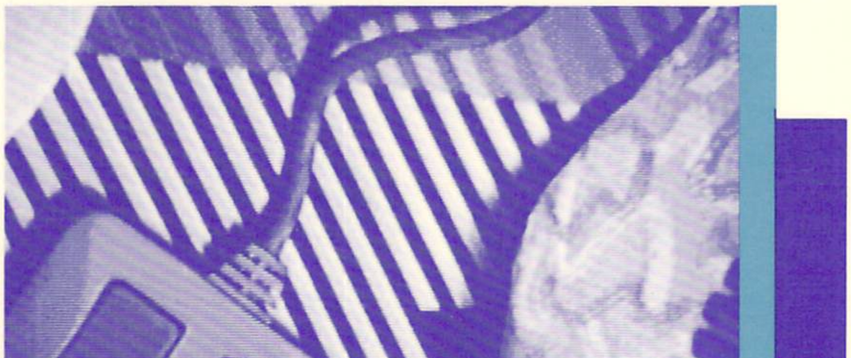
AMIGA™ ROM Kernel Reference Manual: Exec

Commodore Business Machines, Inc.



Amiga

ROM Kernel Reference Manual

Exec

Amiga

ROM Kernel Reference Manual

Exec

Commodore Business Machines, Inc.

Amiga Technical Reference Series

Addison-Wesley Publishing Company, Inc.

Reading, Massachusetts   Menlo Park, California   Don Mills, Ontario
Wokingham, England   Amsterdam   Bonn   Sydney   Singapore
Tokyo   Madrid   Bogotá   Santiago   San Juan

Authors: Carl Sassenrath, Rob Peck, and Susan Deyl
Program examples by Carl Sassenrath

Library of Congress Cataloging-in-Publication Data

Amiga ROM kernel reference manual.

(Amiga technical reference series)
Includes index.
1. Amiga (Computer)–Programming. 2. Read-only
storage. I.. Commodore Business Machines.
QA76.8.A177A655 1986 005.4'46 86-10887
ISBN 0-201-11099-7

Printed from camera-ready mechanicals supplied by the authors.

DEFGHIJ-BA-98987

Fourth Printing, January 1987

PREFACE

System Software Architecture

The Amiga kernel consists of a number of system modules, some of which reside permanently in the protected *kickstart* memory and others that are loaded as needed from the system disk. Figure P-1 illustrates how the various modules interact with one another. At the top of the hierarchy are Workbench and the Command Line Interface (CLI), the user-visible portions of the system. Workbench uses Intuition to produce its displays and AmigaDOS to interact with the filing system. Intuition, in turn, uses the input device to retrieve its input and the graphics and layers library routines to produce its output.

AmigaDOS controls processes and maintains the filing system and is in turn built on Exec, which manages tasks, task switching, interrupt scheduling, message-passing, I/O, and many other functions.

At the lowest level of the hierarchy is the Amiga hardware itself. Just above the hardware are the modules that control the hardware directly. Exec controls the 68000, scheduling its time among tasks and maintaining its interrupt vectors, among other things. The trackdisk device is the lowest-level interface to the disk hardware, performing disk-head movement and raw disk I/O. The keyboard and gameport devices handle the keyboard and gameport hardware, queuing up input events for the input device to process. The audio device, serial device, and parallel device handle their respective hardware. Finally, the routines in the graphics library handle the interface to the graphics hardware.

Programming

The functions of the kernel were designed to be accessed from any language that follows the Amiga's standard interface conventions. These conventions define the proper naming of symbols, the correct usage of processor registers, and the format of public data structures.

REGISTER CONVENTIONS

All system functions follow a simple set of register conventions. The conventions apply when any system function is called; programmers are encouraged to use the same conventions in their own code.

The registers D0, D1, A0, and A1 are always scratch; they are free to be modified at any time. A function may use these registers without first saving their previous contents. The values of all other data and address registers must first be preserved. If any of these registers are used by a function, their contents must be saved and restored appropriately.

If assembly code is used, function parameters may be passed in registers. The conventions in the preceding paragraphs apply to this use of registers as well. Parameters passed in D0, D1, A0, or A1 may be destroyed. All other registers must be preserved.

If a function returns a result, it is passed back to the caller in D0. If a function returns more than one result, the primary result is returned in D0 and all other results are returned by accessing reference parameters.

The A6 register has a special use within the system, and it may not be used as a parameter to system functions. It is normally used as a pointer to the base of a function vector table. All kernel functions are accessed by jumping to an address relative to this base.

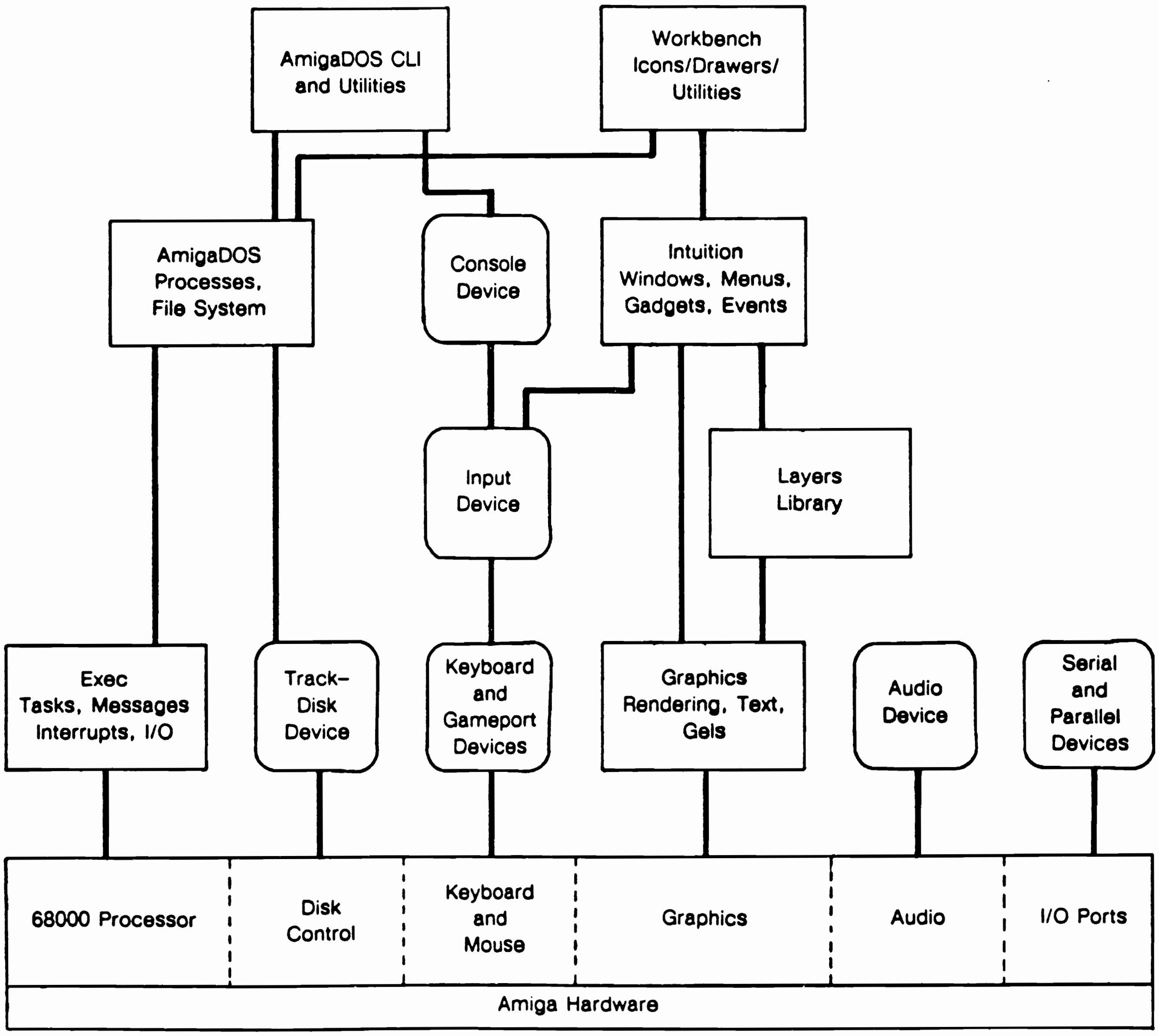

AmigaDOS CLI and Utilities
Workbench Icons/Drawers/ Utilities
AmigaDOS Processes, File System
Console Device
Intuition Windows, Menus, Gadgets, Events
Input Device
Layers Library
Exec Tasks, Messages Interrupts, I/O
Track-Disk Device
Keyboard and Gameport Devices
Graphics Rendering, Text, Gels
Audio Device
Serial and Parallel Devices
68000 Processor
Disk Control
Keyboard and Mouse
Graphics
Audio
I/O Ports
Amiga Hardware

Figure P-1: Amiga System Software Modules

DATA STRUCTURES

The naming, format, and initial values of public data structures must also be consistent The conventions are quite simple and are summarized below.

1. All non-byte fields must be word-aligned. This may require that certain fields be padded with an extra byte.

2. All address pointers should be 32 bits (not 24 bits) in size. The upper byte must never be used for data.

3. Fields that are not defined to contain particular initial values must be initialized to zero. This includes pointer fields

4. All reserved fields must be initialized to zero (for future compatibility).

5. Data structures to be accessed by custom hardware must not be allocated on a program stack.

6. Public data structures (such as a task control structure) must not be allocated on a program stack.

7. When data structures are dynamically allocated, conventions 3 and 4 above can be satisfied by specifying that the structure is to be cleared upon allocation.

OTHER PRACTICES

A few other general programming practices should be noted.

1. Never use absolute addresses. All hardware registers and special addresses have symbolic names (see the include files and *amiga.lib*).

2. Because this is a multitasking system, programs must never directly modify the processor exception vectors (including traps) or the processor priority level.

3. Do not assume that programs can access hardware resources directly. Most hardware is controlled by system software that will not respond well to interference. Shared hardware requires programs to use the proper sharing protocols.

4. Do not access shared data structures directly without the proper mutual exclusion. Remember, it is a multitasking system and other tasks may also be accessing the same structures.

5. Most system functions require a particular execution environment. For example, DOS functions can be executed only from within a process; execution from within a task is not sufficient. As another example, most kernel functions can be executed from within tasks, but cannot be executed from within interrupts.

6. The system does not monitor the size of a program stack. Take care that your programs do not cause it to overflow.

7. Tasks always execute in the 68000 processor user mode. Supervisor mode is reserved for interrupts, traps, and task dispatching. Take extreme care if your code executes in supervisor mode. Exceptions while in supervisor mode are deadly.

8. Do not disable interrupts or multitasking for long periods of time.

9. Assembly code functions that return a result do not necessarily affect the processor condition codes. By convention, the caller must test the returned value before acting on a condition code. This is usually done with a **TST** or **MOVE** instruction. Do not trust the condition codes returned by system functions.

68010 AND 68020 COMPATIBILITY

If you wish your code to be upwardly compatible with the 68010/68020 processors, you must avoid certain instructions and you must not make assumptions about the format of the supervisor stack frame. In particular, the **MOVE SR,<ea>** instruction is a privileged instruction on the 68010 and 68020. If you want your code to work correctly on all 680x0 processors, you should use the **GetCC()** function instead (see the Exec library function descriptions in the appendixes to *Amiga ROM Kernel Reference Manual: Libraries and Devices*).

USING AMIGA EXEC FUNCTIONS

The following guidelines will be helpful when you are trying to determine which functions may be run from within a task or from within interrupt code, when to forbid or permit task switching, and when to disable or enable interrupts.

Functions That Tasks Can Perform

Amiga system software distinguishes between tasks and processes. Figure P-1 illustrated this difference. Specifically, the information in a task control block is a subset of the information contained in a process control block. Consequently, any functions that expect to use process control information will not function correctly if provided with a pointer to a task. Generally speaking, tasks can perform any function that is described in this manual. A task cannot, however, perform any function that is related to AmigaDOS (such as **printf**, **Read**, **Write**, and so on). If you want a task to perform DOS-related functions, you should arrange for the task to send a message to a "process," which in turn can perform the function (filling a buffer that is passed to the task, for example) and signal that the job has been done. The alternative is to use the DOS function **CreateProc()** instead of the Exec support function **CreateTask()** for tasks that you spawn yourself. A process can call all functions, including DOS functions.

More information about tasks can be found in the "Tasks" chapter.

Functions That Interrupt Code Can Perform

The following Exec functions can be safely performed during interrupts:

Alert()	**FindPort()**
Disable()	**FindTask()**
Cause()	**PutMsg()**
Enable()	**ReplyMsg()**
FindName()	**Signal()**

In addition, if you are manipulating your own list structures during interrupt code, you can also use the following functions:

AddHead()
AddTail()
Enqueue()
RemHead()
RemTail()

General Information about Synchronization

The system functions **Enable()** and **Disable()** are provided to enable and disable interrupts. The system functions **Forbid()** and **Permit()** disallow or allow task switching. You need only determine what you are trying to synchronize with before deciding if you must wrap an **Enable()/Disable()** pair around a function call, use **Forbid()/Permit()**, or simply allow the system to interrupt or switch tasks at its whim.

If you are trying to modify a data structure common to two tasks, you must assure that your access to these structures is consistent. One method is to put **Forbid()/Permit()** around anything that modifies (or reads) that structure. This makes the function atomic; that is, the structure is stable and consistent after each full operation by either task. If you are trying to synchronize with something that might happen as a result of interrupt code (for example, Exec data structures), you put **Disable()/Enable()** around any of your own operations that might interact with such operations. There are other methods (sending messages, using semaphores, and so on), but they are somewhat more involved.

Note that if you are trying to read the contents of a data structure while it is being changed, it is possible to generate an address error that will be sensed by the 68000, causing an exception. This is caused by reading a pointer that is supposed to point to where the data is located. If the pointer value is no longer valid, it may point to a nonexistent memory location that, when read, causes an exception.

Contents of This Manual

This manual describes the functions of Amiga's multi-tasking executive (Exec). For information about the graphics support routines (including text and animation) and the I/O devices, see *Amiga ROM Kernel Manual: Libraries and Devices.* Also included in that volume are the Workbench, which is an environment for running programs, and the floating point mathematics library.

The discussion of the data structures and routines in this manual is reinforced through numerous C-language examples. The examples are kept as simple as possible. Whenever possible, each example demonstrates a single function. Where appropriate, there are complete sample programs.

Boldface type is used for the names of functions, data structures, macros, and variables. System header files and other system file names are shown in italics.

In code examples that show data structures and pointers, this book adheres to the following naming conventions. For example, the name **node** refers to an instance of a **Node** and **ln** refers to a pointer to a **Node**.

For more information, see also *Amiga Intuition Reference Manual*, *AmigaDOS User's Manual*, *AmigaDOS Developer's Manual*, and *AmigaDOS Technical Reference Manual.*

Contents

Amiga

ROM Kernel Reference Manual

Exec

Chapter 1

LISTS AND QUEUES

A thorough understanding of the basic elements of Exec lists and queues is necessary to write programs that deal properly with Exec. Subjects related to lists and queues include the node structure of lists, the linkage and initialization of list structures, and the list support functions and macros. Queues and priority sorted lists, which are achieved through the use of the list functions applied in a certain order, are also important.

Introduction

The Amiga system software operates in a highly dynamic environment of control data structures. An early design goal of Exec was to keep the system flexible and open-ended by not creating artificial boundaries on the number of system structures used. Rather than using static sized system tables, Exec uses dynamically created structures that are attached to the system as needed. This concept is central to the design of Exec.

Exec uses lists to maintain its internal database of system structures. Tasks, interrupts, libraries, devices, messages, I/O requests, and all other Exec data structures are supported and serviced through the consistent application of Exec's list mechanism. Lists have a common data structure, and a common set of functions is used for manipulating them. Because all of these structures are treated in a similar manner, only a small number of list handling functions need be supported by Exec.

List Structure

A list is composed of a *header* and a chain of linked elements called *nodes*. The header maintains memory pointers to the first and last nodes of the linked chain of nodes. The address of the header serves as the handle to the entire list. When referring to a list, you refer to the address of its header. In addition, the header specifies the data type of the nodes in a list. Node data typing will be discussed later.

NODE STRUCTURE

A node is divided into two parts: list linkage and node content. The linkage part contains memory pointers to the node's successor and predecessor nodes, the node data type, and the node priority. The content part stores the actual data structure of interest. As a C language structure, the linkage part of a node is defined as follows:

```
struct Node {
        struct Node     *ln_Succ;
        struct Node     *ln_Pred;
        UBYTE           ln_Type;
        BYTE            ln_Pri;
        char            *ln_Name;
};
```

where

ln_Succ

points to the next node in the list (successor),

ln_Pred

points to the previous node in the list (predecessor),

ln_Type

defines the type of the node,

ln_Pri

specifies the priority of the node, and

ln_Name

points to a printable name for the node.

