:       only use PPC-ABI functions
PRIVATE:       includes private declared functions
SECTION:       add section statements to asm texts
SMALLCODE:     generate small code link libraries or assembler text
SMALLDATA:     generate small data link libraries or assembler text
SORTED:        sort generated files by name and not by bias value
SYSTEMRELEASE: special handling of comments for system includes
USESYSCALL:    uses syscall pragma instead of libcall SysBase
```

1.6 examples

```
                Useful examples (with intuition.library):

1) fd2pragma <FD file> TO <pragma dir>

   fd2pragma FD:intuition_lib.h TO INCLUDE:pragma/

        Creates a pragma file for all C compilers and copies it to the
        given directory.

2) fd2pragma <FD file> CLIB <clib file> SPECIAL 12 TO <lib dir>

   fd2pragma FD:intuition_lib.h CLIB INCLUDE:clib/intuition_protos.h
     SPECIAL 12 TO LIB:

        Creates a link library holding stub functions to call tag-functions
        from compilers which do not support them (MaxonC++).

3) fd2pragma <FD file> CLIB <clib file> SPECIAL 13 MODE 3

   fd2pragma FD:intuition_lib.fd CLIB INCLUDE:clib/intuition_protos.h
     SPECIAL 13 MODE 3

        Creates a link library and an include file which allow you to call
        library functions with local base variables in compilers which do
        not support that (MaxonC++). See
                below
                 how to handle these files.

4) fd2pragma <FD file> SPECIAL 34 TO <proto dir>

   fd2pragma FD:intuition_lib.h SPECIAL 34 TO INCLUDE:proto/

        Creates a proto file for the local library base file include, which
        was created in example 3 and copies it to the given directory.

5) fd2pragma <FD file> SPECIAL 35 TO <proto dir>

   fd2pragma FD:intuition_lib.h SPECIAL 35 TO INCLUDE:proto/

        Creates a proto file for all C compilers and copies it to the
        given directory.

6) fd2pragma <FD file> CLIB <clib file> SPECIAL 40
```

```
   fd2pragma FD:intuition_lib.fd CLIB INCLUDE:clib/intuition_protos.h
     SPECIAL 40

        Creates inline include for GCC. This is used by GCC instead of
        pragma files for other compilers.
```

1.7 Option explanation of all user options

```
INFILE is the always needed source file, which describes the library. This
       maybe an FD file (usually together with CLIB file) or an SFD file.

SPECIAL option:

(create a pragma file)
  1: Creates a pragma file for the Aztec compiler; you will see in
     the brackets above what this means.
  2: Same as 1 for DICE compiler.
  3: Same as 1 for SAS compiler.
  4: Same as 1 for MAXON compiler.
  5: Same as 1 for STORM compiler.
  6: This option creates a pragma file usable for nearly all compilers.
     This is the default when no other mode is given.
NOTE: Please always use option 6.
  7: same as 6, but redirects file to inline directory for GCC.

(link libraries and their assembler code)
 10: Creates stub functions in correct C code which handle the varargs
     feature. CLIB parameter is useful with this to get correct functions.
     The only problem with these files is that there is space wasted when
     not all functions are used.

 11: Creates STUB functions for C compilers which are unable to call a
     library directly (the result is ASM source code), and accepts option
     CLIB to create additional function names for C++ compilers like
     MaxonC++.

 12: Same as 11, but the result as a link library, which can be used by the
     C compiler directly.

 13: Creates two files (a link library and a C include) which allows you to
     use local library base variables also in compilers, which do normally
     not support them (MaxonC++). Usually time it is useful to set option
     MODE to 3 or 1. This options needs CLIB keyword for correct results.

 14: Same as 13, but parameters are passed on stack.

 15: Creates STUB functions for PCQ Pascal compilers. The tagcall function
     names are ignored, as they cannot be used with Pascal. The result is
     readable assembler text. The resulting code is the same as for C
     compilers, but the args are taken from the stack in reverse order.
     (C and Pascal normally pass arguments on stack. C passes the last
     argument first (= highest position, stack is filled backwards) and
     first argument last (lowest position). Pascal is the other way round.
     So Pascal stubs are C stubs with the argument order reversed.
```

```
 16: same as 15, but produces a link library.

 17: Creates BMAP files used by certain BASIC variants (e.g. AmigaBASIC,
     MaxonBASIC).

 18: Creates Module for AmigaE.

(assembler LVO files)
 20: Creates lvo file for an assembler.
 21: Same as 20, but with another name.
 22: Same as 20, but there are no XDEF statements in the resulting file.
 23: Same as 22, but with another name.
 24: Creates lvo defintion file as link library (like in amiga.lib)

(proto files - no prototypes)
 30,31,32,33,34: Creates proto files for the C compiler (the difference is
     in the name of the called file).
 35: Creates proto file with calls different inline files for GNU-C and VBCC
     and pragma/xxx_lib.h for all the others.
 36: Creates proto file for GNU-C. This differs from SPECIAL 35 only by
     define __CONSTLIBBASEDECL__ and removed pragma and VBCC call.
 37: Creates proto file, which only calls CLIB and defines library base.
 38: Like 35, but included <inline/xxx_protos.h> for VBCC. This is turned
     of for standard mode 35, as VBCC has problems with (void) argument
     list and inline files.
 39: Like 35, but with special PPC related checks
 fd2pragma knows the correct library base structures for some libraries.
 All the other libraries get 'struct Library' as default.

(GCC inline files)
 40: Creates new style GCC inline files.
 41: Creates old style GCC inline files.
 42: Same as 41, but no "extern" keyword before functions.
 43: New style macro based GCC definitions. Format suggested by Bernardo
     Innocenti. It is still better to use types 40-42, as at the moment it
     produces code which is less optimized. But this type can be used to
     prevent some errors and shortcomings in types 40-42.
 44: Like 43, but uses inline functions instead of macros.
 45: Like 44, but also copies include lines form clib file

(Phase5 PowerUP files)
 50: Creates files like 40, but for PowerUP.
 51: Creates files like 41, but for PowerUP.
 52: Creates files like 42, but for PowerUP.
 53: Creates SAS-C include file for PowerUP (named as pragma files in
     PowerUP files for no logical reason).
 54: Creates a proto file for PowerUP (usable for GNU-C and SAS).

(Pascal stuff)
 60: This creates a unit text file for FPC Pascal compiler.

(VBCC stuff)
 70: This creates inline files for VBCC C compiler.
 71: This is an assembler stub for VBCC-WOS C compiler.
 72: Same as 71, but first argument (r3) is library base (-l of fd2libWOS).
 73: VBCC-WOS link library in EHF format.
 74: Like 73,  but first argument (r3) is library base (-l of fd2libWOS).
```

```
 75: This is an assembler stub for VBCC-PowerUP C compiler.
 76: VBCC-PowerUP link library in ar format.
 77: This creates inline files for VBCC C compiler WOS functions.
 78: VBCC-MorphOS link library in ar format.

