Dillo.h

```
#ifndef DILLO_H
#define DILLO_H

/*
**  dillo.h - Header file for armadillo.library code modules
**              Copyright 1993, Commodore-Amiga Inc.
**            All Rights Reserved.
**            Written by John Wiederhirn
**
**  This is the header file for the armadillo.library code modules.  It
**  just gives a basic structure definition.
**
*/

struct Armadillo
{
    UBYTE   name[32];
    ULONG   weight;
    BOOL    flat;
};

#endif /* DILLO_H */
```

Dillo_lib.c

```
/*
**  dillo_lib.c
**
**  Contains the __UserLibInit() and __UserLibCleanup() routines for
**  the armadillo.library example shared library.
**
**    Copyright 1993, Commodore-Amiga Inc. All Rights Reserved.
**  Written by John Wiederhirn
**
*/

#include    <exec/types.h>
#include    <clib/exec_protos.h>
#include    <pragmas/exec_pragmas.h>

/* These prototypes are just to keep the compiler happy since we don’t
** want them in the distributable proto file.
*/

int __saveds __UserLibInit(void);
void __saveds __UserLibCleanup(void);

/* Following global item (UtilityBase) is created so our library can make
** utility.library calls.  Technically putting it in the near section is a waste
** of memory, but for this example, it serves its purpose.  Note also that we
** don’t actually MAKE any Utility calls, but we COULD.
*/

struct Library *UtilityBase = NULL;
struct Library *SysBase = NULL;

/*
*/

int __saveds
__UserLibInit( void )
{
    int retval = 1;

    SysBase = (*((void **)4));

    /* Here we attempt to open Utility library.  Not a particularly good
    ** example, but it gets the point across.  If exec.library could not
    ** be opened (say for a second it couldn’t), then __UserLibInit would
    ** return 1 indicating failure (where a return of 0 means success).
    */

    if (UtilityBase = OpenLibrary( "utility.library", 0L ))
        retval = 0;

    return( retval );
}


/* About as basic a routine as you can get, this routine cleans up the library
** by providing a matching CloseLibrary of the Utility library base we opened
** in the __UserLibInit() routine.
*/

void __saveds
__UserLibCleanup( void )
{
    CloseLibrary( UtilityBase );
}
```

Dillo_protos.h

```
#ifndef DILLO_PROTOS_H
#define DILLO_PROTOS_H

/*
**  dillo_protos.h - Header file for armadillo.library code modules
**                     Copyright 1993, Commodore-Amiga Inc.
**                   All Rights Reserved.
**                   Generated by SAS/C 6.2 (Modified by JFW)
**
**  Function prototypes for all of the functions in the library.  Used
**  to keep SAS/C docile.
**
*/

struct Armadillo * __asm __saveds CreateArmadillo(void);
void __asm __saveds DeleteArmadillo(register __a0 struct Armadillo * );
BOOL __asm __saveds NameArmadillo(register __a0 struct Armadillo * ,
                                  register __a1 STRPTR ,
                                  register __d0 ULONG );
BOOL __asm __saveds FillArmadillo(register __a0 struct Armadillo * ,
                                  register __d0 ULONG );
BOOL __asm __saveds FlattenArmadillo(register __a0 struct Armadillo * ,
                                     register __d0 BOOL );
BOOL __asm __saveds DilloFlat(register __a0 struct Armadillo * );
ULONG __asm __saveds DilloWeight(register __a0 struct Armadillo * );
BOOL __asm __saveds DilloName(register __a0 struct Armadillo * ,
                              register __a1 STRPTR ,
                              register __d0 ULONG );
void __asm __saveds ClearDillo(register __a0 struct Armadillo * );
ULONG __asm __saveds DilloBirths(void);

#endif /* DILLO_PROTOS_H */
```

Dillo.c