As usual, **node** refers to an instance of a node, and **ln** is a pointer to a node.

The Exec **Interrupt** structure, a complete node, is defined as follows:

```
struct Interrupt {
 struct Node         is_Node;
 APTR                is_Data;
 VOID                (*is_Code)();
};
```

Here the **is_Data** and **is_Code** fields represent the useful content of the node.

NODE INITIALIZATION

Before you link a node into a list, you should initialize it. The initialization consists of setting the **ln_Type**, **ln_Pri**, and **ln_Name** fields to their appropriate values. The **ln_Succ** and **ln_Pred** fields do not require initialization. The **ln_Type** field contains the data type of the node. This indicates to Exec (and other interested subsystems) the type, and hence the structure, of the content portion of the node. Some of the standard system types are defined in the *exec/nodes.i* and *exec/nodes.h* include files. Some examples of standard system types are **NT_TASK**, **NT_INTERRUPT**, **NT_DEVICE**, and **NT_MSGPORT**. These are defined in *exec/nodes.h*.

The **ln_Pri** field uses a signed numerical value ranging from -128 to +127 to indicate the priority of the node relative to other nodes in the same list. Higher-priority nodes have more positive values; for example, 127 is the highest priority, zero is nominal priority, and -128 is the lowest priority. Some Exec lists are kept sorted by priority order. In such lists, the highest-priority node is at the head of the list, and the lowest-priority node is at the tail of the list. For most Exec node types, priority is not used. In such cases it is a good practice to initialize the priority field to zero.

The **ln_Name** field is a pointer to a null-terminated string of characters. Node names are used mostly to bind symbolic names to actual nodes. They are also useful for debugging purposes. It is always a good idea to provide every node with a name.

Here is a C example showing how you might initialize a node called **myInt**, which is an instance of the interrupt structure defined above:

```
struct Interrupt myInt;
myInt.ln_Type = NT_INTERRUPT;
myInt.ln_Pri = 20;
myInt.ln_Name = "sample.interrupt"
```

HEADER STRUCTURE

As mentioned earlier, the header maintains memory pointers to the first and last nodes of the linked chain of nodes. This header also serves as a handle for referencing the entire list.

Here is the C-structure of a list header:

```
struct List {
      struct Node   *lh_Head;
      struct Node   *lh_Tail;
      struct Node   *lh_TailPred;
      UBYTE         lh_Type;
      UBYTE         lh_pad;
};
```

where:

lh_Head

points to the first node in the list,

lh_Tail

is always zero,

lh_TailPred

points to the last node in the list,

lh_Type

defines the type of nodes within the list, and

lh_pad

is merely a structure alignment byte (not used).

As usual, **list** refers to an actual instance of a list, and **lh** is a pointer to a node.

One subtlety here should be explained further. The head and tail portions of the header actually overlap. This is best understood if you think of the head and tail as two separate nodes. The **lh_Head** field is the **ln_Succ** field of the first node in the list, and the **lh_Tail** field is its **ln_Pred**. The **lh_Tail** is set permanently to zero to indicate that this node is the first on the list—that is, it has no successors. A similar method is used for the tail node. The **lh_Tail** field is the **lh_Succ** field of the last node in the list and the **lh_TailPred** field is its **ln_Pred**. In this case, the zero **lh_Tail** indicates that the node is the last on the list—that is, it has no predecessors.

HEADER INITIALIZATION

List headers must be properly initialized before use. It is not adequate to initialize the entire header to zero. The head and tail entries must be set up correctly.

The header should be initialized as follows:

1. Assign the **lh_Head** field to the address of **lh_Tail**.
2. Assign the **lh_TailPred** field to the address of **lh_Head**.
3. Clear the **lh_Tail** field.
4. Set **lh_Type** to the same data type as that of the nodes to be kept in this list.

In C, an example initialization might look like this:

```
struct List list;
list.lh_Head = &list.lh_Tail;
list.lh_TailPred = &list.lh_Head;
list.lh_Tail = 0;
list.lh_Type = NT_INTERRUPTS;
```

In assembly code, only four instructions are necessary to initialize the header:

```
MOVE.L    A0,(A0)
ADDQ.L    #LH_TAIL,(A0)
CLR.L     LH_TAIL(A0)
MOVE.L    A0 ,LH_TAILPRED(A0)
```

Note that this sequence of instructions is the same as is used in the macro **NEWLIST,** contained in the file *exec/lists.i.* The sequence performs its function without destroying the pointer to the list header in A0 (which is why **ADDQ.L** is used). This function may also be accessed from C as a call to **NewList(lh)** where **lh** is the address of the list header. See the source code for **CreatePort()** in chapter 3, "Messages and Ports," for one instance of its use.

List Functions

Exec provides a number of symmetric functions for handling lists. There are functions for inserting and removing nodes in lists, for adding and removing tail and head nodes in lists, for inserting nodes in a priority order, and for searching a list for a node with a particular name.

INSERTION AND REMOVAL

The **Insert()** function is used for inserting a new node into any position in a list. It always inserts the node following a specified node that is already part of the list. For example, **Insert(lh,ln,pred)** inserts the node after **pred** in the specified list. If the **pred** node points to the list header or is null, the new node will be inserted at the head of the list. Similarly, if the **pred** node points to the list **lh_Tail** field, the new node will be inserted at the tail of the list. However, both of these actions can be better accomplished with the functions mentioned in the "Special Case Insertion" section below.

The **Remove()** function is used to remove a specified node from a list. For example, **Remove(ln)** will remove the specified node from whatever list it is in. *Please note:* to be removed, a node must actually be in the list. If you attempt to remove a node that is not in a list, you will cause serious system internal problems.

SPECIAL CASE INSERTION

Although the **Insert()** function allows new nodes to be inserted at the head and the tail of a list, the **AddHead()** and **AddTail()** functions will do so with higher efficiency. Adding to the head or tail of a list is common practice in queue type operations, as in first-in-first-out (FIFO) or last-in-first-out (LIFO or stack) operations. For example, **AddHead(lh,ln)** would insert the node at the head of the specified list.

SPECIAL CASE REMOVAL

The two functions **RemHead()** and **RemTail()** are used in combination with **AddHead()** and **AddTail()** to create special list ordering. When you combine **AddTail()** and **RemHead()**, you produce a first-in-first-out (FIFO) list. When you combine **AddHead()** and **RemHead()** a last-in-first-out (LIFO or stack) list is produced. **RemTail()** exists for symmetry. Other combinations of these functions can also be used productively. For example, **RemTail(lh)** removes the last node from the specified list and returns a pointer to it as a result. If the list is empty, it returns a zero result.

PRIORITIZED INSERTION

None of the list functions discussed so far makes use of the priority field in the list data structure. The **Enqueue()** function makes use of this field and is equivalent to **Insert()** for a priority sorted list. It performs an insert on a priority basis, keeping the higher-priority nodes towards the head of the list. All nodes passed to this function must have their priority assigned prior to the call. For example, **Enqueue(lh,ln)** inserts the node into the prioritized list after the last node of same or higher priority.

As mentioned earlier, the highest-priority node is at the head of the list, and the lowest-priority node is at the tail of the list. The **RemHead()** function will return the highest-priority node, and **RemTail()** will return the lowest-priority node.

Note that if you insert a node that has the same priority as another node in the list, **Enqueue()** will use FIFO ordering. The new node is inserted following the last node of equal priority.

SEARCHING BY NAME

Because most lists contain nodes with symbolic names attached (via the **ln_Name** field), it is possible to find a node by its name. This naming technique is used throughout Exec for such nodes as tasks, libraries, devices, and resources.

The **FindName()** function is provided to search a list for the first node with a given name. For example, **FindName(lh, "Furrbol")** returns a pointer to the first node named "Furrbol." If no such node exists, a zero is returned. The case of the name characters is significant; "foo" is different from "Foo."

To find multiple occurrences of nodes with identical names, the **FindName()** function is called multiple times. For example, if you want to find the second node with the "Furrbol" name:

```
struct List *lh;
struct Node *ln, *FindName();
ln = FindName(lh, "Furrbol");
if (ln != 0) {
   ln = FindName(ln, "Furrbol");
}
```

Notice that the second search uses the node found by the first search. The **FindName()** function never compares the specified name with that of the starting node. It always begins the search with the successor of the starting node.

List Macros

Assembly code programmers may want to optimize their code by using assembly code list macros. Because these macros actually embed the specified list operation into the code, they result in slightly faster operations. The file *exec/lists.i* contains the recommended set of macros. For example, the following instructions implement the **REMOVE** macro:

```
MOVE.L    (A1),A0             * get successor
MOVE.L    LN_PRED(A1),A1      * get predecessor
MOVE.L    A0,(A1)             * fix up predecessor's succ pointer
MOVE.L    A1,LN_PRED(A0)      * fixup successor's pred pointer
```

Empty Lists

It is often important to determine if a list is empty. This can be done in many ways, but only two are worth mentioning. If either the **lh_TailPred** field is pointing to the list header or the **ln_Succ** field of the **lh_Head** is zero, then the list is empty.

In C, for example, these methods would be written as follows:

```
if (list.lh_TailPred == &list) {
      printf ("list is empty");
}
```

or

```
if (list.lh_Head->ln_Succ == 0) {
      printf ("list is empty");
}
```

In assembly code, if A0 points to the list header, these methods would be written as follows:

```
CMP.L      LH_TAILPRED(A0),A0
BEQ        list_is_empty
```

or

```
MOVE.L     LH_HEAD(A0),A1
TST.L      LN_SUCC(A1)
BEQ        list_is_empty
```

Because **LH_HEAD** and **LN_SUCC** are both zero offsets, the second case can be simplified.

Scanning a List

Occasionally a program may need to scan a list to locate a particular node, find a node that has a field with a particular value, or just print the list. Because lists are linked in both the forward and backward directions, the list can be scanned from either the head or tail.

Here is an example of C code that uses a **for** loop to print the names of all nodes in a list:

```
struct List *lh;
struct Node *ln;
for (ln = lh -> lh_Head; ln -> ln_Succ; ln = ln -> ln_Succ) {
      printf ("node %lx is named %s", ln, ln -> ln_name);
}
```

In assembly code, it is more efficient to use a lookahead cache pointer when scanning a list. In this example the list is scanned until the first zero-priority node is reached:

```
      MOVE.L  (A1),D1        * first node
scan:
      MOVE.L  D1,A1
      MOVE.L  (A1),D1        * lookahead to next
      BEQ.S   not_found      * end of list
      TST.B   LN_PRI(A1)
      BNE.S   scan
      ...                    * found one

not_found:
```

Important Note: It is possible to collide with other tasks when manipulating shared system lists. For example, if some other task happens to be modifying a list while your task scans it, an inconsistent view of the list may be formed. This can result in a corrupted system. Generally it is not permissible to read or write a shared system list without first locking out access from other tasks (and in some cases locking out access from interrupts). This technique of mutual exclusion is discussed in the "Tasks" chapter.

Chapter 2

TASKS

The management of tasks on the Amiga involves task creation, termination, event signals, traps, exceptions, and mutual exclusion. The discussions in this chapter assume that you have a basic understanding of lists (see chapter 1) and some understanding of multitasking principles.

Introduction

Multitasking is one of the primary features supported by Exec. Multitasking is the ability of an operating system to manage the simultaneous execution of multiple independent processor contexts. In addition, good multitasking does this in a transparent fashion: a task is not forced to recognize the existence of other tasks. In Exec this involves sharing the 68000 processor among a number of concurrent programs, providing each with its own virtual processor.

SCHEDULING

Exec accomplishes multitasking by *multiplexing* the 68000 processor among a number of task contexts. Every task has an assigned priority, and tasks are scheduled to use the processor on a priority basis. The highest-priority ready task is selected and receives processing until a higher-priority task becomes active, the running task exceeds a preset time period (a quantum) and there is another equal-priority task ready to run, or the task needs to wait for an external event before it can continue.

Task scheduling is normally preemptive in nature. The running task may lose the processor at nearly any moment by being displaced by another more urgent task. Later, when the preempted task regains the processor, it continues from where it left off.

It is also possible to run a task in a nonpreemptive manner. This mode of execution is generally reserved for system data structure access. It is discussed in the "Exclusion" section toward the end of this chapter.

In addition to the prioritized scheduling of tasks, *time-slicing* also occurs for tasks with the same priority. In this scheme a task is allowed to execute for a quantum (a preset time period). If the task exceeds this period, the system will preempt it and give other tasks of the same priority a chance to run. This will result in a time-sequenced *round robin* scheduling of all equal-priority tasks.

Because of the prioritized nature of task scheduling, tasks must avoid performing the *busy wait* technique of polling. In this technique, a piece of code loops endlessly waiting for a change in state of some external condition. Tasks that use the busy wait technique waste the processor and eat up all its spare power. In most cases this prevents lower-priority tasks from receiving any processor time. Because certain devices, such as the keyboard and the disk, depend on their associated tasks, using a busy wait at a high priority may defer important system services. Busy waiting can even cause system deadlocks.

When there are no ready tasks, the processor is halted and only interrupts will be serviced. Because task multiplexing often occurs as a result of events triggered by system interrupts, this is not a problem. Halting the processor often helps improve the performance of other system bus devices.

TASK STATES

For every task, Exec maintains state information to indicate its status. A normally operating task will exist in one of three states:

running	A task that is running is one that currently owns the processor. This usually means that the task is actually executing, but it is also possible that it has been temporarily displaced by a system interrupt.
ready	A task that is ready is one that is not currently executing but that is scheduled for the processor. The task will receive processor time based on its priority relative to the priorities of other running and ready tasks.
waiting	A task that is waiting is in a paused state waiting for an external event to occur. Such a task is not scheduled to use the processor. The task will be made ready only when one of its external events occurs (see the "Signals" section below).

A task may also exist in a few transient states:

added	A task in the added state has just been added to Exec and has not yet been scheduled for processing.
removed	A task in the removed state is being removed. Tasks in this state are effectively terminated and are usually undergoing clean-up operations.
exception	A task in the exception state is scheduled for special exception processing.

TASK QUEUES

Tasks that are not in the running state are linked into one of two system queues. Tasks that are marked as ready to run but awaiting an opportunity to do so are kept in the *ready queue.* This queue is always kept in a priority sorted order with the highest priority task at the head of the queue. A *waiting queue* accounts for tasks that are awaiting external events. Unlike the ready queue, the waiting queue is not kept sorted by priority. New entries are appended to the tail of the queue. A task will remain in the waiting queue until it is awakened by an event (at which time it is placed into the ready queue).

PRIORITY

A task's priority indicates its importance relative to other tasks. Higher-priority tasks receive the processor before lower-priority tasks do. Task priority is stored as a signed number ranging from -128 to +127. Higher priorities are represented by more positive values; zero is considered the neutral priority. Normally, system tasks execute somewhere in the range of +20 to -20.

It is not wise to needlessly raise a task's priority. Sometimes it may be necessary to carefully select a priority so that the task can properly interact with various system tasks. The **ChangePri()** Exec function is provided for this purpose.

STRUCTURE

Exec maintains task context and state information in a task-control data structure. Like most Exec structures, these structures are dynamically linked onto various task queues through the use of a prepended list **Node** structure. The C-language form of this structure is defined in the *exec/task.h* include file as follows:

```
extern struct Task {
      struct      Node tc_Node;
      UBYTE       tc_Flags;
      UBYTE       tc_State;
      BYTE        tc_IDNestCnt;          /* intr disabled nesting */
      BYTE        tc_TDNestCnt;          /* task disabled nesting */
      ULONG       tc_SigAlloc;           /* sigs allocated */
      ULONG       tc_SigWait;            /* sigs we are waiting for */
      ULONG       tc_SigRecvd;           /* sigs we have received */
      ULONG       tc_SigExcept;          /* sigs we will take excepts for */
      UWORD       tc_TrapAlloc;          /* traps allocated */
      UWORD       tc_TrapAble;           /* traps enabled */
      APTR        tc_ExceptData;         /* points to except data */
      APTR        tc_ExceptCode;         /* points to except code */
      APTR        tc_TrapData;           /* points to trap code */
      APTR        tc_TrapCode;           /* points to trap data */
      APTR        tc_SPReg;              /* stack pointer */
      APTR        tc_SPLower;            /* stack lower bound */
      APTR        tc_SPUpper;            /* stack upper bound + 2*/
      VOID        (*tc_Switch)();        /* task losing CPU */
      VOID        (*tc_Launch)();        /* task getting CPU */
      struct      List tc_MemEntry;      /* allocated memory */
      APTR        tc_UserData;           /* per task data */
};
```

A similar assembly code structure is available in the *exec/tasks.i* include file.