(redirect files)
 80: Redirects the call to proto file. Has the name of SAS or DICE pragma
     file. Should be copied to pragmas.
 81: Equal to 80. Has the name of Aztec, Maxon or Storm. Should be copied
     to pragmas (Aztec redirect) or pragma.
 82: Equal to 80. Has the name of GCC inline file. Should be copied to
     inline.
 83: Equal to 80. Has the name of VBCC inline file. Should be copied to
     inline.
Install only redirects for files/compilers you do not have or you will
overwrite real pragmas or inlines.

(multiple files)
 90: Creates STUB functions for C compilers which are unable to call a
     library directly (the result are ASM source codes), and accepts
     option CLIB to create additional function names for C++ compilers
     like MaxonC++.
 91: These are assembler stubs for VBCC-PowerUP C compiler.
 92: These are assembler stubs for VBCC-WOS C compiler.
 93: These are assembler stubs for VBCC-MorphOs C compiler.

(PPC assembler LVO files)
100: Creates lvo file for an PPC assembler.
101: Same as 101, but there are no XDEF statements in the resulting file.
102: Creates lvo defintion file as ELF link library.
103: Creates lvo defintion file as EHF link library.
104: Same as 20, but names without '_'.
105: Same as 21, but names without '_'.
106: Same as 22, but names without '_'.
107: Same as 23, but names without '_'.

(source file recreation)
110: Produce an FD file.
111: Produce an CLIB file.
112: Produce an SFD file.

(auto library open files for VBCC)
120: Produce 2 C code files for auto library opening. Note option
     PRIORITY and LIBNAME for these types.
121: Produce 1 m68k assembler link library for auto library opening.

(FD file)
200: This creates a FD file! The option INFILE has to be a pragma file
     here!
```

1.8 Options for detailed control

```
                MODE:
1) given with SPECIAL 1 to 7:
   - Defines, which #ifdef ...\n#define ... statement is used in the pragma
```

```
     file. Option 1 is default.

2) given with SPECIAL 11 to 14, 40 to 45, 50 to 53, 71 to 76, 78, 90 to 93,
   111 to 112:
   - Defines, which functions should be created. Option 3 is default.
     1 - all functions are taken in normal way with normal name
     2 - only tag-functions are taken with tagcall method and tag name
     3 - means 1 and 2 together

TO: Here you specify the destination directory. The internal names are used,
   but the file(s) will be in the given directory.

ABI: Allows to set initial ABI type for FD file to any of the supported
types "m68k, ppc, ppc0, ppc2". This is equal to ##abi statement.

CLIB: Supply name of the prototypes file in clib directory. If this option
is given together with SPECIAL 11 and 12, additional functions names
with C++ names are created. fd2pragma knows all standard parameter types
defined in exec/types.h and dos/dos.h, all structures and some more types.
All other #typedef's bring a warning. Do not use them in prototypes files!
This parameter is needed by option SPECIAL 10, 12, 13, 40 to 42 and 50 to
53. It may be required for options 71 to 74 (depending on data).
You may define unknown types using the
                fd2pragma.types
                 file.

COPYRIGHT: The copyright text, which is used for CLIB production.

HEADER: This option gives you the ability to specify a file, which should
be inserted after the normal headers and before the clib call of standard
headers (in LVO and ASM files too). If you give "" (empty string) or @ as
filename, the destination file (if already exists) will be scanned for an
existing header. This is useful for updating files. See
                HeaderScan
                to find out how a header must be built to be recognized.

HUNKNAME: This allows to set the HUNK_NAME to any desired value. Default is
'text'.

BASENAME: This allows to overwrite the library basename. The string must be
without leading '_'.

LIBNAME: The name of the library (e.g. dos.library).

LIBTYPE: This allows to define the structure name of base library. The program
knows some types internally and uses "Library" for all the others. Specify the
type here, if it is unknown to fd2pragma.
Example: "LIBTYPE DosLibrary" for dos_lib.fd (is known by fd2pragma).

PRIORITY: Priority for types 120 and above.

COMMENT: Comments which are in the FD file are copied to the pragma or LVO
file, when this option is given! When SORTED is given, this option is
disabled.

EXTERNC: This options adds an #ifdef __cplusplus ... statement to the
pragma file. This options is useful for C compilers running in C++ mode, but
```

it seems, that they do not really need this statement. Only useful with SPECIAL option 1-7, 13, and 14.

FPUONLY: This options is the opposite to NOFPU. It forces fd2pragma to ignore all functions not using FPU registers for passing arguments. It is really useless to both specify FPUONLY and NOFPU.

NEWSYNTAX: Produce new Motorola syntax for ASM output.

NOFPU: This disables usage of FPU arguments. Functions using FPU arguments are not converted with this option (as e.g. Maxon does not support amicall with FPU args). You get a warning for every line containing FPU arguments. It is really useless to both specify NOFPU and FPUONLY.

NOPPC: This disables use of PPC functions. These function are totally ignored now. You get no warning.
It is really useless to both specify NOPPC and PPCONLY.

NOPPCREGNAME: If your PPC assembler does not support the "r" before register names, use this switch and only the register number will be printed.

NOSYMBOL: Does not create symbol hunks for link libraries.

ONLYCNAMES: If CLIB keyword is supplied, but C++ names are not wanted, this keyword prevents creation of these additional names. Also the ASM names are no longer created

OPT040: Optimizes stub functions for 68040, by replacing MOVEM by MOVE's.

PPCONLY: This disables use of all non-PPC functions. These function are ignored totally now. You get no warning.
It is really useless to both specify PPCONLY and NOPPC.

PRIVATE: Also gives you the pragmas or LVO's of private functions. Normally these functions should never be used!

SECTION: Adds sections statements to assembler texts. Only useful with options creating stub files.

SMALLCODE: Forces function references to use (PC) relative addressing instead of absolute addressing.

SMALLDATA: Normally the large data model references the library base as a global variable. In the small data model the reference is relative to register A4 instead. This option is useful for stub texts and link libraries.

SORTED: This option sorts generated files by name and not by bias value. This is only for visibility and does not change the use of the files.

SYSTEMRELEASE: Some special comment handling for system release include files.

USESYSCALL: Instructs fd2pragma to use the syscall pragma instead of a libcall SysBase. This is useful only, when using a SPECIAL option with LIBCALL or by giving LIBCALL directly and converting exec_lib.fd. I think

only SAS compiler supports this statement.

1.9 includes

Useful include system for C compilers:

After programming a long time I arranged my includes in a way that all my
C compilers are able to use the system includes in one directory.

I copied all Amiga system includes to one directory and added some files
created with fd2pragma. These are the system includes you get, for example,
on the Amiga Developer CD.

 - New directory 'pragma' contains xxx_lib.h pragma files for every
   library. These files were created with SPECIAL option 6.
 - New directory 'proto' contains xxx.h proto files which were created with
   with SPECIAL option 35.
 - New directory 'inline' contains xxx.h inline files for GCC. These files
   were created with SPECIAL option 40.
 Directories like 'pragmas' were deleted, when they exist. Only
 'clib', 'pragma', 'proto', and library-specific directories (like 'dos',
 'exec', 'libraries' and 'utility') should remain.

All the others (ISO-C stuff, compiler specials) were copied to another
directory. In S:User-StartUp I use 'Assign ADD' to join the two
directories, so that the compiler may access both.

1.10 The default output file names

fd2pragma automatically produces the names for output files.

If the input file is called xxx_lib.fd or xxx_lib.sfd the xxx is used as
name base, else the basename of ##base statement is used. The "Base" at
the end is stripped.

List of used names:
****

"xxx_cstub.h"   stub functions in C code
 SPECIAL 10

"xxx_lib.h"   C compiler pragma files
 SPECIAL 1, 4 - 7, 81

"xxx_lib.i"   LVO definitions for Assembler
 SPECIAL 21, 23, 100-101, 104-105

"xxx_loc.h"   Local library base definition file for C compilers
 SPECIAL 13, 14

"xxx_lvo.i"   LVO definitions for Assembler
 SPECIAL 20, 22

