FD2Pragma

COLLABORATORS			
	TITLE : FD2Pragma		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 7, 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!

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:
  - following pragma statements for certain C compilers: amicall, libcall,
    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 useable 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 support this normally
  - stub functions for Pascal compilers

Therefor only the FD file telling the library informations is needed. For
SPECIAL options 10-14 you may additionally supply the CLIB keyword giving
FD2Pragma the prototypes file in clib directory.

Special option 50 does the reverse to normal: convert pragma to FD!
```

1.3 options

```
You get the command template with FD2Pragma ? .

FROM=FDFILE/A,SPECIAL/N,TO/K,COMMENT/S,MODE/N,AMICALL/K,LIBCALL/K,
AMITAGS/K,LIBTAGS/K,EXTERNC/S,USESYSCALL/S,CLIB/K,PRIVATE/S,HEADER/K,
STORMFD/S:

In this position you may press <?> again and you get the following text!
Be carefull, because this text is longer than one normal high resolution
screen, so it is usefull to press a key in the middle of the text to stop
the output.

FDFILE:  the FD 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]
  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
  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
  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
  50 - FD file (source is a pragma file!)
TO:  the destination directory (self creation of filename) or
   the destination file
COMMENT: copy comments found in FD file
MODE:  SPECIAL 1-6,AMICALL,LIBCALL,AMITAGS,LIBTAGS,CSTUBS:
   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:
   1 - all functions, normal interface
   2 - only tag-functions, tagcall interface [default]
   3 - all functions, normal and tagcall interface
AMICALL: creates amicall pragmas
LIBCALL: creates libcall pragmas
AMITAGS: creates tagcall pragmas (amicall like method (StormC++))
LIBTAGS: creates tagcall pragmas (libcall like method (SAS C))
The last four need a string as argument. This string is used to set
a #if<given string> before the set method.
EXTERNC: add a #ifdef __cplusplus ... statement to pragma file
USESYSCALL: uses syscall pragma instead of libcall SysBase
```

```
CLIB:    name of the prototypes file in clib directory
PRIVATE: includes private declared functions
HEADER:  inserts given file into header of created file ("" is scan)
STORMFD: converts FD files of strange StormC++ format
```

1.4 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.
```

1.5 Option explanation of all user options

```
FDFILE is the always needed source file, which describes the library.

SPECIAL option:

(create a pragma file)
  1: Creates a pragma file for the Aztec compiler, what this means you
     see in the brackets above.
  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 useable for nearly all compilers.
     This is default, when no other mode is given.

(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 not able to call
     a library directly (result is ASM source code), 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++). Most time it is useful to set option
     MODE to 2. This options needs CLIB keyword for correct results.

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

 15: Creates STUB functions for Pascal compilers. The tagcall function
     names are ignored, as they cannot be used with Pascal. The result is
     readable assembler text. The code equals the one for C compilers, but
     the args are taken from stack in reversed order.

 16: same as 15, but produces link library.

(assembler LVO files)
 20: Creates lvo file for an assembler.
 21: Same as 20, but other name.
 22: Same as 20, but there are no XDEF statements in the resulting file.
 23: Same as 22, but other name.

(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 inline file for GNU-C and
     pragma/xxx_lib.h for all the others.
 FD2Pragma knows the correct library base structures for some libraries.
 All the other libraries get 'struct Library' as default.

(FD file)
```

```
 50: This creates a FD file! The option FDFILE has to be a pragma file
     here!

TO: Here you specify either the destination directory or the destination
    file!
  - If this argument is a directory, the internal names are used, but the
    file will be in the given directory.
  - If this argument is a filename (not existing or already existing), then
    the resulting file has the here given name!
```

1.6 Options for detailed control

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

MODE:
1) given with SPECIAL 1 to 6, AMITAGS, AMICALL, LIBTAGS, LIBCALL or CSTUBS:
   - Defines, which #ifdef ...\n#define ... statement is used in the pragma
     file. Option 1 is default.