```
/*
**  dillo.c
**
**  A code module of armadillo.library, which implements the main library
**  functions.  Strictly do-nothing code for example.
**
**    Copyright 1993, Commodore-Amiga Inc. All Rights Reserved.
**  Written by John Wiederhirn
**
*/

#include    <exec/types.h>
#include    <exec/memory.h>
#include    <clib/exec_protos.h>
#include    <pragmas/exec_pragmas.h>

#include    "dillo.h"
#include    "dillo_protos.h"

/* The next prototype is for a static function which is only available
** to code inside the library itself.
*/

static void ClearDilloName(struct Armadillo * );


/* The following global data item becomes part of the near data section
** for each library client.  Since this library is designed to give a
** different library base to each client, this data item is unique per
** client.
**
** It holds the number of armadillos a given client has open at once.
*/

ULONG TotalDillos = 0L;

/* In contrast to the previous global data item, the following goes in
** the far data section and is global to all library clients.  Read access
** doesn’t need arbitration, but write access needs a semaphor or use of
** a Forbid()/Permit() pair (see CreateDillo() below).
**
** It holds the number of times CreateDillo has been called overall.
*/

ULONG __far TotalDillosCreated = 0L;

/* This routine just allocates a ‘struct Armadillo’, and increments
** the number of armadillos this client has by one.  If the allocation
** cannot be done, this routine returns NULL.
*/

struct Armadillo * __saveds __asm
LIBCreateArmadillo( register __a6 struct Library *DilloBase )
{
    struct Armadillo *newdillo = NULL;

    if ( newdillo = AllocMem( sizeof(struct Armadillo), MEMF_CLEAR ))
    {
        /* Armadillo allocated, so increment number of dillos.
        ** Note that to reference the client-unique data takes no
        ** special coding.
        */

        TotalDillos++;

        /* Since we’ve also added to the overall number created, we
        ** need to also update the TotalDillosCreated variable in
        ** the far data section.  That means a Forbid() and Permit()
        ** around the action (which MUST complete).
        */

        Forbid();
        TotalDillosCreated++;
        Permit();

    }

    /* And return either the address of the new armadillo, or else
    ** return NULL if the allocation failed.
    */

    return( newdillo );
}


/* This function wipes an existing Armadillo structure out of existance
** and decrements the number of Armadillos for this client.  Note that the
** number of Armadillos created overall does not go down.
*/

VOID __saveds __asm
LIBDeleteArmadillo( register __a0 struct Armadillo *dillo,
                    register __a6 struct Library *DilloBase )
{
    /* This routine is ‘‘safe’’ in that it can handle being given a NULL
    ** pointer (in which case it does nothing).
    */

    if ( dillo )
    {
        /* We do indeed appear to have an armadillo on our hands
        ** so we decrement the overall count and deallocate the
        ** memory it uses.
        */

        TotalDillos--;

        FreeMem( dillo, sizeof( struct Armadillo ));
    }

    return;
}

/* This transfers the contents of a string up to 32 characters long into
** the name buffer of an Armadillo.  Any attempt to transfer more than 32
** characters gets truncated to 32 characters.  Returns FALSE if dillo
** was a NULL pointer, the pointer to the string was NULL, or the length
** of the transfer was to be 0L.
*/

BOOL __saveds __asm
LIBNameArmadillo( register __a0 struct Armadillo *dillo,
                  register __a1 STRPTR dname,
                  register __d0 ULONG len,
                  register __a6 struct Library *DilloBase )
{
    BOOL retval = FALSE;

    /* This routine is ‘‘safe’’ in that it can handle being given a NULL
    ** pointer (in which case it does nothing).
    */

    if ( dillo && dname && len )
    {
        CopyMem( (APTR) dname, (APTR) &(dillo->name), ((len>31L)?32L:len) );
        retval = TRUE;
    }

    return( retval );
}