Most of these fields are not relevant for simple tasks; they are used by Exec for state and administrative purposes. A few fields, however, are provided for the advanced programs that support higher level environments (as in the case of *processes*) or require precise control (as in *devices*). The following sections explain these fields in more detail.

Creation

To create a new task you must allocate a task structure, initialize its various fields, and then link it into Exec with a call to **AddTask()**. The task structure may be allocated by calling the **AllocMem()** function with the **MEMF_CLEAR** and **MEMF_PUBLIC** allocation attributes. These attributes indicate that the data structure is to be pre-initialized to zero and that the structure is shared.

The **Task** fields that require initialization depend on how you intend to use the task. For the simplest of tasks, only a few fields must be initialized:

tc_Node
: The task list node structure. This includes the task's priority, its type, and its name (refer to the "Lists and Queues" chapter).

tc_SPLower
: The lower memory bound of the task's stack

tc_SPUpper
: The upper memory bound of the task's stack

tc_SPReg
: The initial stack pointer. Because task stacks grow *downward* in memory, this field is usually set to the same value as **tc_SPUpper**.

Zeroing all other unused fields will cause Exec to supply the appropriate system default values. Allocating the structure with the **MEMF_CLEAR** attribute is an easy way to be sure that this happens.

Once the structure has been initialized, it must be linked to Exec. This is done with a call to **AddTask()** in which the following parameters are specified:

task	A pointer to an initialized task structure.
initialPC	The entry point of your task code. This is the address of the first instruction the new task will execute.
finalPC	The finalization code for your task. This is a code fragment that will receive control if the **initialPC** routine ever performs a return (**RTS**). This exists to prevent your task from being launched into random memory upon an accidental return. The **finalPC** routine should usually perform various program-related clean-up duties and should then remove the task. If a zero is supplied as this parameter, Exec will use its default finalization code (which simply calls the **RemTask()** function)

Depending on the priority of the new task and the priorities of other tasks in the system, the newly added task may immediately begin execution.

Here is an example of simple task creation:

```
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/tasks.h"
#define STACK_SIZE 1000
extern APTR AllocMem();
extern EntryPoint();

SimpleTask()
{
  struct Task *tc;
  APTR stack;
  stack = (APTR) AllocMem (STACK_SIZE, MEMF_CLEAR );
  if (stack == 0) {
    printf ("not enough memory for task stack");
    return(0);
  }

  tc = (struct Task *) AllocMem (sizeof(struct Task),
    MEMF_CLEAR | MEMF_PUBLIC);
  if (tc == 0) {
    printf ("not enough memory for task control structure");
    FreeMem (stack, STACK_SIZE);
    return(0);
  }

  task = (struct Task *) AllocMem (sizeof(struct Task),
    MEMF_CLEAR | MEMF_PUBLIC);
  if (tc == 0) {
    printf ("not enough memory for task name");
    FreeMem (stack, STACK_SIZE);
    return(0);
  }

  tc -> tc_SPLower = (APTR) stack;
  tc -> tc_SPUpper = (APTR) (STACK_SIZE + (ULONG) stack);
  tc -> tc_SPReg   = tc -> tc_SPUpper;

  tc -> tc_Node.ln_Type = NT_TASK;
  tc -> tc_Node.ln_Name = "example.task";

  AddTask (tc, EntryPoint, 0);
}
```

STACK

Every task requires a stack. All task stacks are *user mode* stacks (in the language of the 68000) and are addressed through the A7 CPU register. All normal code execution occurs on this task stack. Special modes of execution (processor traps and system interrupts for example) execute on a single *supervisor mode* stack and do not directly affect task stacks.

Task stacks are normally used to store local variables, subroutine return addresses, and saved register values. Additionally, when a task loses the processor, all of its current registers are preserved on this stack (with the exception of the stack pointer itself, which must be saved in the task structure).

The amount of stack used by a task can vary widely. The minimum stack size is 70 bytes, which is the number required to save 17 CPU registers and a single return address. Of course, a stack of this size would not give you adequate space to perform any subroutine calls (because the return address occupies stack space). On the other hand, a stack size of 1K would suffice to call most system functions but would not allow much in the way of local variable storage.

Because stack-bounds checking is not provided as a service of Exec, it is important to provide enough space for your task stack. Stack overflows are always difficult to debug and may result not only in the erratic failure of your task but also in the mysterious malfunction of other Amiga subsystems.

Termination

Task termination may occur as the result of a number of situations:

1. A program returning from its **initialPC** routine and dropping into its **finalPC** routine or the system default finalizer.

2. A task trap that is too serious for a recovery action. This includes traps like processor bus error, odd address access errors, etc.

3. A trap that is not handled by the task. For example, the task might be terminated if your code happened to encounter a processor **TRAP** instruction and you did not provide a trap handling routine.

4. An explicit call to the Exec **RemTask()** function.

Task termination involves the deallocation of system resources and the removal of the task structure from Exec. The most important part of task termination is the deallocation of system resources. A task must return all memory that it allocated for its private use, it must terminate any outstanding I/O commands, and it must close access to any system libraries or devices that it has opened.

It is wise to adopt a strategy for task clean-up responsibility. You should decide whether resource allocation and deallocation is the duty of the creator task or the newly created task. Sometimes it is easier and safer for the creator to handle the necessary resource allocation and deallocation on behalf of its offspring. On the other hand, if you expect the creator to terminate before its offspring, it would not be able to handle resource deallocation. In such a case, each of its child tasks would need to deallocate its own resources.

Signals

Tasks often need to coordinate with other concurrent system activities (other tasks and interrupts). Such coordination is achieved through the synchronized exchange of specific event indicators called *signals*. This is the primary mechanism responsible for all intertask communication and synchronization on the Amiga.

The signal mechanism operates at a low level and is designed for high performance. Signals often remain hidden from the user program. The message system, for instance, may use signals to indicate the arrival of a new message. The message system is described in more detail in chapter 3.

The signal system is designed to support independent simultaneous events. Signals may be thought of as occurring in parallel. Each task may define up to 32 independent signals. These signals are stored as single bits in a few fields of the task control structure, and one or more signals can occur at the same time.

All of these signals are considered *task relative*: a task may assign its own significance to a particular signal. Signals are not broadcast to all tasks; they are directed only to individual tasks. A signal has meaning to the task that defined it and to those tasks that have been informed of its meaning. For example, signal bit 12 may indicate a timeout event to one task, but to another task it may indicate a message arrival event.

ALLOCATION

As mentioned above, a task assigns its own meaning to a particular signal. Because certain system libraries may occasionally require the use of a signal, there is a convention for signal allocation. It is unwise ever to make assumptions about which signals are actually in use.

Before a signal can be used, it must be allocated with the **AllocSignal()** function. This marks the signal as being in use and prevents the accidental use of the same signal for more than one event. You may ask for either a specific signal number or the next free signal. The state of the newly allocated signal is cleared (ready for use). Generally it is best to let the system assign you the next free signal. Of the 32 available signals, the lower 16 are usually reserved for system use. This leaves the upper 16 signals free for the user. Other subsystems that you may call depend on **AllocSignal()**.

The following C example asks for the next free signal to be allocated for its use:

```
signal = AllocSignal(-1);
if (signal == -1) {
      printf("no signal bits available");
      return;
} else {
      printf("allocated signal number %ld", signal);
}
```

Note that the value returned by **AllocSignal()** is a signal bit number. This value cannot be used directly in calls to signal-related functions without first being converted to a mask:

```
mask = 1 << signal;
```

When a signal is no longer needed, it should be freed for reuse with **FreeSignal()**.

It is important to realize that signal bit allocation is relevant only to the running task. You cannot allocate a signal from another task.

WAITING FOR A SIGNAL

Signals are most often used to wake up a task upon the occurrence of some external event. This happens when a task is in its wait state and another task (or a system interrupt) causes a signal. The **Wait()** function specifies the set of signals that will wake up the task and then puts the task to sleep (into the waiting state). Any one signal or any combination of signals from this set are sufficient to awake the task. **Wait()** returns a mask indicating which signals from this set satisfied the wait. The **Wait()** function implicitly clears those signals that satisfied the wait. This effectively resets those signals for reuse.

Because tasks (and interrupts) normally execute asynchronously, it is often possible to receive a particular signal before a task actually waits for it. To avoid missing any events, programs should hold signals until the **Wait()** function is called, or until it is explicitly cleared (with **SetSignal()**). In such cases a wait will be immediately satisfied, and the task will not be put to sleep.

As mentioned earlier, a task may wait for more than one signal. When the task returns from the wait, the actual signal mask is returned. Usually the program must check which signals occurred and take the appropriate action. The order in which these bits are checked is often important. Here is a hypothetical example:

```
signals = Wait (newCharSig | cancelSig | timeOutSig);
if (signals & cancelSig) {
     printf ("canceled");
}
if (signals & newCharSig) {
     printf ("new character");
}
if (signals & timeOutSig) {
     printf ("timeout");
}
```

This will put the task to sleep, waiting for a new character, a cancel event, or the expiration of a time period. Notice that this code checks for a cancel signal before checking for a new character or a timeout. Although a program can check for the occurrence of a particular event by checking whether its signal has occurred, this may lead to busy wait polling. Such polling is wasteful of the processor and is usually detrimental to the proper function of the system.

GENERATING A SIGNAL

Signals may be generated from both tasks and system interrupts with the **Signal()** function. For example **Signal(tc,mask)** would signal the task with the mask signals. More than one signal can be specified in the mask.

Exclusion

From time to time the advanced system program may find it necessary to access global system data structures. Because these structures are shared by the system and by other tasks that execute asynchronously to your task, it is wise for you to exclude simultaneous access to these structures. This can be accomplished by *forbidding* or *disabling*, or with the use of *semaphores*. A section of code that requires the use of any of these mechanisms to lock out access by others is termed a *critical section.*

FORBIDDING

Forbidding is used when a task is accessing shared structures that might also be accessed at the same time from another task. It effectively eliminates the possibility of simultaneous access by imposing *nonpreemptive* task scheduling. This has the net effect of disabling multitasking for as long as your task remains in its running state. While forbidden, your task will continue running until it performs a call to **Wait()** or exits from the forbidden state. Interrupts will occur normally, but no new tasks will be dispatched, *regardless of their priorities.*

When a task running in the forbidden state calls the **Wait()** function, it implies a temporary exit from its forbidden state. While the task is waiting, the system will perform normally. When the task receives one of the signals it is waiting for, it will again reenter the forbidden state. To become forbidden, a task calls the **Forbid()** function. To escape, the **Permit()** function is used. The use of these functions may be nested with the expected affects; you will not exit the forbidden mode until you call the outermost **Permit()**.

As an example, Exec memory region lists should be accessed only when forbidden. To access these lists without forbidding jeopardizes the integrity of the entire system.

```
struct ExecBase *eb;
struct MemHeader *mh;
APTR firsts[ARRAYSIZE];
int count;

Forbid();
for (mh = (struct MemHeader *) eb -> MemList.lh_Head;
          mh -> mh_Node.ln_Succ;
          mh = mh -> mh_Node.ln_Succ) {
     firsts[count++] = mh -> mh_First;
}
Permit();
```

As this program traverses down the memory region list, it remains forbidden to prevent the list from changing as it is being accessed.

DISABLING

Disabling is similar to forbidding, but it also prevents interrupts from occurring during a critical section. Disabling is required when a task accesses structures that are shared by interrupt code. It eliminates the possibility of an interrupt accessing shared structures by preventing interrupts from occurring.

To disable interrupts you can call the **Disable()** function. If you are writing in assembly code, the **DISABLE** macro is more efficient (but consumes more code space). To enable interrupts again, use the **Enable()** function and **ENABLE** macros.

Like forbidden sections, disabled sections can be nested. Also like forbidden sections, the **Wait()** function implies an **Enable()** until the task again regains the processor.

It is important to realize that there is a danger in using disabled sections. Because the software on the Amiga depends heavily on its interrupts occurring in nearly real time, you cannot disable for more than a very brief instant. A rule of thumb is to disable for no more than 250 microseconds.

Masking interrupts by changing the 68000 processor interrupt priority levels with the **MOVESR** instruction can also be dangerous and is generally discouraged. The disable- and enable-related functions and macros control interrupts through the 4703 custom chip and *not* through the 68000 priority level. In addition, the processor priority level can be altered only from supervisor mode (which means this process is much less efficient).

It is never necessary to both disable and forbid. Because disable prevents interrupts, it also prevents preemptory task scheduling. Many Exec lists can only be accessed while disabled. Suppose you want to print the names of all waiting tasks. You would need to access the task list from a disabled section. In addition, you must avoid calling certain system functions that require multitasking to function properly (**printf()** for example). In this example, the names are gathered into a name array while the code section is disabled. Then the code section is enabled and the names are printed.

```
#include "exec/types.h"
#include "exec/execbase.h"
#include "exec/tasks.h"
extern struct ExecBase *SysBase;

main()
{
   struct Task *task;
   char *names[20];
   int count, i;
   count = 0;
   Delay(50);
   Disable();

   for (task = (struct Task *)SysBase->TaskWait.lh_Head;
       task->tc_Node.ln_Succ; /* stop when Successor node == 0 */
            task =  (struct Task *)task->tc_Node.ln_Succ) {
      names[count++] = task->tc_Node.ln_Name;
   }
   Enable();

   for (i = 0; i < count; i++)
   {
      printf (" %s\n ", names[i]);
   }
}
```

Of course, the code in this example will have problems if a waiting task is removed before its name is printed. If this were to happen, the name-string pointer would no longer be valid. To avoid such problems it is a good programming practice to copy the entire name string into a temporary buffer.

SEMAPHORES

Messages and message ports can be used as semaphores for the purposes of mutual exclusion. With this method of locking, all tasks agree on a locking convention before accessing shared data structures. Tasks that do not require access are not affected and will run normally, so this type of exclusion is considered preferable to forbidding and disabling. Unfortunately, semaphores also represent a considerable amount of overhead for simple system operations and are not used internal to Exec for efficiency reasons. This form of exclusion is explained in more detail in the "Messages and Ports" chapter.

Exceptions

Tasks can specify that certain asynchronous events cause *exceptions*, which are *task-private* interrupts that redirect a task's flow of control. The task essentially suspends what it is doing and enters a special routine to process its exceptional event.

Exceptions are driven by the task signal mechanism described earlier in this chapter. Instead of waiting for a signal to occur, you indicate that it is an exception signal with the **SigExcept()** function. When the signal occurs, the task will be "interrupted" from its normal execution and placed in a special exception handler.

The **tc_ExceptCode** and **tc_ExceptData** task fields are used to establish the exception handler. The field **tc_ExceptCode** points to the routine that will handle the initial processing of all exceptions. If this field is zero, Exec will ignore all exceptions. The **tc_ExceptData** field can be used to provide a pointer to related data structure.

On entry to the exception code, the system passes certain parameters in the processor registers. **D0** contains a signal mask indicating which exception has just occurred, and **A1** points to the related exception data (from **tc_ExceptData**). In addition, the previous task context is pushed onto the task's stack. This includes the previous PC, SR, D0-D7, and A0-A6 registers. You can think of an exception as a subtask outside of your normal task. Because task exception code executes in *user* mode, however, the task stack must be large enough to supply the extra space consumed during an exception.

While processing a given exception, Exec prevents that exception from occurring recursively. At exit from your exception-processing code you should return the same value in **D0** to re-enable that exception signal. When the task executes the **RTS** at the end of the handler, the system restores the previous contents of all of the task registers and resumes the task at the point where it was interrupted by the exception signal. When two or more exception codes occur simultaneously, the exception-processing code determines the order in which they are handled by the order in which the signal bits are examined.

Traps

Task *traps* are synchronous exceptions to the normal flow of program control. They are always generated as a direct result of an operation performed by your program's code. Whether they are accidental or purposely generated, they will result in your program being forced into a special condition in which it must immediately handle the trap. Address error, privilege violation, zero divide, and trap instructions all result in task traps. They may be generated directly by the 68000 processor (Motorola calls them "exceptions") or simulated by software.

A task that incurs a trap has no choice but to respond immediately. The task must have a module of code to properly handle the trap. Your task may be aborted if a trap occurs and no means of handling it has been provided.

You may choose to do your own processing of traps. The **tc_TrapCode** field is the address of the handler that you have designed to process the trap. The **tc_TrapData** field is the address of the data area for use by the trap handler.