2) given with SPECIAL 11 to 14:
   - Defines, which functions should be created. Option 2 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

EXTERNC: This options adds an #ifdef __cplusplus ... statement to the
pragma file. This options is useful for C compilers runing in C++ mode, but
it seems, that they do not really need this statement. Only useful with
SPECIAL option 1-6, 13 and 14.

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.

CLIB: Supply name of the prototypes file in clib directory. If this option
is given together with SPECIAL 11 to 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 CSTUBS and SPECIAL 13 and 14. If it is
not given you get a nearly useless result.

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

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 "" as filename, the
destination file (if already exists) will be scanned for an existing
header.

STORMFD: This option allows to convert FD files in strange StormC++ format.
It’s a FD file format defining the C tag-function name directly in FD file.
```

These files cannot be used with other FD scanners without changes.

1.7 Options for direct pragma generation

The now following options are for not recommended to be used. They are designed to be used without any SPECIAL option, but you also can give SPECIAL and any of the following options! In this case the corresponding settings of SPECIAL are overwritten, when they are in conflict.

```
AMICALL: creates amicall pragmas
--> #pragma amicall(IntuitionBase,0x294,SetGadgetAttrsA(a0,a1,a2,a3))

LIBCALL: creates libcall pragmas
--> #pragma libcall IntuitionBase SetGadgetAttrsA 294 BA9804

AMITAGS: creates tagcall pragmas (amicall like method (StormC++))
--> #pragma tagcall(IntuitionBase,0x294,SetGadgetAttrs(a0,a1,a2,a3))

LIBTAGS: creates tagcall pragmas (libcall like method (SAS C))
--> #pragma tagcall IntuitionBase SetGadgetAttrs 294 BA9804
```

These four functions need a string as argument. It is used to set a

```
  #if<given string>
```

before the data of that option. So it is possible to create a file like this:

```
--> FD2Pragma FDFILE xxx AMICALL " defined(__MAXON__) || defined(AZTEC_C)"
    LIBCALL "def __SASC"

#if defined(__MAXON__) || defined(AZTEC_C)
  /* do amicalls */
#endif
#ifdef __SASC
  /* do libcalls */
#endif
```

If you give "" as string, then no '#if<text>' statement will be added.

As you see, the text is added without space after '#if'. This gives you the ability to use also other '#if' clauses, than '#ifdef' (f.e. #ifndef). If needed, you have to add the space in the parameter text (" defined...").

1.8 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,

```
which were created with FD2Pragma. The system includes you get for example
on 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 fd2inline.
 Directories like 'pragmas' were deleted, when existing. Remain should only
 'clib', 'pragma', 'proto' and library specific directories (like 'dos',
 'exec', 'libraries' and 'utility').

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

1.9 linker libraries

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

The created link libraries are relatively big 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 I define a lot of texts (which are NOT copied to the
created executable program). These texts are for easier identification and
every function also gets different names:

  1) the normal asm name:  <name>   (f.e.  CopyMem)
  2) the normal C name:   _<name>   (f.e. _CopyMem)
  3) the normal C++ name:  <name>_<params>  (f.e.  CopyMem_PvPvUj)
  4) when a function parameter is STRPTR, a second C++ name is created

Forms 3 and 4 occur only, when you use CLIB keyword. With SPECIAL options
13 and 14 the number of strings is twice as much. 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 most time
is a lot smaller than the link library!

I think the code part of the link libraries is optimized as good as
possible. There may be some things making the code shorter, but this will
be in the area of 2 to 10 bytes for some functions. There is no need to
program hours and hours for 10 bytes. The stub functions of amiga.lib are
still a little bit better than mine, but amiga.lib only holds standard
system libraries.
```

1.10 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 file created with SPECIAL option 35 supports 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.

1.11 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 internally 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

Non-FD2Pragma names are:
 inline files for GCC     _INLINE_INTUITION_H
 clib files       CLIB_INTUITION_PROTOS_H
 other includes (path_name_extension) INTUITION_INTUITION_H
```

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

1.12 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. Use always 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 equal to the prototypes in directory clib, but with struct Library * as 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 always the CLIB keyword together with SPECIAL option 13 and 14 or the resulting files are nearly useless.