/* Assigns a value to the weight field of an Armadillo structure.  It
** returns NULL if a NULL pointer is passed in or amt was 0L.
*/

BOOL __saveds __asm
LIBFillArmadillo( register __a0 struct Armadillo *dillo,
                  register __d0 ULONG amt,
                  register __a6 struct Library *DilloBase )
{
    BOOL retval = FALSE;

    /* This routine is ‘‘safe’’ in that it can handle being given a NULL
    ** pointer (in which case it does nothing).
    */

    if ( dillo && amt )
    {
        dillo->weight = amt;
        retval = TRUE;
    }

    return( retval );
}


/* In homage to the Texas state animal, the roadkill armadillo, this function
** sets whether a given Armadillo is flattened or not.  Returns NULL if a
** NULL pointer was passed as the Armadillo structure.
*/

BOOL __saveds __asm
LIBFlattenArmadillo( register __a0 struct Armadillo *dillo,
                     register __d0 BOOL flatd,
                     register __a6 struct Library *DilloBase )
{
    BOOL retval = FALSE;

    /* This routine is ‘‘safe’’ in that it can handle being given a NULL
    ** pointer (in which case it does nothing).
    */

    if ( dillo )
    {
        dillo->flat = flatd;
        retval = TRUE;
    }

    return( retval );
}


/* Returns whether or not the Armadillo has been flattened.  If a NULL
** pointer is passed in, this function returns FALSE (not really distinct).
*/

BOOL __saveds __asm
LIBDilloFlat( register __a0 struct Armadillo *dillo,
              register __a6 struct Library *DilloBase )
{
    BOOL retval = FALSE;

    /* This routine is ‘‘safe’’ in that it can handle being given a NULL
    ** pointer (in which case it returns FALSE).
    */

    if ( dillo )
    {
        retval = dillo->flat;
    }

    return( retval );
}


/* Returns the weight of a given Armadillo or 0L if a NULL pointer
** is passed instead of an Armadillo (no pointer == no weight ).
*/

ULONG __saveds __asm
LIBDilloWeight( register __a0 struct Armadillo *dillo,
                register __a6 struct Library *DilloBase )
{
    ULONG retval = 0L;

    /* This routine is ‘‘safe’’ in that it can handle being given a NULL
    ** pointer (in which case it returns 0L).
    */

    if ( dillo )
    {
        retval = dillo->weight;
    }

    return( retval );
}


/* This function copies the name of an Armadillo into the caller-specified
** buffer (which MUST be at least 32 characters in length).  A NULL pointer
** for the Armadillo, buffer or len will get a FALSE return, otherwise a
** return of TRUE if the transfer occurred.
*/

BOOL __saveds __asm
LIBDilloName( register __a0 struct Armadillo *dillo,
              register __a1 STRPTR buf,
              register __d0 ULONG len,
              register __a6 struct Library *DilloBase )
{
    BOOL retval = FALSE;

    /* This routine is ‘‘safe’’ in that it can handle being given a NULL
    ** pointer (in which case it does nothing).
    */

    if ( dillo && buf && len)
    {
        CopyMem( (APTR) &(dillo->name), (APTR) buf, ((len>31L)?32L:len) );
        retval = TRUE;
    }

    return( retval );
}


/* Following are non-public but externally-accessible entry points. */

/* This routine clears out the contents of an Armadillo.  It also is an
** example of using a non-public non-ext.-accessible routine in a shared
** library.
*/

VOID __saveds __asm
LIBClearDillo( register __a0 struct Armadillo *dillo,
               register __a6 struct Library *DilloBase )
{
    /* This routine is ‘‘safe’’ in that it can handle being given a NULL
    ** pointer (in which case it does nothing).
    */

    if ( dillo )
    {
        dillo->flat = FALSE;
        dillo->weight = 0L;
        ClearDilloName( dillo );
    }
}