The 68000 traps of interest are:

2	Bus error
3	Address error
4	Illegal instruction
5	Zero divide
6	CHK instruction
7	TRAPV instruction
8	Privilege violation
9	Trace
10	Line 1010 emulator
11	Line 1111 emulator
32-47	Trap instructions

The actual stack frames generated for these traps are processor-dependent. The 68010 and 68020 processors will generate a different type of stack frame than the 68000. If you plan on having your program handle its own traps, you should not make assumptions about the format of the supervisor stack frame. Check the flags in the **AttnFlags** field of the **ExecBase** structure for the type of processor in use and process the stack frame accordingly.

HANDLERS

For compatibility with the 68000, Exec performs trap handling in supervisor mode. This means that all task switching is disabled during trap handling. At entry to the task's trap handler, the system stack does contain the trap frame as defined in the 68000 manual. A longword exception number is added at the bottom of this frame. That is, when a handler gains control, the top of stack contains the exception number and the 68000 frame immediately follows.

To return from trap processing, remove the exception number from the stack (note that this is the supervisor stack, not the user stack) and then perform a return from exception (**RTE**).

Because trap processing takes place in supervisor mode, with task dispatching disabled, it is strongly urged that you keep trap processing as short as possible or switch back to user mode from within your trap handler. If a trap handler already exists when you add your own trap handler, it is smart to propagate any traps that you do not handle down to the previous handler. This can be done by saving the previous **tc_TrapCode** and **tc_TrapData** for use by your handler.

TRAP INSTRUCTIONS

The **TRAP** instructions in the 68000 generate traps 32-47. Because many independent pieces of system code may desire to use these traps, the **AllocTrap()** and **FreeTrap()** functions are provided. These work in a fashion similar to that used by **AllocSignal()** and **FreeSignal()**, mentioned above.

Allocating traps is simply a bookkeeping job within a task. It does not affect how the system calls the trap handler; it helps coordinate who owns what traps. Exec does nothing to determine whether or not the task is prepared to handle this particular trap. It simply calls your code. It is up to your program to handle the trap.

To allocate any trap, you can use the following code:

```
trap = AllocTrap(-1);
if (trap == -1) {
     printf("all trap instructions are in use");
     return;
}
```

or you can select a specific trap using this code:

```
trap = AllocTrap(3);
if (trap == -1) {
     printf("trap #3 is in use");
     return;
}
```

To free a trap you use **FreeTrap()**.

Chapter 3

MESSAGES AND PORTS

Introduction

For intersystem communication, Exec provides a consistent, high-performance mechanism of messages and ports. This mechanism is used to pass message structures of arbitrary sizes from task to task, interrupt to task, or task to software interrupt. In addition, messages are often used to coordinate operations between a number of cooperating tasks.

A *message* data structure has two parts: system linkage and message body. The system linkage is used by Exec to attach a given message to its destination. The message body contains the actual data of interest. The message body is any arbitrary data block less than 64K bytes in size.

Messages are always sent to a predetermined destination *port.* At a port, incoming messages are queued in a first-in-first-out (FIFO) order. There are no system restrictions on the number of ports or the number of messages that may be queued to a port (other than the amount of available system memory).

Messages are always queued by *reference.* For performance reasons message copying is not performed. In essence, a message between two tasks is a temporary license for the receiving task to use a portion of the memory space of the sending task—that portion being the message itself. This means that if task A sends a message to task B, the message is still part of the task A context. Task A, however, should not access the message until it has been *replied* —that is, until task B has sent the message back, using the **ReplyMsg()** function. This technique of message exchange imposes important restrictions on message access.

Ports

Ports are rendezvous points at which messages are collected. A port may contain any number of outstanding messages from many different originators. When a message arrives at a port, the message is appended to the end of the list of messages for that port, and a prespecified arrival action is invoked. This action may do nothing, or it may cause a predefined task signal or software interrupt (see the "Interrupts" chapter).

Like many Exec structures, ports may be given a symbolic name. Such names are particularly useful for tasks that must rendezvous with dynamically created ports. They are also useful for debugging purposes.

STRUCTURE

A message port consists of a **MsgPort** structure as defined in the *exec/ports.h* and *exec/ports.i* include files. The C structure for a port is as follows:

```
struct MsgPort {
      struct   Node mp_Node;
      UBYTE mp_Flags;
      UBYTE mp_SigBit;
      struct   Task *mp_SigTask;
      struct   List mp_MsgList;
};
```

where

mp_Node
: is a standard **Node** structure. This is useful for tasks that might want to rendezvous with a particular message port by name.

mp_Flags
: are used to indicate message arrival actions. See the explanation below.

mp_SigBit
: is the signal bit *number* when a port is used with the task signal arrival action.

mp_SigTask
: is a pointer to the task to be signaled. If a software-interrupt arrival action is specified, this is a pointer to the interrupt structure.

mp_MsgList
: is the list header for all messages queued to this port. (See the "Lists and Queues" chapter).

The **mp_Flags** field contains a subfield indicated by the **PF_ACTION** mask. This sub-field specifies the message arrival action that occurs when a port receives a new message. The possibilities are as follows:

PA_SIGNAL
: This subfield tells the program to signal the specified task on the arrival of a new message. Every time a message is put to the port another signal will occur regardless of how many messages have been queued to the port.

PA_SOFTINT
: This subfield causes the specified software interrupt. Like **PA_SIGNAL**, **PA_SOFTINT** will cause the software interrupt to be posted every time a message is received.

PA_IGNORE

This subfield tells the program to perform no operation other than queuing the message. This action is often used to stop signaling or software interrupts without disturbing the contents of the **mp_SigTask** field.

It is important to realize that a port's arrival action will occur for each new message queued, and that there is not a one-to-one correspondence between messages and signals. Task signals are only single-bit flags so there is no record of how many times a particular signal occurred. There may be many messages queued and only a single task signal. All of this has certain implications when designing code that deals with these actions. Your code should not depend on receiving a signal for every message at your port. All of this is also true for software interrupts.

CREATION

To create a new message port, you must allocate and initialize a **MsgPort** structure. If you desire to make the port *public*, you will also need to call the **AddPort()** function. Port structure initialization involves setting up a **Node** structure, establishing the message arrival action with its parameters, and initializing the list header. The following example of port creation is equivalent to the **CreatePort()** function as supplied in *amiga.lib*:

```
extern APTR AllocMem();
extern UBYTE AllocSignal();
extern struct Task *FindTask();

struct MsgPort *
CreatePort (name, pri)
char *name;
BYTE  pri;
{
    int sigBit;
    struct MsgPort *mp;

    if ((sigBit = AllocSignal (-1)) === -1)
        return ((struct MsgPort *) 0);

    port = AllocMem (sizeof(*port), MEMF_CLEAR | MEMF_PUBLIC);
    if (port == 0) {
        FreeSignal (sigBit);
        return ((struct MsgPort *) (0));
    }
```

```
        mp->mp_Node.ln_Name = name;
        mp->mp_Node.ln_Pri = pri;
        mp->mp_Node.ln_Type = NT_MSGPORT;

        mp->mp_Flags = PA_SIGNAL;
        mp->mp_SigBit = sigBit;
        mp->mp_SigTask = FindTask (0);

        if (name != 0) {
            AddPort (mp);
        } else {
            NewList (&mp->mp_MsgList);
        }
        return (mp);
    }
```

DELETION

Before a message port is deleted, all outstanding messages from other tasks must be returned. This is done by replying to each message until the message queue is empty. Of course, there is no need to reply to messages owned by the current task (the task performing the port deletion). Public ports attached to the system with **AddPort()** must be removed from the system with **RemPort()**.

RENDEZVOUS

The **FindPort()** function provides a means of finding the address of a public port given its symbolic name. For example, **FindPort("Spyder")** will return either the address of the message port or a zero indicating that no such public port exists. Names should be made rather unique to prevent collisions among multiple applications. It is a good idea to use your application name as a prefix for your port name.

Messages

As mentioned earlier, a message contains both system header information and the actual message content. The system header is of the **Message** form defined in *exec/ports.h* and *exec/ports.i*. In C this structure is as follows:

```
struct Message {
      struct    Node mn_Node;
      struct    MsgPort *mn_ReplyPort;
      UWORD mn_Length;
};
```

where

mn_Node
: is a standard **Node** structure used for port linkage.

mn_ReplyPort
: is used to indicate a port to which this message will be returned when a reply is necessary.

mn_Length
: indicates the length of the message body in bytes.

This structure is always attached to the head of all messages. For example, if you want a message structure that contains the x and y coordinates of a point on the screen, you could define it as follows:

```
struct XYMessage {
      struct    Message xy_Msg;
      UWORD x,y;
}
```

For this structure, the **mn_Length** field should be set to **sizeof XYMessage**.

PUTTING A MESSAGE

A message is delivered to a given destination port with the **PutMsg()** function. The message is queued to the port, and that port's arrival action is invoked. If the action specifies a task signal or a software interrupt, the originating task may temporarily lose the processor while the destination processes the message. If a reply to the message is required, the **mn_ReplyPort** field must be set up prior to the call to **PutMsg()**.

Here is a simple program fragment for putting a message to a public port:

```
struct MsgPort *mp, *replymp;
struct XYMessage *xymsg;

xymsg = (struct XYMessage*) AllocMem (sizeof(*xymsg), MEMF_PUBLIC);
if (xymsg == 0) {
      printf ("not enough memory for message");
      return;
}

replymp = CreatePort ("xyreplyport",0);
      /* as defined earlier in this chapter */
if (replymp == 0) {
      printf ("could not create the reply port");
      FreeMem (xymsg, sizeof(*xymsg));
      return;
}

xymsg -> xy_Msg.mn_Node.ln_Type = NT_MESSAGE;
xymsg -> xy_Msg.mn_ReplyPort = replyport;

mp = FindPort ("Spyder");
if (mp == 0) {
      printf ("Spyder port not found");
      return;
}

PutMsg (mp, xymsg);
```

WAITING FOR A MESSAGE

A task may go to sleep waiting for a message to arrive at one or more ports. This technique is widely used on the Amiga as a general form of event notification. For example, it is used extensively by tasks for I/O request completion.

To wait for the arrival of a message, the message port must be properly initialized. In particular, the **mp_SigTask** field must contain the address of the task to be signaled and **mp_SigBit** must contain a preallocated signal number (as described in the "Tasks" chapter). You can call the **WaitPort()** function to wait for a message to arrive at a port. This function will return the first message queued to a port. If the port is empty, your task will go to sleep waiting for the first message. If the port is not empty, your task will not go to sleep.

A more general form of waiting for a message involves the use of the **Wait()** function (see the "Tasks" chapter). This function waits for task event signals directly. If the signal assigned to the message port occurs, the task will awaken. Using the **Wait()** function is more general because you can wait for more than just a single message port. For example, you may want to wait for a message and a timeout signal. The **Wait()** function lets you specify a mask containing the signals associated with your message port and your timeout signal.

Here's an example using **WaitPort()**:

```
struct MsgPort *mp;
struct Message *msg, *WaitPort();
int SigBit;

SigBit = AllocSignal (-1);
if (SigBit == -1) {
      printf ("no free signal bits");
      return;
}

mp -> mp_Flags |= PA_signal;
mp -> mp_SigBit = SigBit;
mp -> mp_SigTask = FindTask (0);          /* self */

msg = WaitPort (mp);
```

Note that **WaitPort()** only returns a pointer to the first message in a port. It does not actually remove the message from the port queue.

GETTING A MESSAGE

Messages are usually removed from ports with the **GetMsg()** function. This function removes the next message at the head of the port queue and returns a pointer to it. If there are no messages in a port, this function returns a zero.

The example below illustrates the use of **GetMsg()** to print the contents of all messages in a port:

```
while ((msg = GetMsg (mp)) != 0) {
      printf ("x=%ld y=%ld", msg->x, msg->y);
}
```

Certain messages may be more important than others. Because ports impose FIFO ordering, these important messages may get queued behind other messages regardless of their priority. If it is necessary to recognize more important messages, it is easiest to create another port for these special messages.

REPLYING

When the operations associated with receiving a new message are finished, it is usually necessary to send the message back to the originator. The receiver replies the message by returning it to the originator using the **ReplyMsg()** function. This is important because it notifies the originator that the message can be reused or deallocated. The **ReplyMsg()** function serves this purpose. It returns the message to the port specified in the **mn_ReplyPort** field of the message. If this field is zero, no reply is returned.

The previous example can be enhanced to reply to each of its messages:

```
while ((msg = GetMsg (mp)) != 0) {
      printf ("x=%ld y=%ld", msg->x, msg->y);
      ReplyMsg (msg);
}
```

Notice that the reply does not occur until *after* the message values have been used.

Often the operations associated with receiving a message involve returning *results* to the originator. Typically this is done within the message itself. The receiver places the results in fields defined (or perhaps reused) within the message body before replying the message back to the originator. Receipt of the replied message at the originator's reply port indicates it is once again safe for the originator to use or change the values found within the message.

Chapter 4

INPUT/OUTPUT

Introduction

One of the primary purposes of Exec is to provide a standard form for all device input/output (I/O). This includes the definition of a standard device interface, the format for I/O requests, and the establishment of rules for normal device/task interaction. In addition, the guidelines for nonstandard device I/O are also defined. In the design of the Amiga I/O system, great care has been taken to avoid dictating the form of implementation or the internal operational characteristics of a device.

In its purest sense, a *device* is an abstraction that represents a set of well-defined interactions with some form of physical media. This abstraction is supported by a standard Exec data structure and an independent system code module. The data structure provides the external interface and maintains the current device state. The code module supplies the operations necessary to make the device functional. (In many operating systems, this code module is referred to as a device *driver*. See *Amiga ROM Kernel Reference Manual: Libraries and Devices* for the source assembly language code for a disk-resident device driver with its own task for handling I/O requests.)

A device *unit* is an instance of a device. It shares the same device data structure and code module with all other units of the same device; however, it operates in an independent fashion. Often units correspond to separate physical subsystems of the same general device class. For example, each Amiga floppy disk drive is an independent unit of the same device. There is only one device data structure and one code module to support all of these units.

Exec I/O is often performed using the message system described in the chapter 3. Most aspects of message passing are concealed within the Exec I/O support routines. However, it is important to realize that I/O request blocks, once issued, must *not* be modified or reused until they are returned to your program's control by Exec.

Request Structure

An I/O *request* is always directed to a device unit. This request is organized as a control block and contains a *command* to be performed on a specified unit. It is passed through a standard device interface function, where it is processed and executed by the device's code module. All request parameters are included in the request control block, and I/O request results are returned in the same control block.

Every device unit responds to a standard set of commands, and may optionally provide a nonstandard set of commands as well. The standard commands are explained later in this chapter. Nonstandard commands are discussed in the documentation pertaining to the particular device involved.

An I/O request always includes at least an **IORequest** data structure. This is a standard header used for all I/O requests. It is defined in the *exec/io.h* and *exec/io.i* include files as follows:

```
struct IORequest {
        struct      Message  *io_Message;
        struct      Device  *io_Device;
        struct      Unit  *io_Unit;
        UWORD       io_Command;
        UBYTE       io_Flags;
        BYTE        io_Error;
};
```

where

io_Message
: is a message header (see the "Messages and Ports" chapter). This header is used by the device to return I/O requests upon completion. It is also used by devices internally for I/O request queuing. This header must be properly initialized for I/O to work correctly.

io_Device
: is a pointer to the device data structure node. This field is automatically set up by an Exec function when the device is opened.

io_Unit
: specifies a unit to the device internally. This is a device *private* field and should not be accessed by the user. The format of this field is device dependent and is set up by the device during the open sequence.

io_Command
: is the command requested. This may be either one of the system standard commands or a device-specific command.

io_Flags
: is used to indicate special request options and state. This field is divided into two subfields of four bits each. The lower four bits are for use by Exec and the upper four bits are available to the device.

io_Error
: is an error or warning number returned upon request completion.

The **io_Device**, **io_Unit**, and **io_Command** fields are not affected by the servicing of the request. This permits repeated I/O using the same request.

The standard I/O requests use an expanded form of the **IORequest** structure:

```
struct IOStdReq {
      struct     Message  io_Message;
      struct     Device  *io_Device;
      struct     Unit  *io_Unit;
      UWORD      io_Command;
      UBYTE      io_Flags;
      BYTE       io_Error;
      ULONG      io_Actual;
      ULONG      io_Length;
      APTR       io_Data;
      ULONG      io_Offset;
}
```

where the additional fields are used as follows:

io_Actual
: indicates the actual number of bytes transferred. This field is valid only upon completion.

io_Length
: is the requested number of bytes to transfer. This field must be set up prior to the request. A special length of -1 is often used to indicate variable-length transfers.

io_Data
: is a pointer to the transfer data buffer.

io_Offset
: indicates a byte offset (for structured devices). For block-structured devices (such as a floppy disk device) this number must be a multiple of the block size.