```
"xxx_pragmas.h"  C compiler pragma files
 SPECIAL 2, 3, 8, 53, 80

"xxx_protos.h"   VBCC inline files
 SPECIAL 70, 83

"xxx_stub.s"     stub functions as Assembler text (68K, PPC)
 SPECIAL 11, 15, 71, 72, 75

"libxxx.a"       ar object file archive (link library)
 SPECIAL 76,78

"xxx.bmap"       BASIC definition file for function calls
 SPECIAL 17

"xxx.h"          C compiler proto or inline files
 SPECIAL 30 - 44, 50 - 52, 54, 82

"xxx.lib"        link library (68K, PPC)
 SPECIAL 12, 16, 73, 74

"xxxlvo.o"       link library lvo entries (68K, PPC)
 SPECIAL 24, 103, 107

"xxxlvo.o"       link library lvo entries (PowerUP)
 SPECIAL 102, 106

""xxx.m"         E module
 SPECIAL 18

"xxx.pas"        FPC unit text file
 SPECIAL 60

"xxx_lib.fd"     FD file
 SPECIAL 110, 200

"xxx_protos.h"   CLIB file
 SPECIAL 111

"xxx_lib.sfd"    SFD file
 SPECIAL 112

List of used single-file names:
******************************

The xxx in these files is the funtion name.

"xxx.s"          stub function as Assembler text (68K, PPC)
 SPECIAL 90-93
```

1.11 link libraries

```
About created link libraries (SPECIAL Option 12-14):
```

The created link libraries are relatively large compared to other link libraries. The size of the link library has nothing to do with the size of the resulting program you create. The code part of my link libraries is relatively short, but fd2pragma defines a lot of texts (which are NOT copied to the executable program created). These texts are for easier identification and every function also gets different names:

```
  1) the normal asm name:  <name>               (e.g.  CopyMem)
  2) the normal C name:   _<name>               (e.g. _CopyMem)
  3) the normal C++ name:  <name>_<params>      (e.g.  CopyMem_PvPvUj)
  4) when a function parameter is STRPTR, a second C++ name is created
  5) string 3 with different register spec (only func-ptr args with reg-params)
  6) string 4 with different register spec (only func-ptr args with reg-params)
```

Forms 3-6 occur only, when you use CLIB keyword. Forms 5 and 6 should nearly never occur! With SPECIAL options 13 and 14 the number of strings is doubled. The different names give a lot more flexibility and only make the link library bigger. These names are only visible to the linker program. The resulting executable usually is a lot smaller than the link library!

If ONLYCNAMES is given the number of names is reduced to 1 (type 2).

I think the code part of the link libraries is totally optimized. I do not know of any possible improvement to make it shorter.

If you join all the link libraries made with SPECIAL 12 and join it with small.lib from Developer kit, you get a replacement for amiga.lib.

1.12 proto files

fd2pragma is able to generate different proto files, but I suggest using only the file generated with SPECIAL option 35.

For system libraries and some others the correct base structure is used. Other unknown basenames get "struct Library *" as default. You may change that in the created proto files when another structure is correct.

The proto files support following define:

```
 __NOLIBBASE__
```

When this is set before calling the proto file, the declaration of the global library base is skipped, so that can be done in source-code. This define is also used for GCC.

The file created with SPECIAL 36 supports an additionally define:

```
  __CONSTLIBBASEDECL__
```

When this is set to "const" the library base is handled as const. This may shorten the produced code, but you need a sourcefile without this define to initialize the base variable.

1.13 defines used in include files

```
                The first #ifdef/#define statements of created C includes:

fd2pragma has a set of different define names for different include files.
These names are internal to allow double-inclusion of one include files
without getting errors. Standard system includes use the same system.

The normal names are: (example intuition.library)
 proto files:                           _PROTO_INTUITION_H
 local library base files:              _INCLUDE_PROTO_INTUITION_LOC_H
 standard pragma files:                 _INCLUDE_PRAGMA_INTUITION_LIB_H
 C stubs files:                         _INCLUDE_INTUITION_CSTUB_H
 inline files for GCC                   _INLINE_INTUITION_H
 PPC inline files for GCC               _PPCINLINE_INTUITION_H
 SAS-C PPC pragma files                 _PPCPRAGMA_INTUITION_H
 vbcc inline files                      _VBCCINLINE_INTUITION_H
 clib files                             CLIB_INTUITION_PROTOS_H

Non-fd2pragma names are:
 other includes (path_name_extension)   INTUITION_INTUITION_H

These names should never be used in other files or sources! This rule is
broken for some standard system includes, but is generally true. Compiling
may be a few seconds faster when you check these names before the #include
line, but in this case the names must be standard and they are not!

Some defines allow the user to change the behaviour of the includes:

  __NOLIBBASE__
This is used in proto files. See
                Proto files
                 for more information.

  NO_INLINE_STDARG or NO_PPCINLINE_STDARG
For GCC inline files a lot of defines exist, but this seems to be the most
important. It disables the creation of varags/tagcall functions.
For other inline defines check the created inline files.

  NO_OBSOLETE
This is used in SPECIAL 7 and 80-83 redirect files. If it is used, the
compiler will return an error instead of using the redirect.

  <library>_BASE_NAME
Function definitions in the inline files refer to the library base variable
through the <library>_BASE_NAME symbol (e.g. INTUITION_BASE_NAME). At the
top of the inline file, this symbol is redefined to the appropriate library
base variable name (e.g., IntuitionBase), unless it has been already
defined. This way, you can make the inlines use a field of a structure as
a library base, for example.
```

1.14 local library base files