/* This routine does an "unprotected" query (legal, since the access is
** read-only) of the TotalDillosCreated variable in the far data section.
*/

ULONG __saveds __asm
LIBDilloBirths( register __a6 struct Library *DilloBase )
{
    return( TotalDillosCreated );
}


/* Following call is a non-public non-externally-accessible function */

/* This function is callable ONLY from within this module.  It clears out
** the name buffer for a given Armadillo.
*/

static VOID
ClearDilloName( struct Armadillo *dillo )
{
    int i;

    /* This routine is NOT ’safe’.  Params must be pre-checked. */

    for(i=0;i<31;i++)
        dillo->name[i] = ’\0’;

}
```

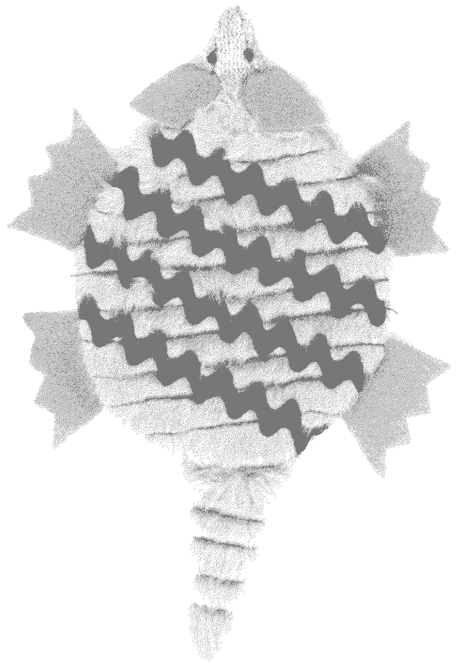
Makefile

```
##
##  armadillo.library makefile
##
##  This is a more-or-less generic makefile, which is currently set
##  to compile armadillo.library but which can easily be changed to
##  use your own files...

MODNAME=            armadillo
VERSION=            37
REVISION=           0

LIBFILE=            $(MODNAME).library

FD_CONV=            SC:C/FD2PRAGMA
FD_FILE=            $(MODNAME)_lib.fd
PRAGMA_FILE=        $(MODNAME)_pragmas.h

C_COMPILER=         SC:C/SC
C_OPTS=             STREQ STRMER NOSTKCHK LIBCODE

LINKER=             SC:C/SLINK

C_SOURCES=          dillo_lib.c dillo.c

OBJECTS=            dillo_lib.o dillo.o

LIBENT=             LIB:libent.o
LIBINIT=            LIB:libinitr.o
LIBPREFIX=          _LIB

##############################################################################
# Build the library...

$(LIBFILE): $(OBJECTS) $(LIBS) $(PRAGMA_FILE)

    $(LINKER) WITH <<
TO $(LIBFILE)
FROM $(LIBENT) $(LIBINIT) $(OBJECTS)
LIBFD $(FD_FILE)
LIBPREFIX $(LIBPREFIX)
LIBVERSION $(VERSION)
LIBREVISION $(REVISION)
<

$(PRAGMA_FILE): $(FD_FILE)

##############################################################################
# Default rules...
#
.c.o:
    $(C_COMPILER) $(C_OPTS) $*.c

.fd.h:
    $(FD_CONV) $(FD_FILE) $(PRAGMA_FILE)

##############################################################################
# Delete all object files
#
clean:
    @Delete $(OBJECTS)
    @Delete $(LIBFILE)(|.info)
    @Delete $(MODNAME).map(|.info)

##############################################################################
# Load the new library into the system
#
reload:
    @copy $(LIBFILE) LIBS:
    @copy $(FD_FILE) FD:
    @flushlibs
    @version $(LIBFILE)
```

Armadillo_lib.fd

```
*
* armadillo.library - Sample SAS/C run-time library
*
*   Copyright Commodore-Amiga, Inc.
* All Rights Reserved.