Devices with nonstandard commands may add their own special fields to the I/O request structure as needed. Such extensions are device specific.

Interface Functions

Four Exec functions are responsible for interfacing I/O requests to actual device drivers. These functions operate independently of the particular device command requested. They deal with the request block as a whole, ignoring its command and its command parameters.

DoIO() is the most commonly used I/O function. It initiates an I/O request and waits for its completion. This is a *synchronous* form of device I/O; control is not returned to the caller until completion.

SendIO()
is used to initiate an I/O request without waiting for completion. This is an *asynchronous* form of device I/O; control is returned even if the request has not completed.

WaitIO()
is used to wait for the completion of a previously initiated asynchronous I/O request. This function will not return control until the request has completed (successfully or unsuccessfully).

CheckIO()
is used to see if an asynchronous I/O request has completed.

In addition to the above Exec functions, there are two I/O related functions that are actually direct entries into the device driver itself. These functions are part of the actual device driver interface to the system and should be used with care. They incur slightly less overhead but require more knowledge of the I/O system internals (you must know how quick I/O works, for instance):

BeginIO()
initiates an IO request. The request will be synchronous or asynchronous depending on the device driver.

AbortIO()
attempts to cancel a previous I/O request. This function is easily accessed as an assembly code macro **ABORTIO** or through the C library Exec support function **AbortIO()**.

Standard Commands

There are eight standard commands to which all devices are expected to respond. If the device is not capable of performing one of these commands, it will at least return an error indication that the command is not supported. These commands are defined in the *exec/io.h* and *exec/io.i* include files.

CMD_RESET
This command resets the device unit. It completely initializes the device unit, returning it to its default configuration, aborting all of its pending I/O, cleaning up any internal data structures, and resetting any related hardware.

CMD_READ
This command reads a specified number of bytes from a device unit into the data buffer. The number of bytes to be read is specified in the **io_Length** field. The number of bytes actually read is returned in the **io_Actual** field.

CMD_WRITE
This command writes a specified number of bytes to a device unit from a data buffer. The number of bytes to be written is specified in the **io_Length** field. The number of bytes actually written is returned in the **io_Actual** field.

CMD_UPDATE
This command forces out all internal buffers, causing device internal memory buffers to be written out to the physical device unit. A device will transparently perform this operation when necessary, but this command allows you to request explicitly that such an action take place. It is useful for devices that maintain internal caches, such as the floppy disk device.

CMD_CLEAR
This command clears all internal buffers. It deletes the entire contents of a device unit's internal buffers. No update is performed; all data is lost.

CMD_STOP
This command stops the device unit immediately (at the first opportunity). All I/O requests continue to queue, but the device unit stops servicing them. This command is useful for devices that may require user intervention (printers, plotters, data networks, etc.).

CMD_START

This command causes the device unit to continue after a previous **CMD_STOP** command. The device resumes from where it was stopped.

CMD_FLUSH

This command aborts all I/O requests, returning all pending I/O requests with an error message.

CMD_NONSTD

Any nonstandard commands begin here. Non standard commands are designated as CMD_NONSTD+0, CMD_NONSTD+1, and so on.

CMD_INVALID

This is a command to which the device should not respond.

Performing I/O

In Exec, I/O is always performed using I/O request blocks. Before I/O is performed, the request block must be properly initialized by both the system and the user. Once this has been done, normal I/O may commence.

PREPARATION

Devices are identified within the system by name (a null-terminated character string). Device units are usually identified by number. The **OpenDevice()** function maps the device name to an actual device and then calls the device to perform its initialization. The device will map the unit number into an internal form for later use. Both Exec and the device driver will initialize the I/O request passed to **OpenDevice()**.

For example, **OpenDevice("trackdisk.device",1,ior,0)** will attempt to open unit one of the floppy disk device, mapping its symbolic name into the address of a device data structure. It also sets up a few internal fields of the request. **OpenDevice()** will return a zero if it was successful and a nonzero error number if it was not.

SYNCHRONOUS REQUESTS

Synchronous I/O requests are initiated with the **DoIO()** function mentioned earlier. **DoIO()** will not return control until the request has completed. Because the device may respond to a request immediately or queue it for later action, an undetermined amount of time may pass before control is returned. With this type of I/O, only one request is serviced at a time.

To perform synchronous I/O, the I/O request block must be prepared as described in the previous section. In addition, **io_Message**, **io_Command**, and perhaps other fields must be initialized.

The **io_Message** field is set up in the same manner as a message. This is described in the "Messages and Ports" chapter.

The **io_Command** field is set to the desired command. For example:

```
    ior->io_Command = CMD_RESET;
    DoIO (ior);
```

performs a reset command.

More involved commands require other fields to be initialized. For example, the commands to read a sector from a disk might look something like the following:

```
    ior->io_Command = CMD_READ;
    ior->io_Length = TD_SECTOR;
    ior->io_Offset = 20 * TD_SECTOR;
    ior->io_Data = buffer;
    DoIO (ior);
```

When the request has completed, the request block is returned with the command results. If an error occurred, **DoIO()** will return the error number. The error number is also indicated in the **io_Error** field of the request.

ASYNCHRONOUS REQUESTS

More efficient programs can take advantage of the multitasking characteristics of the I/O system by using asynchronous I/O, which allows many requests to be performed at the same time. This type of I/O is supported by the **SendIO()**, **WaitIO()**, **CheckIO()**, **BeginIO()**, and **AbortIO()** functions. Asynchronous I/O requests will return almost immediately to the user regardless of whether the request has actually completed. This lets the user maintain control while the I/O is being performed. Multiple I/O requests can be posted in this fashion.

In the disk read example above, asynchronous I/O could be performed by changing the **DoIO()** call to a **SendIO()**:

```
ior->io_Command = CMD_READ;
ior->io_Length = TD_SECTOR;
ior->io_Offset = 20 * TD_SECTOR;
ior->io_Data = buffer;
SendIO (ior);
```

From the time the I/O has been initiated to the time it completes, the request block should not be directly accessed by the program. The device can be said to "own" the request block. Only after the request has completed or successfully aborted should your program access it.

When the I/O completes, the device will return the I/O request block to the reply port specified in its **io_Message** field. After this has happened, you know that the device has finished the I/O. The reply port used to receive the returned request can be set up to cause a task signal when the reply arrives. This technique lets a task sleep until the the request is complete. The **WaitIO()** function can be called to wait for the completion of a previously initiated request.

WaitIO() will handle all of the interaction with the message reply port automatically. If you are using just the **Wait()** function, do not forget to remove the I/O request from your reply port with **GetMsg()**. Once this is done, the request may be reused.

The **CheckIO()** function is handy to determine if a particular I/O request has been satisfied. This function deals with some of the subtleties of I/O in the proper manner.

If you wish to queue several I/O requests to a device, you must issue multiple **SendIO()** requests, each with its own separately-opened request structure. This type of I/O is supported by most devices. A task can also request I/O from a number of devices and then check later for their completion.

Exec also allows for certain types of optimization in device communication. One form of optimization, in which you call the device driver directly, is called quick I/O. This concept is discussed later in this chapter.

CONCLUSION

When a request has completed its I/O, access to the device should be concluded with **CloseDevice()**. This function will inform the device that no further I/O is to be performed with this request. For every **OpenDevice()** there must be a corresponding **CloseDevice()**.

QUICK I/O

For some types of I/O, the normal internal mechanisms of I/O may present a large amount of overhead. This is mostly true for character-oriented I/O, in which each character might be transferred with a separate I/O request. The overhead for such requests could significantly overload the I/O system, resulting in a loss of efficiency for the overall system.

To allow devices to optimize their I/O handling, a mechanism called quick I/O was created. In the **IORequest** data structure, one of the **io_flags** is reserved for quick I/O. When set prior to an I/O request, this flag indicates that the device is allowed to handle the I/O in a special manner. This enables some devices to take certain "short-cuts" when it comes to performing and completing the request.

The quick I/O bit (IOB_QUICK) allows the device to avoid returning the I/O request to the user via the message system (for example, via **ReplyMsg()**) if it can complete the request immediately. If the IOB-QUICK bit is still set at the end of the **BeginIO()** call, the request has already completed and the user will not find the I/O request on his reply port.

The **DoIO()** function normally requests the quick I/O option, whereas the **SendIO()** function does not. Complete control over the mode for quick I/O is possible by calling a device's **BeginIO()** entry directly.

It is up to the device to determine whether it can handle a request marked as quick I/O. If the quick I/O flag is still set when the request has completed, the I/O was performed quickly. This means that no message reply occurred, so the message has not been queued to the reply port.

Standard Devices

The following standard system devices are normally available when the Amiga starts up. Each of these devices is described in *Amiga ROM Kernel Reference Manual: Libraries and Devices.*

Timer	Provides a flexible way of causing task signals or interrupts at second and microsecond intervals.
Trackdisk	Provides direct access to the 3 1/2-inch and 5 1/4-inch floppy disk drives. Among the functions provided are format, seek, read, and write. Normally, trackdisk is used only by AmigaDOS; its functions are enumerated here for direct access where required.
Keyboard	Handles raw information from the keyboard and converts it into input events that can be retrieved and interpreted. Keyboard input events are queued so that no keystrokes will be missed.
Gameport	Handles raw information from the mouse or a joystick device. Gameport events are queued so that no movements will be missed. You can tell the system what type of device is connected and how often to check and report the current status of the device.
Input	The input device combines requests from both the keyboard and the gameport device. Input events from both are merged into a single input event stream on a first-in-first-out basis.
Console	The console device receives its input from the input device. The input portion of the console device is simply a handler for input events filtered by Intuition. It provides what might be called the "traditional" user interface.
Audio	The audio device is provided to control the use of the audio channels.
Narrator	The narrator device is loaded from disk and uses the audio device to produce humanlike synthesized speech.
Serial	The serial device is loaded from disk and initialized on being loaded. It controls serial communications buffering of the input/output, baud rate, and so on.

Parallel	The parallel device is loaded from disk and initialized on being loaded. It controls parallel communications. The parallel device is most often used by a parallel printer driver.
Printer	The printer device driver is loaded from disk. Printers that are supported as of this writing are specified in the "Printer Device Support Code" appendix of the *Amiga ROM Kernel Reference Manual: Libraries and Devices.*
Clipboard	The clipboard device provides a means of "cutting" data from and "pasting" data into applications.

Chapter 5

INTERRUPTS

Introduction

Exec manages the decoding, dispatching, and sharing of all system interrupts. This includes control of hardware interrupts, software interrupts, task-relative interrupts (see the "Tasks" chapter), and interrupt disabling/enabling. In addition, Exec supports a more extended prioritization of interrupts than that provided in the 68000.

The proper operation of multitasking depends heavily on the consistent management of the interrupt system. Task activities are often driven by intersystem communication that is originated by various interrupts.

SEQUENCE OF EVENTS

Before useful interrupt handling code can be executed, a considerable amount of hardware and software activity must occur. Each interrupt must propagate through several hardware and software interfaces before application code is finally dispatched:

1. A hardware device decides to cause an interrupt and sends a signal to the interrupt control portions of the 4703 custom chip.

2. The 4703 interrupt control logic notices this new signal and performs two primary operations. First, it records that the interrupt has been requested by setting a flag bit in the INTREQ register. Second, it examines the INTENA register to determine whether the corresponding interrupt and the interrupt master are enabled. If both are enabled, the 4703 generates a set of three 68000 interrupt request signals. See the *Amiga Hardware Reference Manual* for a more complete explanation of how this is done.

3. These three signals correspond to seven interrupt priority levels in the 68000. If the priority of the new interrupt is *greater* than the current processor priority, an interrupt sequence is initiated. The priority level of the new interrupt is used to index into the top seven words of the processor address space. The odd byte (a vector number) of the indexed word is fetched and then shifted left by two to create a low memory vector address.

4. The 68000 then switches into *supervisor* mode (if it is not already in that mode), and saves copies of the status register and program counter (PC) onto the top of the *system* stack. The processor priority is then raised to the level of the active interrupt.

5. From the low memory vector address (calculated in step three above), a 32-bit *autovector* address is fetched and loaded into the program counter. This is an entry point into Exec's interrupt dispatcher.

6. Exec must now further decode the interrupt by examining the INTREQ and INTENA 4703 chip registers. Once the active interrupt has been determined, Exec indexes into an **ExecBase** array to fetch the interrupt's handler entry point and handler data pointer addresses.

7. Exec now turns control over to the interrupt handler by calling it as if it were a subroutine. This handler may deal with the interrupt directly or may propagate control further by invoking interrupt server chain processing.

You can see from the above discussion that the interrupt autovectors *should never be altered by the user.* If you wish to provide your own interrupt handler, you must use the Exec **SetIntVector()** function. Changing the content of any autovector location violates the design rules of the Multitasking Executive.

Task multiplexing usually occurs as the result of an interrupt. When an interrupt has finished and the processor is about to return to user mode, Exec determines whether task-scheduling attention is required. If a task was signaled during interrupt processing, the task scheduler will be invoked. Because Exec uses preemptive task scheduling, it can be said that the interrupt subsystem is the heart of task multiplexing. If, for some reason, interrupts do not occur, a task might execute forever because it cannot be forced to relinquish the CPU.

INTERRUPT PRIORITIES

Interrupts are prioritized in hardware and software. The 68000 CPU priority at which an interrupt executes is determined strictly by hardware. In addition to this, the software imposes a finer level of *pseudo-priorities* on interrupts with the same CPU priority. These pseudo-priorities determine the order in which simultaneous interrupts of the same CPU priority are processed. Multiple interrupts with the same CPU priority but a different pseudo-priority will not interrupt one another.

Table 5-1 summarizes all interrupts by priority.

Table 5-1: Interrupts by Priority

4703 Name	**CPU Priority**	**Pseudo Priority**	**Purpose**
NMI	7	15	Nonmaskable
INTEN	6	14	Special (Copper)
EXTER	6	13	8520B, external level 6
DSKSYNC	5	12	Disk byte
RBF	5	11	Serial input
AUD1	4	10	Audio channel 1
AUD3	4	9	Audio channel 3
AUD0	4	8	Audio channel 0
AUD2	4	7	Audio channel 2
BLIT	3	6	Blitter done
VERTB	3	5	Vertical blank
COPER	3	4	Copper
PORTS	2	3	8520A, external level 2
TBE	1	2	Serial output
DSKBLK	1	1	Disk block done
SOFTINT	1	0	Software interrupts

The 8520s (also called CIAs) are peripheral interface adapter chips. For more information about them, see *Amiga Hardware Reference Manual.*

As described in the Motorola 68000 programmer's manual, interrupts may nest only in the direction of higher priority. Because of the time-critical nature of many interrupts on the Amiga, the CPU priority level *must never be lowered* by user or system code. When the system is running in user mode (multitasking), the CPU priority level must remain set at zero. When an interrupt occurs, the CPU priority is raised to the level appropriate for that interrupt. Lowering the CPU priority would permit unlimited interrupt recursion on the system stack and would "short-circuit" the interrupt-priority scheme.

Because it is dangerous on the Amiga to hold off interrupts for any period of time, higher-level interrupt code must perform its business and exit promptly. If it is necessary to perform a time-consuming operation as the result of a high-priority interrupt, the operation should be deferred either by posting a *software interrupt* or by signalling a task. In this way, interrupt response time is kept to a minimum. Software interrupts are described in a later section.

NONMASKABLE INTERRUPT

The 68000 provides a nonmaskable interrupt (NMI) of CPU priority 7. Although this interrupt cannot be generated by the Amiga hardware itself, it can be generated on the expansion bus by external hardware. Because this interrupt does not pass through the 4703 interrupt controller circuitry, it is capable of violating system code critical sections. In particular, it short-circuits the **DISABLE** mutual-exclusion mechanism. Code that uses NMI must not assume that it can access system data structures.

Servicing Interrupts

Interrupts are serviced on the Amiga through the use of interrupt *handlers* and *servers*. An interrupt handler is a system routine that exclusively handles all processing related to a particular 4703 interrupt. An interrupt server is one of possibly many system routines that are invoked as the result of a single 4703 interrupt. Interrupt servers provide a means of interrupt sharing. This concept is useful for general-purpose interrupts such as vertical blanking.

At system start, Exec designates certain 4703 interrupts as handlers and others as server chains. The PORTS, COPER, VERTB, BLIT, EXTER, and NMI interrupts are initialized as server chains; hence, each of these may execute multiple interrupt routines per each interrupt. All other interrupts are designated as handlers and are always used exclusively.

DATA STRUCTURE

Interrupt handlers and servers are defined by the Exec **Interrupt** structure. This structure specifies an interrupt routine entry point and data pointer. The C definition of this structure is as follows:

```
struct Interrupt {
   struct  Node is_Node;
   APTR is_Data;
   VOID  (*is_Code)();
};
```

Once this structure has been properly initialized, it can be used for either a handler or a server.