```
When using SPECIAL options 13 and 14 you get two files called libname_loc.h
```

and libname_loc.lib. The second one is a link library and should be passed to the compiler with program settings or in makefile. The first one is a C header file and should be used as a replacement for files in clib, pragma, proto and pragmas directories. Always use the libname_loc.h file instead of these files and not together with them! Do not mix them.
I suggest copying the header file into a directory called "local".

This file holds prototypes like those in the clib directory, but with struct Library * as the first parameter and the name prefix LOC_. Together with the prototypes there are some defines redefining the function name to the old one and passing the library base as first parameter. These defines allow you to use the local library bases as normal as global bases. For tag-functions and some exceptions these defines do not work and you have to call the LOC_ function directly and pass the library base as first parameter.

Use need the CLIB keyword together with SPECIAL option 13 and 14.

1.15 How the header scan works

Giving the HEADER option lets fd2pragma insert the file (you have to give a filename with HEADER option) at the start of the LVO/Pragma/Proto/stub file. When you pass "" or @ as filename, fd2pragma scans the destination file (if it already exists) for a header and copies this header to the new file.

How scanning is done:
fd2pragma scans for a block of comment lines. So when a line starting with '*', ';' or '//' is found, this line is the first header line. The header ends before the first line not starting this way. Additionally, when fd2pragma finds first a line starting with '/*' it scans until a line holds '*/'. This then is the last line of header. Same is done for Pascal comments using '(*' and '*)' or '{' and '}'.

C and ASM files are scanned the same way, so sometimes fd2pragma may get a wrong header.

1.16 Method of tag-function handling

The tag-functions are supported by certain comments. Note that the official includes from the Native Developer Update Kit do not have these comments included. Let's look at an excerpt from the FD file muimaster_lib.fd:

```
MUI_NewObjectA(class,tags)(a0,a1)
*tagcall
MUI_DisposeObject(obj)(a0)
MUI_RequestA(app,win,flags,title,gadgets,format,params)(d0,d1,d2,a0,a1,a2,a3)
*tagcall
MUI_AllocAslRequest(type,tags)(d0,a0)
*tagcalltags
```

The comments tell us that MUI_NewObjectA, MUI_RequestA, and MUI_AllocAslRequest should have stub routines. The respective names are MUI_NewObject, MUI_Request, (as the comment has just the word tagcall) and MUI_AllocAslRequestTags (as the comment has the word tags included).

Another possibility would be to write something like

```
SystemTagList(command,tags)(d1/d2)
*tagcall-TagList+Tags
```

This would create a stub routine or tagcall pragma SystemTags (dropping the word TagList, adding the word Tags).

fd2pragma is also able to create the names automatically. Most times this should be enough, so you do not have to use the abovementioned method. In case you really use the above method, I suggest using always the one with '+' and '-' signs!

Tag-functions in standard Amiga includes differ a bit in their naming conventions, so it is easy to find them:

normal function name	tag-function name
xxxA	xxx
xxxTagList	xxxTags
xxxArgs	xxx

Also the arguments given in the FD file may define a function as a tag-function. If the last argument is one of the words "tags", "taglist" or "args", then the function has a tag-function named xxxTags or xxxArgs.

There are some exceptions to these rules (some dos.library and utility.library functions) which are handled automatically.

Sometimes fd2pragma detects functions as tag-functions which are no tag-functions. This is a problem of the detection, which is based on function name. Not all programmers follow the unofficial guidelines and thus fd2pragma produces wrong stuff. If you detect such functions, inform me and I will remove that behaviour. In any case you may add the line:

```
*notagcall
```

as next line in FD file and the detection is turned off for that function.

1.17 fd2pragma.types definition file

This file allows you to define unknown types so that correct C++ names can be build and also created inline files are valid. The file must be in current directory or in program directory. If no file can be read, the internal version is used.

When fd2pragma does not recognize a type you get a warning, telling you the line number and argument number. Argument number 0 means the return value of the function. Check the type and add it to the list and fd2pragma will know it afterwards.

```
Lines starting with * are seen as a comment and are ignored.
The description format is one 'unknown type : known type' in every line.

Unknown type: A type fd2pragma does not know. It consists of only
one word. All other stuff like C keywords and * is handled internally and
cannot be supplied here.

Known type: The type which is hidden behind the unknown one. It consists of
struct + name, enum + name, union + name, signed, unsigned, const, long
(32 bit), short (16 bit), char (8 bit), int, double and float.
For nameless structs or enums set ? as name.

Structs without a name, but used by a typedef (like: typedef struct {...}
name) get the typedef name (struct name).

Also register definitions in style of "register __d0 long" can be used.

See already-added example types in the supplied file and send me new types
whenever you get one. It's best to send the definition file also, so I can
check the definitions myself.
```

1.18 Functions using FPU registers

```
There may be FD files containing functions with FPU register arguments.

You may use NOFPU keyword to disable processing of these functions.

For link library stub creation of such functions the clib file is
required to determine if argument is double (64 bit) or float/single
(32 bit). An warning message appears when it was not possible to get the
correct type. In this case float is used always.
Extended format is currently not supported.

fd2pragma creates amicall and flibcall pragmas for these functions, but
some compilers cannot use them (e.g. MaxonC).
```

1.19 Files used and created for Phase5's PowerUP boards

```
                It is very hard to tell something about these as I myself do not ↩
                   know very
much about it. Maybe that is caused by missing documentation.

fd2pragma creates needed files with SPECIAL options starting at number 50.
fd2pragma produces nearly the same files as are made by the special
fd2inline version 1.12 (by Ralph Schmidt). I fixed the protos a bit and
arranged files to follow my general style, but changed nothing in the way
they work.

There are also PowerUp files for
                VBCC
                . See the related
```

```
chapter for more information.

The files need following directory structure
PPCINLUDE:
  powerup
    gcclib              support files
    ppcinline           GCC inline files (SPECIAL 50)
    ppclib              support files
    ppcpragmas          SAS-C function definitions (SPECIAL 53)
  proto                 proto files for GCC/SAS-C (SPECIAL 54)

Using this SMakeFile with SAS-C I had not yet larger problems:

SCOPTS = RESOPT PARAMETERS=REGISTERS NOSTACKCHECK STRINGMERGE OPTIMIZE \
OPTIMIZERINLINELOCAL MEMORYSIZE=HUGE OPTIMIZERTIME
SLOPTS = SMALLCODE STRIPDEBUG NOICONS

PROGRAM = # enter name here

ALL: ELF PPC
ELF: $(PROGRAM).elf
PPC: $(PROGRAM).ppc

$(PROGRAM).ppc: $(PROGRAM).ppc.o
  slinkppc $(SLOPTS) PPC FROM LIB:c.o $? LIB LIB:scppc.lib TO $@
  StripHunks $@  # *** Aminet/util/misc/StripHunks.lha ***

$(PROGRAM).ppc.o: $(PROGRAM).c
  scppc $(SCOPTS) $? HUNKOBJ OBJNAME=$@ DEFINE=AMIGA

# ************************** GCC stuff needs ADE 2 CD

PPCLD    = ADE-2:AmigaOS/ready-to-run/bin/ppc-amigaos-ld
PPCSTRIP = ADE-2:AmigaOS/ready-to-run/bin/ppc-amigaos-strip

$(PROGRAM).elf: $(PROGRAM).elf.o
  $(PPCLD) -r -s -o $@ lib:c_ppc.o $? lib:scppc.a lib:end.o
  $(PPCSTRIP) --strip-unneeded $@
  Protect $@ rwed

$(PROGRAM).elf.o: $(PROGRAM).c
  scppc $(SCOPTS) $? OBJNAME=$@ DEFINE=AMIGA
```

1.20 VBCC compiler files

```
SPECIAL 70: inline files

 These files can be created using fd2lib with -pr Option as well. This
 is an undocumented option, so you see there may be problems. First the
 inlines do not support tagcall functions. And if functions with lots of
 arguments are used, the code quality decreases really fast. Their use is
 not recommended!