*
##base _DilloBase
##bias 30
##public
CreateArmadillo()()
DeleteArmadillo(dillo)(A0)
NameArmadillo(dillo,name,len)(A0/A1,D0)
FillArmadillo(dillo,weight)(A0,D0)
FlattenArmadillo(dillo,flat)(A0,D0)
DilloFlat(dillo)(A0)
DilloWeight(dillo)(A0)
DilloName(dillo,buf,len)(A0/A1,D0)
ClearDillo(dillo)(A0)
DilloBirths()()
##end
```

Armadillo



Dillo_test.c

```c
;/* dillo_test.c - Execute me to compile me with SAS C 6.2
sc data=far nominc strmer streq nostkchk saveds ign=73 dillo_test.c
slink FROM LIB:c.o,dillo_test.o TO dillo_test LIB LIB:SC.lib,LIB:Amiga.lib
quit
*/

/*
**  dillo_test.c
**
**  This is the sample program to test the functions inside
**  armadillo.library.  It creates a small array of Armadillos
**  and after assigning various values, prints out their status.
**
*/

#include    <exec/types.h>
#include    <exec/libraries.h>

#include    <clib/exec_protos.h>
#include    <pragmas/exec_pragmas.h>

#include    <stdio.h>

#include    "dillo_protos.h"
#include    "armadillo_pragmas.h"

/* Global data for the test program */

struct Library *DilloBase;      /* armadillo.library library base */

APTR dillo[5];                  /* Array of generic pointers to     */
                                /* armadillos, since programs don't */
                                /* need to know what the insides of */
                                /* an armadillo look like.          */

STRPTR names[5] = { "Alex", "Bob", "Chris", "Daniel", "Eustace" };
ULONG namlen[5] = { 5, 4, 6, 7, 8 };
ULONG weight[5] = { 18, 20, 19, 17, 354 };

void
main(void)
{
    BOOL okay = FALSE;
    ULONG i = 0L;

    if (DilloBase = OpenLibrary("armadillo.library",0))
    {

        for (i=0;i<5;i++)
        {
            if (dillo[i] = CreateArmadillo())
            {
                if (okay = NameArmadillo( dillo[i], names[i], namlen[i] ))
                {
                    printf("Armadillo %ld named %s.\n",i,names[i]);
                }
                else
                {
                    printf("Armadillo %ld naming failure, it's anonymous.\n",i);
                }

                if (okay = FillArmadillo( dillo[i], weight[i] ))
                {
                    printf("Armadillo %ld weighs %ld.\n",i,weight[i]);
                }
                else
                {
                    printf("Armadillo %ld fill failure, it's dieting.\n",i);
                }

                if (i>2)
                {
                    if (okay = FlattenArmadillo( dillo[i], TRUE ))
                    {
                        printf("Armadillo %ld had a slight mishap.\n",i);
                    }
                }
            }
            else
            {
                printf("Couldn't create Armadillo %ld\n",i);
            }
        }

        /* Okay, all the armadillos are created (hopefully) and */
        /* so as proof of concept and to test the data access   */
        /* functions, now the program shows the status of each  */
        /* of the armadillos.                                   */

        printf("\nArmadillo Status Report\n");
        printf("-----------------------\n");

        for(i=0;i<5;i++)
        {
            UBYTE namebuf[33];

            printf("Armadillo #%ld\n",i);
            if (DilloName(dillo[i],(STRPTR)&namebuf,32))
            {
                printf("  Name   = \"%s\"\n",namebuf);
            }
            else
            {
                printf("  Name is invalid.\n");
            }
            printf("  Weight = %ld pounds\n\n",DilloWeight(dillo[i]));
            printf("  Dillo is %s\n",
                   (DilloFlat(dillo[i])?"flat":"lucky"));
            printf("-----\n");
        }

        printf("Total Dillos created: %ld\n\n",DilloBirths());

        /* Now that the armadillos have been tested, we can */
        /* delete them with gleeful abandon.                */

        for(i=0;i<5;i++)
        {
            DeleteArmadillo( dillo[i] );
        }

        /* We're done, so close the library... */
        CloseLibrary(DilloBase);
    }
    else
    {
        printf("Couldn't open armadillo.library!\n");
    }
}
```