ENVIRONMENT

Interrupts execute in an environment different from that of tasks. All interrupts execute in *supervisor mode* and utilize a single *system stack*. This stack is large enough to handle extreme cases of nested interrupts (of higher priorities). Obviously, interrupt processing has no effect on task stack usage.

All interrupt processing code, both handlers and servers, is invoked as assembly code subroutines. Normal assembly code CPU register conventions dictate that the D0, D1, A0, and A1 registers be free for scratch use. In the case of an interrupt handler, some of these registers also contain data that may be useful to the handler code. See the section on handlers below.

Because interrupt processing executes outside the context of most system activities, certain data structures will not be self-consistent and must be considered off limits for all practical purposes. This happens because certain system operations are not atomic in nature and may be interrupted only after executing part of an important instruction sequence. For example, memory allocation and deallocation routines forbid task switching but do not disable interrupts. This results in the finite possibility of interrupting a memory-related routine. In such a case, a memory linked list may be inconsistent when examined from the interrupt code itself. To avoid serious problems, the interrupt routine must not use any of the memory allocation or deallocation functions.

INTERRUPT HANDLERS

As described above, an interrupt handler is a system routine that exclusively handles all processing related to a particular 4703 interrupt. There can only be one handler per 4703 interrupt. Every interrupt handler consists of an **Interrupt** structure (as defined above) and a single assembly code routine. Optionally, a data structure pointer may also be provided. This is particularly useful for ROM-resident interrupt code.

An interrupt handler is passed control as if it were a subroutine of Exec. Once the handler has finished its business, it must return to Exec by executing an **RTS** (return from subroutine) instruction rather than an **RTE** (return from exception) instruction. Interrupt handlers should be kept very short to minimize service-time overhead and thus minimize the possibilities of interrupt overruns. As described above, an interrupt handler has the normal scratch registers at its disposal. In addition, A5 and A6 are free for use. These registers are saved by Exec as part of the interrupt initiation cycle.

For the sake of efficiency, Exec passes certain register parameters to the handler (see the list below). These register values may be utilized to trim a few microseconds off the execution time of a handler.

D0 is scratch and contains garbage.

D1 is scratch but contains the 4703 INTENAR and INTREQR registers values ANDed together. This results in an indication of which interrupts are enabled *and* active.

A0 points to the base address of the Amiga custom chips. This information is useful for performing indexed instruction access to the chip registers.

A1 points to the data area specified by the **is_Data** field of the **Interrupt** structure. Because this pointer is always fetched (regardless of whether you use it), it is to your advantage to make some use of it.

A5 is used as a vector to your interrupt code. It is free to be used as a scratch register, and it is not necessary to restore its value prior to returning.

A6 points to the Exec library base (SysBase). You may use this register to call Exec functions or set it up as a base register to access your own library or device. It is *not* necessary to restore this register prior to returning.

Interrupt handlers are established by passing the Exec function **SetIntVector()** your initialized **Interrupt** structure and the 4703 interrupt bit number of interest. The parameters for this function are as follows:

INTB_RBF
This is the bit number for which this interrupt server is to respond. Other possible bits for interrupts are defined in *hardware/intbits.h.*

RBFInterrupt
This is the address of an interrupt server node as described earlier in this chapter.

Keep in mind that certain interrupts are established as server chains and should not be accessed as handlers.

Here is a C code example of proper handler initialization and set-up:

```
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "hardware/custom.h"
#include "hardware/intbits.h"

extern void RBFHandler();
extern struct Interrupt *SetIntVector();
extern struct Custom custom;/* get base of custom chips */
struct Interrupt *RBFInterrupt, *PriorInterrupt;

/* To try this, user must have a 9600 baud terminal connected to serial
 * port and run this from a newcli window and must have a separate
 * way to view buffer contents as characters arrive.  The Wait(0) is
 * used merely to make this a runnable example demonstrating setting
 * the vector.  If the setup routine ever exits, various pointers
 * become invalid (Buffer, ln_Name) and there is no checking for
 * buffer overflow included either. */

main()
{
    setup();
}
setup()
{
    short  *Buffer;
    /* allocate an Interrupt node structure: */

    RBFInterrupt = (struct Interrupt *)
            AllocMem (sizeof(struct Interrupt), MEMF_PUBLIC);
    if (RBFInterrupt == 0) {
        printf ("not enough memory for interrupt handler");
        exit (100);
    }
    /* allocate an input buffer: */
    Buffer = (short *)AllocMem (512, MEMF_PUBLIC);
    if (Buffer == 0) {
        FreeMem (RBFInterrupt, sizeof(struct Interrupt));
        printf ("not enough memory for data buffer");
        exit (100);
    }
    printf("Address of buffer is: %lx\n", Buffer);

    /* initialize the Interrupt node: */
```

```
          RBFInterrupt->is_Node.ln_Type = NT_INTERRUPT;
          RBFInterrupt->is_Node.ln_Pri = 0;
          RBFInterrupt->is_Node.ln_Name = "RBF-example";
          RBFInterrupt->is_Data = (APTR)&Buffer;
          RBFInterrupt->is_Code = RBFHandler;

     /* enable the RBF interrupt if not already enabled */

          custom.intena = INTF_SETCLR | INTF_RBF;

          /* put the new interrupt handler into action: */
          PriorInterrupt = SetIntVector (INTB_RBF, RBFInterrupt);

          if (PriorInterrupt != 0) {
               printf ("we just replaced the %s interrupt handler",
                         PriorInterrupt->is_Node.ln_Name);
          }
     Wait(0);  /* wait forever, ("illustrative example only"... if it exits,
               * pointer to Buffer and ln_Name will become invalid) */
     }
```

In this example, note the correct initialization of the **Node** structure.

The external interrupt handler code used above, **RBFHandler**, grabs the input character from the serial port and stores it into the buffer. Notice that the address of the buffer is passed to the handler (shown below) via the **is_Data** pointer. This pointer is updated for every character stored.

```
     XDEF        _RBFHandler

_RBFHandler:
     MOVE.L      (A1),A5                    ;get buffer pointer
     MOVE.W      SERDATR(A0),(A5)+          ;store the input word
     MOVE.W      #INTF_RBF,INTREQ(A0)       ;clear the interrupt
     MOVE.L      A5,(A1)                    ;restore new buffer pointer
     RTS                                    ;return to exec
     END
```

In this example, the buffer holds complete 4703 serial data *words* that contain not only the input character, but special serial input flags as well (for example, data overrun). This data word is deposited directly into the buffer, and the 4703 RBF interrupt request is cleared. A more sophisticated example might perform various tests on the input word prior to storing it into the buffer.

INTERRUPT SERVERS

As mentioned above, an interrupt server is one of possibly many system interrupt routines that are invoked as the result of a single 4703 interrupt. Interrupt servers provide an essential mechanism for interrupt sharing.

Interrupt servers must be used for PORTS, COPER, VERTB, BLIT, EXTER, or NMI interrupts. For these interrupts, all servers are linked together in a chain. Every server in the chain will be called until one returns with the Z bit of the 68000's condition code register clear (indicating a non-zero result). If the interrupt was specifically for your server, you should return to Exec with the Z bit of the condition codes clear so that the whole chain does not have to be searched to find the interrupt. Note that VERTB servers (that is, servers that are bound to vertical blank) should *always* return with the Z bit set. Note that this is different from the normal calling convention (with the result in D0) to save time during time-critical interrupts.

The easiest way to set the condition code register is to do an immediate move to the D0 register as follows:

```
InterruptNotProcessed:
     MOVEQ #0,D0
     RTS

InterruptProcessed:
     MOVEQ #1,D0
     RTS
```

The same Exec **Interrupt** structure used for handlers is also used for servers. Also, like interrupt handlers, servers must terminate their code with an **RTS** instruction.

Interrupt servers are called in priority order. The priority of a server is specified in its **is_Node.ln_Pri** field. Higher-priority servers are called earlier than lower-priority servers. Adding and removing interrupt servers from a particular chain is accomplished with the Exec **AddIntServer()** and **RemIntServer()** functions. These functions require you to specify both the 4703 interrupt number and a properly initialized **Interrupt** structure.

Servers have different register values passed than handlers do. A server cannot count on the D0, D1, or A6 registers containing any useful information. A server is free to use D0-D1 and A0-A1/A5 as scratch.

In a server chain, the interrupt is cleared automatically by the system. Having a server clear its interrupt is not recommended and not necessary (clearing could cause the loss of an interrupt on PORTS or EXTER).

Here is an example of a program to set up and clean up a low-priority vertical blank interrupt server:

```
/* vertb.c */
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "hardware/custom.h"
#include "hardware/intbits.h"

struct Interrupt *VertBIntr;
long count;
/* To try this program, save as "vertb," then type "vertb" to run it.
 * If you type "run vertb," the program won't be connected to a CLI and
 * it will not be possible to send "q" to it to stop the program.
 * Compiling info: lc2 -v (disable stack checking so no need to use lc.lib)
 * Linking info: Astartup.obj, vertb.c, vb.obj, amiga.lib
 */
main()
{
   extern void VertBServer();

   /* allocate an Interrupt node structure: */
   VertBIntr = (struct Interrupt *)
         AllocMem (sizeof(struct Interrupt), MEMF_PUBLIC);
   if (VertBIntr == 0) {
      printf ("not enough memory for interrupt server");
      exit (100);
   }

   /* initialize the Interrupt node: */
   VertBIntr->is_Node.ln_Type = NT_INTERRUPT;
   VertBIntr->is_Node.ln_Pri = -60;
   VertBIntr->is_Node.ln_Name = "VertB-example";
   VertBIntr->is_Data = (APTR)&count;
   VertBIntr->is_Code = VertBServer;

   /* put the new interrupt server into action: */
   AddIntServer (INTB_VERTB, VertBIntr);
   printf("Type q to quit... reports how many vblanks since start\n");
```

```
        while (getchar () != 'q');    /* wait for user to type 'q' */

        RemIntServer (INTB_VERTB, VertBIntr);
        printf ("%ld vertical blanks occurred\n", count);
        FreeMem (VertBIntr, sizeof(struct Interrupt));
    }
```

The **VertBServer** might look something like this:

```
        XDEF        _VertBServer

    _VertBServer:
        MOVE.L      A1,A0       ;get address of count
        ADDQ.L      #1,(A0)     ;increment value of count
        MOVEQ.L     #0,D0       ;continue to process other vb-servers
        RTS
        END
```

Software Interrupts

Exec provides a means of generating *software interrupts.* This type of interrupt is useful for creating special-purpose asynchronous system contexts. Software interrupts execute at a priority higher than that of tasks but lower than that of hardware interrupts, so they are often used to defer hardware interrupt processing to a lower priority. Software interrupts use the same **Interrupt** data structure as hardware interrupts. As described above, this structure contains pointers to both interrupt code and data.

A software interrupt is usually activated with the **Cause()** function. If this function is called from a task, the task will be interrupted and the software interrupt will occur. If it is called from a hardware interrupt, the software interrupt will not be processed until the system exits from its last hardware interrupt. If a software interrupt occurs from within another software interrupt, it is not processed until the current one is completed.

Software interrupts are prioritized. Unlike interrupt servers, software interrupts have only five priority levels: -32, -16, 0, +16, and +32. The priority should be put into the **ln_Pri** field prior to calling **Cause()**.

Software interrupts can also be caused by message port arrival actions. See the "Messages and Ports" chapter.

Disabling Interrupts

As mentioned in the "Tasks" chapter, it is sometimes necessary to disable all interrupts when examining or modifying certain shared system data structures. Interrupt disabling is controlled with the **DISABLE** and **ENABLE** macros and the **Disable()** and **Enable()** C functions.

In some system code, there are nested disabled sections. This type of code requires that interrupts be disabled with the first **DISABLE** and not re-enabled until the *last* **ENABLE**. The system enable/disable macros and functions are designed to permit this sort of nesting. For example, if there is a section of system code that should not be interrupted, the **DISABLE** macro is used at the head and the **ENABLE** macro is used at the end.

Here is an assembly-code macro definition for **DISABLE**. This routine assumes that A6 holds a pointer to the base of the Exec library.

```
DISABLE     MACRO
            MOVE.W      #$4000,_intena
            ADDQ.B      #1,IDNestCnt(A6)
            ENDM
```

DISABLE increments a counter, **IDNestCnt**, that keeps track of how many levels of disable have been issued up to now. Only 126 levels of nesting are permitted. Notice that interrupts are disabled *before* the **IDNestCnt** variable is incremented.

Similarly, the **ENABLE** macro will reenable macros if the last disable level has just been exited:

```
ENABLE      MACRO
            SUBQ.B      #1,IDNestCnt(A6)
            BGE.S       ENABLE@
            MOVE.W      #$C000,_intena
ENABLE@:
            MEND
```

ENABLE decrements the same counter that **DISABLE** increments. Notice that interrupts are enabled *after* the **IDNestCnt** variable is decremented.

See the "Tasks" chapter for a better explanation of mutual exclusion using interrupt disabling.

Chapter 6

MEMORY ALLOCATION

Introduction

Areas of free memory are maintained as a special linked list of free regions. Each memory allocation function returns the starting address of a block of memory at least as large as the size that you requested to be allocated. Any memory that is linked into this system free list can be allocated by the memory allocation routines. The allocated memory is not tagged or initialized in any way unless you have specified, for example,

MEMF_CLEAR. Only the free memory area is tagged to reflect the size of the chunk that has been freed.

You should return allocated memory to the system when your task completes. As noted above, the system only keeps track of available system memory and has no idea which task may have allocated memory and not returned it to the system free list. If your program does not return allocated memory when its task exits, that memory is unavailable until the system is powered down or reset. This can be critical, especially when using graphics routines that often need large blocks of contiguous RAM space. Therefore, if you dynamically allocate RAM, make sure to return it to the system by using the **FreeMem()** or **FreeEntry()** routines described below.

When you ask for memory to be allocated, the system always allocates blocks of memory in even multiples of eight bytes. If you request more or less than eight bytes, your request is always rounded up to the nearest multiple of eight. In addition, the address at which the memory deallocation is made is always rounded down to the nearest even multiple of eight bytes.

Compatibility Note: Do not depend on this size! Future revisions of the system may require a different size to guarantee alignment of the requested area to a specific boundary. You *can* depend upon allocation being aligned to at least a longword boundary.

Using Memory Allocation Routines

Note: Do not attempt to allocate or deallocate system memory from within interrupt code. The "Interrupts" chapter explains that an interrupt may occur at any time, even during a memory allocation process. As a result, system data structures may not necessarily be internally consistent.

MEMORY REQUIREMENTS

You must tell the system about your memory requirements when requesting a chunk of memory. There are four memory requirement possibilities. Three of these tell where within the hardware address map memory is to be allocated. The fourth, **MEMF_CLEAR**, tells the allocator that this memory space is to be zeroed before the allocator returns the starting address of that space.

The memory requirements that you can specify are listed below:

MEMF_CHIP
: This indicates a memory block that is within the range that the special-purpose chips can access. As of this writing, this is the lowest 512K of the Amiga.

MEMF_FAST
: This indicates a memory block that is outside of the range that the special purpose chips can access. "FAST" means that the special-purpose chips cannot cause processor bus contention and therefore processor access will likely be faster. The special-purpose chips cannot use memory allocated in this way.

MEMF_PUBLIC
: This indicates that the memory requested is to be used for different tasks or interrupt code, such as task control blocks, messages, ports, and so on. The designation **MEMF_PUBLIC** should be used to assure compatibility with future versions of the system.

MEMF_CLEAR
: This indicates that memory is to be cleared before returning.

If no preferences are specified, **MEMF_FAST** is assumed first, then **MEMF_CHIP**.

MEMORY HANDLING ROUTINES

Exec has the following memory allocation routines:

AllocMem() and **FreeMem()**

These are system-wide memory allocation and deallocation routines. They use a memory free-list owned and managed by the system.

AllocEntry() and **FreeEntry()**

These are routines for allocating and freeing different-size and different-type memory blocks with a single call.

Allocate() and **Deallocate()**

These are routines that may be used within a user-task to locally manage a system-allocated memory block. You use these routines to manage memory yourself, using your own memory free lists.

SAMPLE CALLS FOR ALLOCATING SYSTEM MEMORY

The following examples show how to allocate memory.

```
APTR  mypointer,anotherptr;
mypointer = (APTR)AllocMem(100, 0);
```

AllocMem() returns the address of the first byte of a memory block that is at least 100 bytes in size or null if there is not that much free memory. Because the requirement field is specified as 0, memory will be allocated from any one of the system-managed memory regions.

```
anotherptr = (APTR)AllocMem(1000,MEMF_CHIP | MEMF_CLEAR);
```

Memory is allocated only out of chip-accessible memory; zeroes are filled into memory space before the address is returned. If the system free-list does not contain enough contiguous memory bytes in an area matching your requirements and of the size you have requested, **AllocMem()** or **Allocate()** returns a zero.