SPECIAL 71-72: stubs for VBCC-WOS
```

```
 These are normal stub libraries as Assembler text. Mode 72 is equivalent
 to the argument -l of fd2libWOS and says that the first argument (r3) is
 the library base. These files do not support the small data model.

SPECIAL 73-74: stub link libs for VBCC-WOS

 These are normal stub libraries in EHF format. Mode 74 is equivalent to
 the argument -l of fd2libWOS and says that the first argument (r3) is the
 library base. These files do not support the small data model.

SPECIAL 75: stubs for VBCC-PowerUP

 These are normal stub libraries as Assembler text. These files support
 the small data model and thus the SMALLDATA argument.

SPECIAL 76: stub link library for VBCC-PowerUP

 This is a normal stub library in ar format. This file supports the small
 data model and thus the SMALLDATA argument.

SPECIAL 91: stubs for VBCC-PowerUP

 These are normal stub libraries as multiple files Assembler text. These
 files support the small data model and thus the SMALLDATA argument.

SPECIAL 92: stubs for VBCC-WOS

 These are normal stub libraries as multiple files Assembler text. These
 files do not support the small data model.

SPECIAL 93: stubs for VBCC-MorphOS

 These are normal stub libraries as multiple files Assembler text. These
 files support the small data model and thus the SMALLDATA argument.
```

1.21 Words and phrases

```
                clib-files, prototypes:
 For Amiga C functions the prototypes needed in C compilers are stored in
 a directory called clib. The files are named libname_protos.h. The CLIB
 option needs the name of such a file as a parameter. These files are needed
 by fd2pragma to create correct data with some options.

data models:
 Most C compilers offer 2 different data models called large and small
 data (or far and near).
 Large data means all data is stored in a HUNK_DATA as a normal variable,
 which is accessed by its address. One access normally needs 4 bytes of
 space in program code and 4 bytes as relocation entry.
 The small data model needs less space. Here the data is stored as one
 structure. At the program start the compiler adds an instruction, which
 loads the structure address into register A4. In the following program the
 data is always accessed related to the register A4. One access now only
 needs 2 bytes in program code and no relocation entry (saving 6 bytes for
 each access).
```

```
 But this model has some problems:
 - Data size may not exceed 64KB (32KB for older compilers), as relative
   information uses only 2 bytes.
 - When program code is called from outside the program (e.g. hook code),
   it is not guaranteed that the A4 register still holds the base reference.
   So these functions need to be __saveds (SAS-C) or call functions like
   GetBaseReg() (Maxon-C++) to reload A4 register.

inline system calls:
 GNU-C (GCC) uses a different system to call Amiga system functions. The
 needed files are stored in a directory called inline. Starting with
 version 2.45 this program is able to produce inline files as well. Before,
 you needed to use fd2inline program. I suggest using the proto file you
 can create with SPECIAL 35 instead calling inline files directly.

.lib file, link library:
 A link library is a file containing functions which are added to the final
 executable at linking time. The other method are runtime or shared
 libraries (#?.library) which are called in runtime and thus take no space
 in the executable program.

pragma:
 C allows non-standard (compiler private) definitions called pragmas. Most
 Amiga compilers use them to define system library calls. There exists 5
 different
                #pragma
                 statements, which are used by different compilers.
 I suggest using the proto file you can create with SPECIAL 35 instead
 of calling pragma files directly.

proto file:
 C compilers like SAS have a special directory called proto with files in
 it calling the pragma and prototypes files. This is useful, because
 different compilers store their pragma files in different directories (or
 use other methods to define system calls), but all use one proto file. I
 did not use them for a long time and called the pragma files directly, but
 this makes it harder to switch to another compiler. So now I use proto
 always.

stub, stub function:
 A stub function is a function, which converts between different inter-
 faces. For example C supplies function parameters on stack, but Amiga
 libraries get them in registers. A stub function for that gets the
 arguments on stack copies them into the registers and calls the Amiga
 function. Newer C compilers have #pragmas to do that internally, but some
 calling mechanisms are not supported by all compilers. MaxonC++ for
 example does not support the tagcall.

tag-functions:
 C allows functions to get a variable number of parameters
 everytime they are called. These varargs functions have in their
 prototypes "..." at the end (e.g. printf). Amiga system libraries use this
 mechanism for supplying so-called tags. (See Amiga programmers
 documentation for that.)
 The name tag-functions is not the best, because there are also some
 functions getting variable args which are not tags (e.g. Printf), but it
```

expresses well what is meant.

1.22 Known bugs and problems

- Pragma creation with cia_lib.fd fails with no ##basename error. This is desired by Amiga OS programmers to allow passing the library base as the first argument. In this case the C compiler function call does not work. You may add a ##basename statement to the FD file and get a working pragma file, but this file will not work together with clib/cia_protos.h file. Using option 14 or 15 instead generates a valid link file to use with the clib file. The created text file is of no use and can be deleted. When using created proto file (e.g. SPECIAL 35), you may remove the lines calling the pragma file.

- mathieeedoubtrans_lib.fd and mathieeedoubbas_lib.fd both use 2 registers for one double value. fd2pragma creates only link libraries and definition files for them. Certain types do not support such behaviour and aren't created.

- mathieesingtrans_lib.fd, mathieeesingbas_lib.fd, mathffp_lib.fd and mathtrans_lib.fd use float arguments in normal data registers. The PPC data types will produce invalid code for this library. As there is really no sense in using these libraries from PPC this is not really a problem.

- The redefines of SPECIAL 13 and 14 are illegal when a function has the same name as a structure (e.g. DateStamp of dos.library). You have to remove the #define line for that function.

- When clib file contains more than one function with the same name, fd2pragma always takes the first one, which may not always be the right one. For good include files this should never happen.

- The return register is D0 for all functions. Also, flibcall always uses D0 as the return register. All the pragma types normally support other return registers in their design, but I never saw any pragmas really using that and I think some compilers (e.g. MaxonC++) do not even accept such pragmas.

Include file errors (in Includes release 44.1):
- Function JulianMonthDays of datebrowser.gadget is erroneously called JulianMonthsDays in clib file.

- Using created graphics pragma brings an error on GetOutlinePen. This is not my fault, but an include error. Remove the line
  `#define GetOutlinePen(rp) GetOPen(rp)`
  in graphics/gfxmacros.h or turn it around to
  `#define GetOPen(rp) GetOutlinePen(rp)`
  as this works ok.

- OpenAmigaGuideA of amigaguide.library has a '*' as argument name in FD file. This causes an error for inline files.

- ActivateCxObj of commodities.library has 'true' as argument name in clib and FD file. This may cause errors.

```
Always use the newest Include files!