1.13 Which way the header scan works

Giving the HEADER option lets FD2Pragma insert the file (you have to give filename with HEADER option) at start of LVO/Pragma/Proto/stub file. When you pass "" as filename, FD2Pragma scans the destination file (if already existing) for a header and copies this header to the new file.

How is scanned:
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.

C and ASM files are scanned same way, so sometimes FD2Pragma may get a wrong header.

1.14 Way 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. Lets 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 above mentioned 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 there 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 tag-function. If the last argument equals one of the words "tags", "taglist" or "args", then the function has a tag-function named xxxTags or xxxArgs.

The are some exceptions for this rules (some dos.library and utility.library functions) which are handled automatically.

1.15 Words and phrases

stub, stub function:
A stub functions is a function, which converts between different interfaces. 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 call's 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.

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.

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. FD2Pragma does not produce these files, as they are really GCC specific and there is a fine tool (Aminet dev/gcc/fd2inline_bin.lha) to create them. But I suggest using the proto file you can create with SPECIAL 35 instead of the file you may produce with fd2inline, as the fd2inline proto works with GCC only.

.lib file, link library:
A link library is a file holding 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.

tag-functions:
C allows to have functions getting a variable number of parameters, everytime they are called. These varargs functions have in there prototypes "..." at the end (f.e. 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 no tags (f.e. Printf), but it
 expresses good, what is meant.

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 till now and called the pragma files directly, but this
 makes it harder to switch to another compiler. So I now use proto always.

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 parameter. These files are needed
 by FD2Pragma to create correct data with some options. If not given, all
 variables will be of type ULONG, which seems not to be the best.
```

1.16 Known bugs and problems

```
- FD2Pragma handles function pointers given as arguments incorrect.
  (example: RawDoFmt() function.). This error only occurs, when using CLIB
  option. I had no time to do this, because it is not the easiest stuff.
  If really needed, please tell me!
  If you use SPECIAL option 13 and 14 you have to correct the corresponding
  data in the .h file and to add extern "C" or extern "ASM" to that name
  (in C++ mode).

- cia_lib.fd conversion fails with no ##basename error. This is wanted by
  Amiga OS programmers to allow passing the library base as 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. Using option
  13 or 14 may bring you a valid link file to use with clib file.

- mathieeedoubtrans_lib.fd and mathieeedoubbas_lib.fd both use 2 registers
  for one double value. FD2Pragma creates files for these, but I cannot say
  if they are valid.

- 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.

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

- When there are no tag-functions to be created, FD2Pragma does not stop
  its work, but produces empty files (may conatin header informations).
```

```
- Four dos.library functions are not created, as they are not specified in
  FD file:
    AllocDosObjectTagList, CreateNewProcTagList, NewLoadSegTagList,
    System
  You either may add them manually or use the corresponding functions
    AllocDosObject, CreateNewProc, NewLoadSeg, SystemTagList.

* Automatic created files may not always be fully correct (may happen
* seldom, but sometimes). When you find such a condition, please contact me
* and when useful and possible I will include a fix in the program.
```

1.17 Greetings, last words and authors address

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

I want to send greetings also to the author making version 2.0 of this
utility. Although this version of FD2Pragma has not much same with version
2.0 it was a big help. The source code for version 2.0 is included in the
archive and the authors address is also stated in the source file of the
current version!
          Thanks Jochen for your great work!

Because FD2Pragma is very complex, it may be that there are some errors
(maybe also 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 a GUI will never come, because this
utility is for experts, and they do not need a GUI for creating the needed
files. :-)

Please contact me:
```

```
***************************************************************************
* snail-mail:                    * e-mail:                                *
*   Dirk Stoecker                *   stoecker@rcs.urz.tu-dresden.de       *
*   Geschwister-Scholl-Str. 10 *   stoecker@amigaworld.com              *
*   01877 Bischofswerda         * world wide web:                        *
*   GERMANY                      *   http://home.pages.de/~Gremlin/       *
* phone:                         * pgp key:                               *
*   GERMANY +49 (0)3594/706666 *   get with finger or from WWW pages    *
***************************************************************************
```