SAMPLE FUNCTION CALLS FOR FREEING SYSTEM MEMORY

The following examples free the memory chunks shown in the earlier call to the system allocation routines.

```
FreeMem(mypointer,100);

FreeMem(anotherptr,1000);
```

Note: Because of the internal operations of the allocator, your allocation request may result in an allocation larger than the number of bytes you requested in the first place. However, the **FreeMem()** routine adjusts the request to free memory in the same way as **AllocMem()** adjusts the size, thereby maintaining a consistent memory free-list.

The routine **FreeMem()** returns no status. However, if you attempt to free a memory block in the middle of a chunk that the system believes is already free, you will cause a system crash.

ALLOCATING MULTIPLE MEMORY BLOCKS

Exec provides the routines **AllocEntry()** and **FreeEntry()** to allocate multiple memory blocks in a single call. **AllocEntry()** accepts a data structure called a **MemList**, which contains the information about the size of the memory blocks to be allocated and the requirements, if any, that you have regarding the allocation. The **MemList** structure is found in the include file *exec/memory.h* and is defined as follows:

```
struct MemList {
     struct Node        ml_Node;
     UWORD              ml_NumEntries;    /* number of MemEntrys */
     struct MemEntry    ml_me[1];         /*where the MemEntrys begin*/
};
```

where:

Node
allows you to link together multiple **MemLists**. However, the node is ignored by the routines **AllocEntry()** and **FreeEntry()**.

ml_NumEntries
tells the system how many **MemEntry** sets are contained in this **MemList**. Notice that a **MemList** is a variable-length structure and can contain as many sets of entries as you wish.

The **MemEntry** structure looks like this:

```
struct MemEntry {
       union {
            ULONG         meu_Reqs;/* the AllocMem requirements */
            APTR          meu_Addr;/* address of your memory */
       }me_Un;
       ULONG  me_Length;      /* the size of this request */
};

#define me_Reqs me_Un.meu_Reqs
#define me_Addr me_Un.meu_Addr
```

Sample Code for Allocating Multiple Memory Blocks

```
#include "exec/types.h"
#include "exec/memory.h"

struct MemList *mymemlist;             /* pointer to a MemList */
       /* define new structure because C cannot initialize unions */
struct myneeds {
       struct MemList mn_head;         /* one entry in the header */
       struct MemEntry mn_body[3];     /* additional entries follow
                                        * directly as part of
                                        * same data structure */
} myneeds;

myneeds.mn_head.ml_NumEntries = 3;
myneeds.mn_body[0].me_Reqs = MEMF_PUBLIC;
myneeds.mn_body[0].me_Length = 104;
myneeds.mn_body[1].me_Reqs=MEMF_FAST|MEMF_CLEAR;
myneeds.mn_body[1].me_Length = 8000;
myneeds.mn_body[2].me_Reqs=MEMF_CHIP | MEMF_CLEAR;
myneeds.mn_body[2].me_Length = 256;

mymemlist = (struct MemList*)AllocEntry( &myneeds );

/* saying "struct MemEntry mn_body[3]" is simply
 * a way of adding extra MemEntry structures
 * contiguously at the end of the first such
 * structure at the end of the MemList.  Thus
 * members of the MemList of type MemEntry can
 * be referenced in C as additional members of
 * the "me[ ]" data structure.
 */
```

AllocEntry() returns a pointer to a new **MemList** of the same size as the **MemList** that you passed to it. For example, ROM code can provide a **MemList** containing the requirements of a task and create a RAM-resident copy of the list containing the addresses of the allocated entries.

Result of Allocating Multiple Memory Blocks

The **MemList** created by **AllocEntry()** contains **MemEntry** entries. **MemEntrys** are defined by a union statement, which allows one memory space to be defined in more than one way.

If **AllocEntry()** returns a value with bit 31 clear, then all of the **meu_Addr** positions in the returned **MemList** will contain valid memory addresses meeting the requirements you have provided.

To use this memory area, you would use code similar to the following:

```
struct    MemList *ml;
APTR      mydata, moredata;

if (((ml & (1<<31)) < 0)
{
     mydata = ml->ml_me[0].me_Addr;
     moremydata = ml->ml_me[1].me_Addr;
}
else
}
     exit (200);       /* error during AllocEntry */
{
```

If **AllocEntry()** has problems while trying to allocate the memory you have requested, instead of the address of a new **MemList**, it will return the memory requirements value with which it had the problem. Bit 31 of the value returned will be set, and no memory will be allocated. Entries in the list that were already allocated will be freed.

Memory Allocation and Tasks

If you want your task to cooperate fully with Exec, use the **MemList** and **AllocEntry()** facility to do your dynamic memory allocation.

In the task control block structure, there is a list header named **tc_MemEntry**. This is the list header that you initialize to point to the **MemLists** that your task has created by call(s) to **AllocEntry()**. Here is a short program segment that handles task memory list header initialization only. It assumes that you have already run **AllocEntry()** as shown in the simple **AllocEntry()** example above.

```
struct Task *tc;
struct MemList *ml;
NewList( tc->.tc_MemEntry );                /* Initialize the task's
                                             * memory list header */

AddTail( tc->.tc_MemEntry, ml );
```

Assuming that you have only used the **AllocEntry()** method (or **AllocMem()** and built your own custom **MemList**), your task now knows where to find the blocks of memory that your task has dynamically allocated. If your clean-up routine (the task's **finalPC** routine) finds items on the **tc_MemEntry** list when **RemTask(&mytask)** is executed, your routine can wind through all linked lists of **MemLists** and return all allocated memory to the system free-list.

MEMORY ALLOCATION AND MULTITASKING

To make sure that you are working effectively in the multitasking system as a cooperating task, you can do one of the following:

- o Globally allocate and free memory blocks by using **AllocMem()** and **FreeMem()**, adding each block when allocated and deleting each when it is freed.

- o Allocate one or more blocks of memory from the system global pool using **AllocEntry()** when your task begins and then manage those blocks internally using **Allocate()** and **Deallocate()**.

MANAGING MEMORY WITH ALLOCATE() AND DEALLOCATE()

Allocate() and **Deallocate()** use a memory region header, called **MemHeader**, as part of the calling sequence. You can build your own local header to manage memory locally. This structure takes the form:

```
struct MemHeader {
        UWORD mh_Attributes;          /* characteristics */
        struct    MemChunk *mh_First;/* first free region */
        APTR    mh_Lower;             /* lower memory bounds */
        APTR    mh_Upper;             /* upper memory bounds + 1 */
        ULONG mh_Free;                /* number of free bytes */
};
```

where

mh_Attributes
: is ignored by **Allocate()** and **Deallocate()**.

mh_First
: is the pointer to the first **MemChunk** structure.

mh_Lower
: is the lowest address within the memory block. This must be a multiple of eight bytes.

mh_Upper
: is the highest address within the memory block + 1. The highest address will itself be a multiple of eight if the block was allocated to you by **AllocMem()**.

mh_Free
: is the total free space.

This structure is included in the include-files *exec/memory.h* and *exec/memory.i*.

The following sample program fragment shows the correct initialization of a **MemHeader** structure. It assumes that you wish to allocate a block of memory from the global pool and thereafter manage it yourself using **Allocate()** and **Deallocate()**.

```
struct MemHeader memheader;
APTR myblock;

struct MemChunk {
     struct MemChunk *mc_Next;
     ULONG mc_bytes;
};

struct MemChunk *mc;

/* get a block from the system */
myblock = (APTR) AllocMem( 8000, MEMF_PUBLIC | MEMF_CLEAR );

memheader.mh_Lower   = myblock;
memheader.mh_First   = (ULONG) myblock;
memheader.mh_Upper   = (ULONG)myblock + 8000;

/* takes 8 bytes for the memory chunk headers that tag free memory */
memheader.mh_Free    = 8000 - (sizeof (struct MemChunk) );

/* initialize the free memory list */
mc = (struct MemChunk *) myblock;
mc->mc_Next = NULL;
mc->mc_Size = memheader.mh_Free;

/* now mymemhead is ready to use with
 * calls to Allocate( &memheader, size );
 * or Deallocate( &memheader, size ); */
```

Note that only free memory is "tagged" using a **MemChunk** linked list. Once memory is allocated, the system has no way of determining which task now has control of that memory.

If you allocate a large chunk from the system, you can assure that in your **finalPC** routine (specified when you perform **AddTask()**) you deallocate this large chunk as your task exits. Thus, local memory allocation and deallocation from a single large block can perhaps save some bookkeeping—that which might have been required if you had extensively used **AllocMem()** and **FreeMem()** instead. This can most easily be done by recording the allocated block in your task's **tc_MemEntry** structure.

Chapter 7

LIBRARIES

Using a properly designed machine code interface, it is possible to call any of the system routines without knowing in advance its absolute location in the system. This chapter shows how libraries are designed and used but does not cover the internal library structure. For more information, see the "Library Base Offsets" appendix.

What Is a Library?

A library is a collection of jump instructions, a system library node, and a data segment. System library conventions require that each code vector occupy six bytes. The size and content of a library node is specified below in the topic titled "Structure of a Library Node." The size of the data segment varies, depending on the needs of the library itself.

How To Access a Library

You must perform two steps to access a library that is already initialized. First, you must open the library. Second, you must access the jump instructions or data by specifying an offset (negative or positive) from the library base pointer returned by **OpenLibrary()**. This form of indirection allows you to develop code that is not dependent on the absolute locations of the system routines. Note that in the same release of an Exec kernel, it is possible that routines can have different addresses. This depends, for example, on whether the hardware options are different or if the user asks for a different configuration. Therefore, accessing the system routines through library calls is the most expedient way of assuring that your code will work on different machines.

OPENING A LIBRARY

You prepare a library for use by calling the routine **OpenLibrary()**. This call takes the form

```
LibPtr = OpenLibrary(LibName, Version)
  D0                    A1       D0
```

where

LibPtr
: is a pointer value that is nonzero if the requested library has been located. Be sure to check that the returned value is nonzero *before* attempting to use **LibPtr**. If it is zero, the open failed.

LibName
: is a pointer to a string variable (null-terminated) that contains the name of the library that you wish to open.

Version
is the version number of the library that you expect to use. Libraries of the same name will be compatible with previous versions. However, if you specify a newer version than is present, the open will fail. Use the value 0 if you simply want "any" version of the named library.

The routine **OpenLibrary()** causes the system to search for a library of that name within the system library list. If such an entry is found, the library's open-entry routine is called. If the library is not currently RAM-resident, AmigaDOS will search the directory currently assigned to LIBS:. If that library is present, it will be loaded, initialized, and added to the system library list. If the library allows you access, the library pointer will be returned in **LibPtr**.

USING A LIBRARY TO CALL A ROUTINE

A typical way to use the library interface once a library has been opened is to use assembly language code as follows. Note that this save/restore is necessary only if A6 does not already contain the correct value.

```
move.l   A6,-(SP)                   ;save current contents of A6
move.l   <libptr>,A6                ;move library pointer into A6
jsr      _LVO<routineName>(A6)      ;through library vector table
move.l   (SP)+,A6                   ;restore A6 to original value
```

The example above is the actual assembly code generated by the use of a machine language macro named LINKLIB:

```
LINKLIB functionOffset, libraryBase
```

where

functionOffset
is "_LVO" followed by the name of the routine as called from C.

libraryBase
is the address of the base of the library.

For example,

```
LINKLIB _LVODisplayBeep,IntuitionBase
```

produces the same code sequence as shown above. This macro is located in the file *exec/libraries.h.* Notice that it handles *only* the linkage to the routine. It does not save any registers or preload any registers for passing values to the routine. Negative offsets, in multiples of six bytes, access the code vectors within the library.

By convention, A6 must contain the library pointer when a library routine is called. This allows any library routine to locate the library and access its data or any of its other entry points. Registers A0, A1, D0, and D1 may be used as scratch registers by any routine. All other registers, both address and data, if used in a routine, should be saved and restored before exit.

USING A LIBRARY TO REFERENCE DATA

You can use the **LibPtr** to reference a data segment associated with a library by specifying a positive offset from **LibPtr**, such as:

```
move.l    <libptr>,A1          ; Move library base
move.l    <offset>(A1),D0      ; Retrieve data located at <offset>
```

Library data is not usually accessed directly from outside of a library, but rather is accessed by the routines that are part of the library itself. The sample code retrieves data specifically associated with that library. Note that different languages have different interface requirements. This example shows only a typical assembly language interface. When you design your own libraries, you may decide how the associated data segment is to be used. The system itself places no restrictions on its use.

CACHING LIBRARY POINTERS

To make your library calls more efficient, you may cache various pointers if you wish. These pointers are are the **libPtr** itself (because the library node, while it is open, may not be moved) and the address within the library at which a jump instruction is located (because offsets from the **libPtr** do not change). You should not, however, cache the jump vector from within the library. You will always expect to be calling the current library routine and therefore should not cache the jump vector.

CLOSING A LIBRARY

When your task has finished using a specific library, your program should call the routine **CloseLibrary()**. This call takes the form:

```
CloseLibrary(libPtr)
             A1
```

where **libPtr** is the value returned to you by the call to **OpenLibrary()**.

You close a library to tell the library manager that there is one fewer task currently using that library. If there are no tasks using a library, it is possible for the system, on request, to purge that library and free up the memory resources it is currently using. Each successful open should be matched by exactly one close. Do not attempt to use a library pointer after you have closed that library.

Adding a Library

You can add your own library to the system library list, provided that it is constructed as indicated below. You add a library to the system by using the **AddLibrary()** function. The format of the call to this function is as follows:

```
AddLibrary(libPtr)
           A1
```

This command links a new library to the system and makes it available to all tasks.

MAKING A NEW LIBRARY

A function called **MakeLibrary()** is a convenient way for you to construct a library. After running **MakeLibrary()**, you will normally add that library to the system library list.

```
libAddr = MakeLibrary(vectors, structure, init, dataSize, SegList)
  D0                  A0       A1         A2    D0        D1
AddLibrary(libAddr)
           A1
```

MakeLibrary() allocates space for the code vectors and data area, initializes the library node, and initializes the data area according to your specifications. Its parameters have the following meanings:

vectors
This is a pointer to a table of code pointers terminated with a **-1**. **vectors** must specify a valid table address.

structure
This parameter points to the base of an **InitStruct()** data region. That is, it points to the first location within a table that the **InitStruct()** routine can use to initialize various memory areas. **InitStruct()** will typically be used to initialize the data segment of the library, perhaps forming data tables, task control blocks, I/O control blocks, etc. If this entry is a 0, then **InitStruct()** is not called.

init
This parameter points to a routine that is to be executed after the library node has been allocated and the code and data areas have been initialized. When this routine is called, the **libAddr** (address of this library) is placed into data register D0. If **init** is zero, no init routine is called.

dataSize
This variable specifies the size of the data area to be reserved for the library. It includes the standard library node data as well as the reserved data area itself.

SegList
This is a pointer to the AmigaDOS memory segment list (for libraries loaded by DOS).

MINIMUM SUBSET OF LIBRARY CODE VECTORS

The code vectors of a library must include at least the following entries: OPEN, CLOSE, EXPUNGE, and one reserved entry.

OPEN	is the entry point called when you use the command **OpenLibrary()**. In the system libraries, OPEN increments the library variable **OpenCnt**. This variable is also used by CLOSE and EXPUNGE.
CLOSE	is the entry point called when you use the command **CloseLibrary()**. It decrements the library variable **OpenCnt** and may do a delayed EXPUNGE.

EXPUNGE

prepares the library for removal from the system. This often includes deallocating memory resources that were reserved during initialization. EXPUNGE not only frees the memory allocated for data structures, but also the areas reserved for the library node itself.

The remaining vector is reserved for future use. It should always return zero.