* Automatically-created files may not always be fully correct (it may happen
* rarely, but it sometimes happens). When you find such a condition (if not
* mentioned above), please contact me, and when useful and possible I will
* include a fix in the program.
* There are so many exceptions in the official include files. How many are
* in non-default system include files?
```

1.23 How self-made libraries should be designed

```
                As the main author of xpkmaster.library packer interface system ↩
                   and include
creator for other libraries (and, not to forget, the author of fd2pragma) I
gained some experience in how library functions should be designed.

Expanding the possibilities of functions usually brings certain problems.
Some ways you can reduce these problems:
- Design your functions as tag-functions. For these it is really easy to
  implement new functionality. Believe me, it was a hard lesson for me
  to learn. :-)
- It is always a good idea to add an "AllocStructures" function, which
  allocates all structures which are needed by your library system. Force
  the user to ALWAYS use this function. Future additions to the structures
  are then really easy. A "FreeStructures" function frees the stuff later.
  The "AllocStructures" may get an ULONG type and tags as argument. The
  tags allow to change initialisation behaviour.
- Structures should use pointers instead of byte-arrays (for texts) or
  other directly included structures. This makes expansion possible, but is
  a little more complicated.
- The function name should reflect your library name; for example, xpkmaster
  functions all start with "Xpk" and xfdmaster functions start with "xfd".
- D0-D7, A0-A3
               registers
                are always available, but you should not use
  more than 8 of them in one function (except in very special cases).
  Usually, it is better to move lots of the arguments into an argument
  structure passed in one of the address registers. This also allows easy
  future additions as described above. Also, tag systems allow reduction of
  the argument count (and the user does not need to pass default values for
  unused arguments).

A bit about how you should design names to make the work of utilities like
fd2pragma possible:
- Tag-functions should always receive the tagitem array (struct TagItem *)
  as last element.
- The normal function should end in a big 'A', the tag-function has same
  name without 'A'.
               Other methods
                are described in that document, but this
  seems to be the best.
- Pointers should be in A-registers, data should be in D-registers. The
  tagitem array pointer should always be in an address register.
- Try to sort your registers in a order so that MOVEM.L can be used to get
```

```
  them from stack into registers: D0,...,D7,A0,...A3(,A4,A5)
  This affects autodocs, prototype files, and FD files. Assembler
  programmers have no problems with argument order, but it is useful for C.
- Do not use A6 and A7 registers for arguments (which is nearly impossible).
- Try not to use A4 and A5 register. Some C compilers store data in these
  registers. There may be problems with functions using these (e.g. inline
  creation has some restrictions).
- The return value should always be in D0 (and only there). It is really
  complicated to access multi-return functions from C programs.

How to design
               FD files
               :
- For tag-functions the last element in the FD file should be named 'tags'.
- The argument names should not be the same for different arguments as
  these names are used for certain files by fd2pragma and other utilities.
- If you do not follow the above-mentioned name convention for
  tag-functions, use the
               tagcall comments
                to define tag-function names.
- For separating registers, you may use a "," or a "/". In standard system
  FD files, the "/" is used to separate these arguments that are in
  correct order to be used with MOVEM.L call (see above). It is not
  important which separator you use, as fd2pragma and nearly all other
  tools accept both, but I suggest either using "," only or the standard
  style.
- The file should be named <library>_lib.fd, as fd2pragma interprets the
  file name and uses the first part (before _lib.fd) to generate the
  destination filename. If the file does not end in _lib.fd the defined
  basename is used.

For plain assembler programmers: contact a experienced C programmer to get
a fine interface (or learn C :-).
```

1.24 Design description of varargs pragmas

```
Pragmas are a method to implement compiler-specific behaviour in C language
programs. Refer to the ISO-C standard to find out what this means.

Most Amiga compilers use the pragma system to implement calls to the Amiga
shared libraries. Since there exist different compilers, there are different
pragmas also, so I describe the design of these here:

The used format descriptors:
  base          Name of the library base (e.g. IntuitionBase)
  function      Name of the function (e.g. OpenWindowTagList)
  hexoffest     Bias value of the library (e.g. 25E)
  offset        Bias value of the library (e.g. 0x25E or 606)
  retret        return register (e.g. d0)
  reglist       used registers (e.g. a0,a1), maybe empty
  magic         magic value, see below how this is build
  fmagic        magic value for floating point

pragma libcall (SAS-C, DICE, supported by Storm):
 Normal call method for shared libraries.
```

```
 #pragma libcall base function hexoffset magic
 #pragma libcall IntuitionBase OpenWindowTagList 25E 9802

pragma flibcall (SAS-C, supported by Storm):
 Call method, when arguments are passed in FPU registers (maybe mixed with
 CPU arguments).

 #pragma flibcall base function hexoffset fmagic
 #pragma flibcall mesamainBase glAlphaFunc 3C 10000002

pragma syscall (SAS-C, supported by Storm):
 Special method to call exec.library function without a Sysbase, but
 directly with address 4.W.

 #pragma syscall function hexoffset magic
 #pragma syscall AllocMem C6 1002

pragma tagcall (SAS-C, supported by Storm):
 This type allows to call shared library functions with variable argument
 lists (useful for tag-functions or e.g. Printf).

 #pragma tagcall base function hexoffset magic
 #pragma tagcall IntuitionBase OpenWindowTags 25E 9802

pragma amicall (Aztec-C, Maxon++, Storm, [SAS-C supports this with errors]):
 Same as libcall for other compilers. fd2pragma can produce amicall pragmas
 containing FPU arguments (FP0..FP7), but Storm seems to be the only
 compiler accepting these.

 #pragma amicall(base, offset,[retreg=]function(reglist))
 #pragma amicall(IntuitionBase,0x25E,d0=OpenWindowTagList(a0,a1))

 Most compilers do not support the return register place (and fd2pragma
 also does not), so this really is:

 #pragma amicall(base, offset,function(reglist))
 #pragma amicall(IntuitionBase,0x25E,OpenWindowTagList(a0,a1))

pragma tagcall (Storm):
 This type allows calling shared library functions with variable argument
 lists (useful for tag-functions or e.g. Printf).

 #pragma tagcall(base, offset,function(reglist))
 #pragma tagcall(IntuitionBase,0x25E,OpenWindowTags(a0,a1))

The magic value:
First thing of every register as one hexadecimal number:
d0..d7,a0..a7,fp0..fp7  -->     0..7,8..F,10..17

Now the magic value is built in the following form:
  <reglist><retreg><numregs>
The reglist is built in reverse order, so the first argument is last entry:

OpenWindowTags(a0, a1): returns in d0, has 2 arguments
arguments in reverse order:     98
return register:                0
```

```
number of arguments:            2
result:                         9802

A void function thus gets:      00

The magic value for flibcall is like that, but with 2 spaces for EVERY
entry:

glAlphaFunc(d0,fp0): returns in d0, has 2 arguments
arguments in reverse order:     1000
return register:                00
number of arguments:            02
result:                         10000002
```

1.25 Design description of SFD files

The sfd fileformat holds nearly all the information needed to create developer files for library programming. It consists of command lines, comment lines and function descriptions.

The commands always start with "==". Comment lines start with "*" sign.

Following commands exist for file start:

==id Describes the RCS-ID string for the SFD file. This is ALWAYS the first line!
==id $Id: xadmaster_lib.sfd,v 10.0 2001/05/01 10:58:11 stoe Exp $

==base This defines the name of the library base. Normally starts with an "_" and ends in "Base". Required always.
==base _xadMasterBase

==basetype This allows to define the type of the library base structure. It defaults to "struct Library *". It starts with "struct" and ends with pointer sign "*". Should be used, if not "struct Library".
==basetype struct xadMasterBase *

==libname This allows to define the name of the library.
By default this is created from the basename of the library by removing base extension, converting it to lower and adding ".library". Should be used, if this default does not work.
==libname xadmaster.library

==include To get #include lines generated in clib file, you need to specify this keyword. You can use as much of them as necessary. This data normally starts with "<" and ends in ">" (normal C style.
==include <exec/types.h>

==bias This defines the function offset the function starts. The values are always multiple of 6. For libraries it starts with 30, devices normally starts with 42. Required always.
==bias 30

Now some commands which may be used in whole file:

==varargs This introduces an varargs alias name. The following function definition keeps the bias of the previous one.
There are mainly 2 types of varargs funtions:
1) Using one instance of the varargs parameter followed by ...

```
struct Menu *CreateMenusA(struct NewMenu *newmenu,
  struct TagItem *taglist) (A0,A1)
==varargs
struct Menu *CreateMenus(struct NewMenu *newmenu,
  Tag tag1, ...) (A0,A1)
```

2) Using only ...

```
LONG VFPrintf(BPTR fh, UBYTE *format, LONG *argarray)
  (d1,d2,d3)
==varargs
LONG FPrintf(BPTR fh, UBYTE *format, ...) (d1,d2,d3)
```

The first type is used for functions, which need at least one parameter, like all the functions using TagItem structure.

The second type is needed for functions like Printf(), which can be called without any additional variable data.

NOTE: The register list of varargs functions equals the list of corresponding function always! The arguments are equal as well except for the last argument.

==alias To allow backwards compatibility it is sometimes necessary to define multiple names for one function. Using this command you can define a new additional name. This function has same bias as the previous one! You can specify an alias for varargs as well, if you added an varargs definition before. Always think of correct order

```
LONG DoPkt(struct MsgPort *port, LONG action, LONG arg1,
  LONG arg2, LONG arg3, LONG arg4) (d1,d2,d3,d4,d5,d6)
==alias
LONG DoPkt1(struct MsgPort *port, LONG action, LONG arg1)
  (d1,d2,d3)
```

NOTE: The register and argument lists may be shorter than the lists for corresponding function, but the registers and arguments must be equal.

==reserve This allows to skip function slots which are internal, obsolete or should not be visible for any other reason. The command is followed by the number of slots to reserve.

```
==reserve 2
```

==private This allows to declare entries as private. Programs parsing SFD normally handle following lines as if declared with ==reserve. With another call the are handled as normal functions. This allows to create private use definition files and release files, which do not have internal function definitions.

```
==private
```

==public This is the opposite to ==private and turns back normal behaviour. It is always a good idea to place this at the top of

```
            SFD file.
            =public

==version   To describe the version of the module that subsequent functions
            appear in. This causes comments to be generated in output files.
            ==version 5

==end       Required always at end of file.
            ==end

==abi       Like Frank Wille did for FD file I extended this format to
            support PPC function descriptions also. To specify normal Amiga
            functions, the command
            ==abi M68k
            must be used. To specify PPC functions PPC, PPC0 or PPC2 must be
            used.
            Differences of PPC ABI types:
             PPC   gets librarybase in r3, other arguments follow
             PPC2  gets librarybase in r2
             PPC0  no librarybase passed

            The register part of function definition contains empty brackets
            for PPC ABI's!

Function definitions:

Anything that is not an sfd command or a comment must be part of a function
definition. A function definition consists of three parts:
 - the return value/function part,
 - the parameter definition,
 - and the register definition:
<return/function>(<parameters>)(<registers>)

All three parts must be present. They may cross lines. A particular function
definition is terminated by the second close paranthesis. A function
definition must start on a fresh line.

Examples:
VOID OpenIntuition() ()

 - Even an empty parameter list needs an empty register list.
 - The C types void, unsigned, int, short, char, float and double are not
   recommended. Use types of <exec/types.h>.
 - Don't put a VOID in an empty parameter list.

PLANEPTR AllocRaster(UWORD width, UWORD height) (D0,D1)

 - The parameter list must contain typed variables.
 - The register list is generally delimited by commas.

DOUBLE IEEEDPDiv(DOUBLE dividend, DOUBLE divisor) (d0-d1,d2-d3)

 - DOUBLE registers must be delimited by a dash.
 - The register list may use either capital or lower case.

struct Layer *CreateUpfrontLayer(struct Layer_Info *li,
struct BitMap *bm, LONG x0, LONG y0, LONG x1, LONG y1,
```

```
LONG flags, [struct BitMap *bm2]) (A0,A1,D0,D1,D2,D3,D4,A2)

 - The function definition can cross lines, but one argument should be
   unbroken (e.g. break after "," or ")" signs).
 - Optional parameters are indicated by being enclosed in braces.
   This currently has no effect, but may be supported in future programs.

struct Window *OpenWindowTagList(struct NewWindow *newWindow,
struct TagItem *tagList) (A0,A1)
==varargs
struct Window *OpenWindowTags(struct NewWindow *newWindow,
ULONG tag1Type, ...) (A0,A1)

 - The only time "..." may appear is for a varargs definition.
```

1.26 Design description of FD files

```
                FD files are a standard way to describe the interface for Amiga  ↩
                   shared
libraries. This format consists of 3 main line types:

1) Commands - always start with ##
2) Comments - always start with *
2a) Plain comments - describe the contents for the user
2b) Command comments - add some
                functionality
                 (like *tagcall or *notagcall)
3) Function descriptions - provide the main information

1)
Know commands are:
  ##abi         define Application Binary Interface
                Allowed types: M68k, PPC, PPC0, PPC2
  ##base        Define the library base name, always starts with a "_"
  ##bias        Define a bias value, minimum value is 30, always a sum of 6
  ##end         Always in last line, finishes the file
  ##public      Declare the following functions as public
  ##private     Declare the following functions as private

Normally a FD file is build this way:
##base _IntuitionBase
##bias 30
##public
....
##end

Frank Wille extended this format to support PPC function descriptions
also. He introduced the ##abi command. To specify normal Amiga functions,
the command
##abi M68k
must be used. To specify PPC functions PPC, PPC0 or PPC2 must be used.
Differences of PPC ABI types:
 PPC   gets librarybase in r3, other arguments follow
 PPC2  gets librarybase in r2
 PPC0  no librarybase passed
```

```
The register part of function definition contains empty brackets for PPC
ABI's!

2)
Comments always start with a * and are normally ignored (except when
they are "Command comments". Check the
                description
                 of these to know what
this means.

3)
A function description is build always like that:
name(argnames)(registers)

e.g. OpenWindowTagList(newWindow,tagList)(a0/a1)

For void functions argnames and registers are both empty. Registers are
separated by "," or "/". See
                library design
                 to find out the difference.
fd2pragma accepts registers d0..d7,a0..a7,fp0..fp7.
If the last argument is "tags", "taglist" or "args", fd2pragma creates
additionally a varargs or tag-function. See
                tag-function
                 section to learn
about that.

In the file you may add ##bias lines, when you want to skip some internal
functions, or ##private lines, when you want some functions not to be used
in normal case. The ##public keyword can be used to enable the normal
system again.

There is an incompatible FD file format addition created by
Haage&Partner for their Storm compiler. It supports an additional
command ##shadow.

 The format is:
 function description
 ##shadow
 tag-function description

 The first line is like above a normal function description. The ##shadow
 command sets back the bias to the same value as before and accepts the next
 line as a tag-function!

 example:
 OpenWindowTagList(newWindow,tagList)(a0/a1)
 ##shadow
 OpenWindowTags(newWindow,tagList)(a0/a1)

 Do not use that format!
```

1.27 How 680x0 processor registers are used

```
                On Amiga computers there exist some conventions about how  ↩
                   registers of
680x0 processors are used:

1) Register A7 either holds "user stack pointer" USP or "supervisor stack
   "pointer" SSP. The second should not be important for the normal Amiga
   user. The stack is used to store variables, return addresses and, for
   some compiler languages (e.g. C), to store function arguments.
2) Register A6 holds a pointer to the library base when you call a
   library function. In front of the library base is a jump table, which
   has an entry for every library function. The function itself is reached
   by jumping into the corresponding entry, which is specified by the base
   value in FD file (always negative).
3) Registers D0, D1, A0, A1 are scratch registers. This means after
   calling a library function the contents of these registers is no longer
   valid! The contents of all the other registers are preserved.
4) Register D0 normally contains the return value of a called function.
5) Register A4 holds the data base pointer when you use C compiler in the
   small data model. You should not use this register for argument passing.
6) Register A5 is used by C compilers to store a pointer to a field (part
   of the stack) for local variables. You should not use this register for
   argument passing.

This means altogether you have 8 free data registers and 4 free address
registers for passing arguments to functions. See how to
                design libraries
                ,
if you need more than 4 address registers or altogether more than 8
arguments.
```

1.28 Scripts for automatic file creation

```
In the directory Scripts there are some scripts which allow you to generate
necessary files automatic.

MakeInline - Generates inline files for standard system libraries, which
can be used with GCC compiler:
 Options:
  FDPATH     This is the path where your system FD files are stored. The
             path must end in ':' or '/'.
  CLIBPATH   This is the path where your system prototype files are
             stored. The path must end in ':' or '/'.
  INLINEPATH This is the path where created files should be stored. It
             must already exist and should be empty.

MakePragma - Generates pragma files for standard system libraries which
can be used with all compilers:
 Options:
  FDPATH     This is the path where your system FD files are stored. The
             path must end in ':' or '/'.
  PRAGMAPATH This is the path where created files should be stored. It
             must already exists and should be empty.

MakeStubLib - Generates a stub link library for all standard system
```

```
libraries like amiga.lib (but with C++ name support). This library only
holds stubs and none of the additional amiga.lib stuff. The result is
a file called stubs.lib in the current directory. When you join this with
small.lib distributed on NDUK, you get a complete replacement for
amiga.lib and you need not to modify clib files for C++.
You need a Join command, which supports patterns matching like
"Aminet/util/sys/JoinReplace.lha" to get this script to work.
 Options:
  FDPATH     This is the path where your system FD files are stored. The
             path must end in ':' or '/'.
  CLIBPATH   This is the path where your system prototype files are
             stored. The path must end in ':' or '/'.

MakeProto - Generates proto files for standard system libraries which
can be used with all compilers:
 Options:
  FDPATH     This is the path where your system FD files are stored. The
             path must end in ':' or '/'.
  PROTOPATH  This is the path where created files should be stored. It
             must already exist and should be empty.

MakeUnit - Generates FPC Pascal compiler unit text files for standard
system libraries:
 Options:
  FDPATH     This is the path where your system FD files are stored. The
             path must end in ':' or '/'.
  CLIBPATH   This is the path where your system prototype files are
             stored. The path must end in ':' or '/'.
  UNITPATH   This is the path where created files should be stored. It
             must already exist and should be empty.

I added another Shell script called 'MakeStuff', which allows you to
generate all the needed files for a single library. This is mostly useful
for library programmers, who need to release include files for these
libraries.

It gets 3 arguments:
 FDFILE   The FD file, which normally is named 'xxx_lib.fd'.
 CLIBFILE The prototypes file, which normally is called 'xxx_protos.h'.
 DEST     The destination directory, which must already exist and should
          be empty.

In the destination path all needed directories are created, the FD and
prototypes file are copied and inline, lvo, pragma and proto files are
created. You only need to copy your library include files and the include
stuff is complete.

MakePPCStuff  - has the same arguments, creates needed directories, but does
not copy the clib and FD file. It produces the files needed to support PPC
PowerUP system.

MakeVBCC - has the same arguments as MakeStuff. It produces link libraries
for VBCC (68k, WOS, PPC).
```

1.29 Useful script interface

usefd2pragma is a shell script file which asks you some questions and
creates the related files afterwards. The command called is printed to
the window which appears so you can use it afterwards for batch processing.

This script allows use of nearly all possible options of fd2pragma.

1.30 Greetings, last words, and author's address

Everyone using this program should tell me in a brief message which options
he uses and if there are some problems. It took me many of hours to make
fd2pragma as powerful as it is now. So please tell me if you are using it!

This program is in the public domain. Use it as you want, but WITHOUT ANY
WARRANTY!

The program is compiled using SAS-C 6.58 and has additionally been tested
with MaxonC++, GCC, and StormC. Any other ISO-C compiler should be
sufficient as well.

Because fd2pragma is very complex, it may be that there are some errors
(even serious ones) in the code. So if you find one, please tell me!
I will also be glad if someone tells me what can be improved in the
program! I will add new options but there will never be a GUI because this
utility is for experts, and they do not need a GUI to create the files
needed :-) Besides, a GUI would really make a lot of work and increase the
file size greatly.

Please contact me at:


```
*********************************************************************
* snail-mail:                  * e-mail:                            *
*   Dirk Stoecker              *   stoecker@epost.de                *
*   Geschwister-Scholl-Str. 10 *   dirk@dstoecker.de                *
*   01877 Bischofswerda        * world wide web:                    *
*   GERMANY                    *   http://www.dstoecker.de/         *
* phone:                       * pgp key:                           *
*   GERMANY +49 (0)3594/706666 *   get from WWW pages or keyservers *
*********************************************************************
```


1.31 index

The entries given here link to the correct node entry, but not ↩
always to
correct line. This is because I often change the texts and the line
numbers would need to be updated every time.

prototypes
 R
registers
 S
Scripts

SECTION option

SFD file design

small data

SMALLDATA option

SORTED option

SPECIAL option

stub function
 T
tag function

tag-functions definition
 U
usefd2pragma

USESYSCALL option
 V
VBCC compiler
 W
WarpOS - WOS