STRUCTURE OF A LIBRARY NODE

A library node contains all of the information that the system needs to manage a library. Here is the library structure as it appears in the *exec/libraries.h* include file:

```
struct Library {
        struct    Node libNode;       /* link into the system library list */
        UBYTE     lib_Flags;          /* flag variables */
        UBYTE     lib_pad;            /* unused */
        UWORD     lib_NegSize;        /* size of jump vectors in bytes. */
        UWORD     lib_PosSize;        /* data size */
        UWORD     lib_Version;
        UWORD     lib_Revision;
        APTR      lib_IdString;
        ULONG     lib_Sum;            /* checksum */
        UWORD     lib_Open_Cnt;       /* count how many tasks
                                       * have this library open */
};

                                        /* meaning of the flag bits: */

#define LIBF_SUMMING (1 << 0)           /* bit position says some task
                                         * is currently running a
                                         * checksum on this library */

#define LIBF_CHANGED (1 << 1)           /* bit position says one or more entries
                                         * have been changed in the library
                                         * code vectors used by SumLibrary */

#define LIBF_SUMUSED (1 << 2)           /* bit position says user wants a check-
                                         * sum fault to cause a system panic */

#define LIBF_DELEXP (1 << 3)            /* says there is a delayed expunge.
                                         * Some user has requested expunge but
                                         * another user still has the library open. */
```

CHANGING THE CONTENTS OF A LIBRARY

After a library has been constructed and linked to the system library list, you can use the routine **SetFunction()** either to add or to replace the contents of one of the library vectors. The format of this routine is as follows:

```
SetFunction( Library, FuncOffset, FuncEntry)
                A1        A0          D0
```

where

Library
: is a pointer to the library in which a function entry is to be changed.

FuncOffset
: is the offset (negative) at which the entry to be changed is located.

FuncEntry
: is a longword value that is the absolute address of the routine that is to be inserted at the selected position in the library code vectors.

When you use **SetFunction()** to modify a function entry in a library, it automatically recalculates the checksum of the library.

Relationship of Libraries to Devices

A device is an interface specification and an internal data structure based on the library structure. The interface specification defines a means of device control. The structures of libraries and devices are so similar that the routine **MakeLibrary()** is used to construct both libraries and devices. Devices require the same basic four code vectors but have additional code vectors that must be located in specific positions in the code vector table. The functions that devices are expected to perform, at minimum, are shown in chapter 4, "Input/Output." Also, a skeleton device (source code) is provided in the "Skeleton Device/Library Code" appendix of the *Amiga ROM Kernel Reference Manual: Devices and Libraries.*

Chapter 8

ROM-WACK

Introduction

Wack is a keystroke-interactive bug exterminator used with Amiga hardware and software. *ROM-Wack* is a small, ROM-resident version primarily useful for system-crash data-structure examination. ROM-Wack's command syntax and display formats are identical to *Grand-Wack*, of which it is functionally a subset. Grand-Wack includes both the ROM-resident and the remote versions of Wack.

Getting to Wack

ROM-Wack will be invoked by Exec automatically upon a fatal system error, or it can be explicitly invoked through the Exec **Debug()** function. Once invoked, communication is performed through the RS-232-C serial data port at 9600 baud.

When a fatal system error occurs, Wack can be used to examine memory in an attempt to locate the source of the failure. The state of the machine will be frozen at the point in which the error occurred and Wack will not disturb the state of system beyond using a small amount of supervisor stack, memory between 200 and 400 hex, and the serial data port.

A program may explicitly invoke Wack by calling the Exec **Debug()** function. This is useful during the debug phase of development for establishing program breakpoints. For future compatibility, **Debug** should be called with a single, null parameter—for example, **Debug(0)**. Please note however, that calling the **Debug()** function does not necessarily invoke ROM-Wack. If Grand-Wack or a user supplied debugger has been installed, it will be invoked in place of ROM-Wack.

When Wack is called from a program, system interrupts continue to process, but multitasking is disabled. Generally this is not harmful to the system. Your graphics will still display, keys may be typed, the mouse can be moved, and so on. However, many interrupts deposit raw data into bounded or circular buffers. These interrupts often signal related device tasks to further process these buffers. If too many interrupts occur, device buffers may begin to overflow or wrap around. You should limit the number of interrupt actions (typing keys on the Amiga keyboard for example) you perform while executing in Wack.

Finally, certain system failures are so serious that the system is forced to reboot. Before rebooting takes place, the power LED will flash slowly. If you type a Del character (hex 7F) while the LED is flashing, the system will enter Wack before rebooting.

Keystrokes, Numbers, and Symbols

Wack performs a function upon every keyboard keystroke. In ROM-Wack, these functions are permanently bound to certain keys. For example, typing ">" will immediately result in the execution of the **next-word** function. This type of operation gives a "keystroke-interactive" feel to most of the common Wack commands.

Whenever a key is pressed, it is mapped through a *KeyMap*, which translates it into an action. A key can have different meanings in different contexts. For simplicity, ROM-Wack applies keys consistently in all contexts (the Grand-Wack feature of arbitrary key binding is not available in ROM-Wack).

In the default keymap, most punctuation marks are bound to simple actions, such as displaying a memory frame, moving the frame pointer, or altering a single word. These actions are always performed immediately. In contrast, the keys A-Z, a-z, and 0-9 are bound to a function that collects the keys as a string. When such a string is terminated with <RETURN>, the keys are interpreted as a single *symbol* or *number*.

In ROM-Wack, symbols are treated only as *intrinsic functions*. Macros, constants, offsets, and bases are not supported. Hence, typing a symbol name will always result in the invocation of the symbol's statically bound function.

If a string of keys forms a number, that number is treated as a hexadecimal value. If a string of keys is neither a number nor a known symbol, the message "unknown symbol" is presented.

During the "collection" of a symbol or number string, typing a backspace deletes the previous character. Typing <CTRL-X> deletes the entire line.

Register Frame

When Wack is invoked for any reason, a *register frame* is displayed:

```
ROM-Wack
PC:  F00AB4 SR: 0000 USP: 001268 SSP: 07FFE8 TRAP: 0000 TASK: 0008B8
DR:  00000001  00000004  0000000C 00000AB4 00000001  0000001C 00000914  00000914
AR:  00000AB4  00F0D348 00011A80 00000B9C 00F20770  00F20380  00000604
SF:  0000 00F0 0AB4 0014 00F0 0AB4 0014 00F0 0AB4 0004 00F0 0AB4 0000 0004 0000
```

This frame displays the current processor state and system context from which you entered Wack. If you are familiar with the M68000 processor, most of this frame should be obvious: USP for user stack pointer, SSP for system stack pointer, etc.

The TRAP field indicates the trap number that forced us into Wack. Motorola uses the term *exceptions* for these traps. In Exec, the term *exception* is used for asynchronous task events. The standard TRAP numbers are

0	normal entry
2	bus error
3	address error
4	illegal instruction
5	zero divide
6	CHK instruction (should not happen)
7	TRAPV instruction (should not happen)
8	privilege violation
9	trace (single step)
A	line 1010 emulator
B	line 1111 emulator
2N	trap instruction N (2F normally for breakpoint)

The TASK field indicates the task from which the system entered Wack. If this field is zero, the system entered Wack from supervisor mode.

The SF line provides a backtrace of the current stack frame. This is often useful for determining the current execution context (last function called, for example). The user stack is displayed for entry from a task; the system stack for entry from supervisor mode. (*Note*: Version 25.1 always shows the system stack, never the user stack. This will change.)

Display Frames

Wack displays memory in fixed size *frames.* A frame may vary in size from 0 to 64K bytes. Frames normally show addresses, word size hex data, and ASCII equivalent characters:

```
F000C4 6578 6563 2E6C 6962 7261 7279 0000 4AFC  e x e c . l i b r a r y...
F000D4 00F0 00D2 00F0 2918 0019 0978 00F0 00C4 ............ )^X..^Y^I x...
```

By default, Wack will pack as much memory content as it can onto a single line. Sometimes it is preferable to see more or less than this default frame size. The frame size may be modified with **:n**. Here "n" represents the number of bytes (rounded to the next unit size) that will be displayed.

```
:4
F000C4 6578 6563  e x e c
:20
F000C4 6578 6563 2E6C 6962 7261 7279 0000 4AFC  e x e c . l i b r a r y...
F000D4 00F0 00D2 00F0 2918 0019 0978 00F0 00C4 ............ )^X..^Y^I.....
```

A ":0" frame size is useful for altering the write-only custom chip registers.

Relative Positioning

Wack functions as a *memory editor*; nearly all commands are performed relative to your current position in memory. The following commands cause relative movement:

.	forward a frame
,	backward a frame
>	forward a word
<	backward a word
+n	forward n bytes
-n	backward n bytes
<RETURN>	redisplay current frame
<SPACE>	forward a word

<BKSP>
  backward a word

An example of the use of these commands is provided below:

```
< RETURN >
f00200 7072 6573 656e 7429 0d0a 0000 2028 6372  p r e s e n t )^M^J...
.
f00210 6173 6820 2d20 6361 6e6e 6f74 2072 6563  a s h   -   c a n n o t
,
f00200 7072 6573 656e 7429 0d0a 0000 2028 6372  p r e s e n t )^M^J...
>
f00202 6573 656e 7429 0d0a 0000 2028 6372 6173  e s e n t )^M^J....   (
<
f00200 7072 6573 656e 7429 0d0a 0000 2028 6372  p r e s e n t )^M^J...
+24
f00224 290d 0a00 2028 626f 6f74 2064 6576 6963  )^M^J..   ( b o o t   d
-38
f001ec 6c65 290d 0a00 2028 6e6f 2064 6562 7567  l e )^M^J..   ( n o   d
```

Absolute Positioning

There are a few commands that perform absolute positioning. Typing a hex number moves you to that position in memory:

```
10ec
0010ec 00f0 17c0 4ef9 00f0 179a 4ef9 00f0 1786 ....^W..N......^W..
```

Also, Wack maintains an indirection stack to help you walk down linked lists of absolute pointers:

```
4
000004 0000 11ec 00f0 0a8e 00f0 0a90 00f0 0a92 ....^Q......^J......^J.....
[      (use current longword as the next address)
0011ec 0000 18f6 0000 1332 0900 00f0 086a 0000 ....^X......^S 2^I.........
]      (return to the previous "indirected" address)
000004 0000 11ec 00f0 0a8e 00f0 0a90 00f0 0a92 ....^Q......^J......^J.....
```

The **find** command finds a given pattern in memory, and the **limit** command determines the upper bound of the search. The pattern may be from one to four bytes in length. The pattern is not affected by the alignment of memory; that is, byte alignment is used for all searches regardless of the pattern size.

To set the upper bound for a **find** command, type an address followed by **limit** or **^** . The default bound is 1000000 hex.

Altering Memory

The = command lets you modify your current memory word:

```
20134
020134 0000 0000 0000 ............
020134 0000 = 767
020134 0767 0000 0000 ^G g........
```

If the frame size is zero, the contents of the word will not be displayed prior to your modification of that word:

```
:0
dff09c
DFF09C xxxx = 7fff
```

If you decide not to modify the contents after typing a =, press **<RETURN>** without typing a number. If you have already typed a number, type **<CTRL-X>**.

The **alter** command performs a repeated = which is handy for setting up tables. While in this mode, the > and < will move you forward or backward one word. To exit from this mode, type a **<RETURN>** with no preceding number.

```
alter
001400 0280 = 222
001402 00C8 = <
001400 0222 = 333
001402 00C8 = 444
001404 0000 = 0
001406 3700 = >
001408 0000 = 666
00140A 0000 = < RETURN >
```

You can modify registers when single-stepping or breakpointing. Typing ! followed by the register name (D0-D7, A0-A6), U) lets you make modifications. SR and SSP cannot be modified.

The **fill** command fills memory with a given pattern from the current location to an upper bound. The **limit** command determines the upper bound of the fill. The size of the fill pattern determines the number of bytes the pattern occupies in memory. For example, typing

```
fill <RETURN>
45
```

fills individual bytes with the value 45. Typing

```
fill <RETURN>
045
```

fills words, and

```
fill <RETURN>
0000045
```

fills longwords.

Caution: Using the **fill** command without properly setting the limit can destroy data in memory. To set the upper bound for a fill, type an address followed by **limit** or a ^ .

Execution Control

These commands control program execution and system reset:

go	execute from current address
resume	resume at current PC address
^D	resume at current PC address
^I (tab)	single instruction step
boot	reboot system (cold-reset)
ig	reboot system (cold-reset)

Breakpoints

ROM-Wack has the ability to perform limited program breakpoints. Up to 16 breakpoints may be set. The breakpoint commands are as follows:

set	set breakpoint at current address
clear	clear breakpoint at current address
show	show all breakpoint addresses
reset	clear all breakpoints

To set a breakpoint, position the address pointer to the break address and type **set**. Resume program execution with **go** or **resume**. When your breakpoint has been reached, Wack will display a register frame. The breakpoint is automatically cleared once the breakpoint is reached.

Returning to Multitasking After a Crash

The **user** command forces the machine back into multitasking mode after a crash that invoked ROM-Wack. This gives your system a chance to flush disk buffers before you reset, thus securing your disk's super-structures.

Once you type **user**, you cannot exit from ROM-Wack, so you should use this command only when you want to reboot after debugging. Give your disk a few seconds to write out its buffers. If your machine is in serious trouble, the **user** command may not work.

Appendix A

C EXEC INCLUDE FILES

This appendix contains the C-language include files that define the system data structures used by Exec routines.

This appendix is a printed copy of the Exec portion of the the *SYS:includes* directory on the Amiga C (Lattice C) disk.

```
#ifndef EXEC_ALERTS_H
#define EXEC_ALERTS_H
/*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*   $Header: alerts.h,v 1.0 85/08/28 15:05:44 carl Exp $
*
*   $Locker:  $
*
*********************************************************************/

#define SF_ALERTWACK (1<<1)   /* in ExecBase.SysFlag */


/*********************************************************************
*
*  Format of the alert error number:
*
*    +-+-------------+----------------+--------------------------------+
*    |D|  SubSysId   |  General Error |     SubSystem Specific Error   |
*    +-+-------------+----------------+--------------------------------+
*
*          D:  DeadEnd alert
*        SubSysId:  indicates ROM subsystem number.
*   General Error:  roughly indicates what the error was
*  Specific Error:  indicates more detail
**********************************************************************/

/*********************************************************************
*
*  General Dead-End Alerts
*
*********************************************************************/

/*------ alert types */
#define AT_DeadEnd    0x80000000
#define AT_Recovery   0x00000000

/*------ general purpose alert codes */
#define AG_NoMemory      0x00010000
#define AG_MakeLib       0x00020000
#define AG_OpenLib       0x00030000
#define AG_OpenDev       0x00040000
#define AG_OpenRes       0x00050000
#define AG_IOError       0x00060000

/*------ alert objects: */
#define AO_ExecLib       0x00008001
#define AO_GraphicsLib   0x00008002
#define AO_LayersLib     0x00008003
#define AO_Intuition     0x00008004
```

```
#define AO_MathLib       0x00008005
#define AO_CListLib      0x00008006
#define AO_DOSLib        0x00008007
#define AO_RAMLib        0x00008008
#define AO_IconLib       0x00008009
#define AO_AudioDev      0x00008010
#define AO_ConsoleDev    0x00008011
#define AO_GamePortDev   0x00008012
#define AO_KeyboardDev   0x00008013
#define AO_TrackDiskDev  0x00008014
#define AO_TimerDev      0x00008015
#define AO_CIARsrc       0x00008020
#define AO_DiskRsrc      0x00008021
#define AO_MiscRsrc      0x00008022
#define AO_BootStrap     0x00008030
#define AO_Workbench     0x00008031


/*********************************************************************
*
*   Specific Dead-End Alerts
*
*********************************************************************/

/*------ exec.library */
#define AN_ExecLib       0x01000000
#define AN_ExcptVect     0x81000001  /* 68000 exception vector checksum */
#define AN_BaseChkSum    0x81000002  /* execbase checksum */
#define AN_LibChkSum     0x81000003  /* library checksum failure */
#define AN_LibMem        0x81000004  /* no memory to make library */
#define AN_MemCorrupt    0x81000005  /* corrupted memory list */
#define AN_IntrMem       0x81000006  /* no memory for interrupt servers */
#define  AN_InitAPtr     0x81000007  /* InitStruct() of an APTR source */

/*------ graphics.library */
#define AN_GraphicsLib   0x02000000
#define AN_CopDisplay    0x82010001  /* copper display list, no memory */
#define AN_CopInstr      0x82010002  /* copper instruction list, no memory */
#define AN_CopListOver   0x82000003  /* copper list overload */
#define AN_CopIListOver  0x82000004  /* copper intermediate list overload */
#define AN_CopListHead   0x82010005  /* copper list head, no memory */
#define AN_LongFrame     0x82010006  /* long frame, no memory */
#define AN_ShortFrame    0x82010007  /* short frame, no memory */
#define AN_FloodFill     0x82010008  /* flood fill, no memory */
#define AN_TextTmpRas    0x02010009  /* text, no memory for TmpRas */
#define AN_BltBitMap     0x8201000A  /* BltBitMap, no memory */

/*------ layers.library */
#define AN_LayersLib     0x03000000

/*------ intuition.library */
#define AN_Intuition     0x04000000
#define AN_GadgetType    0x84000001  /* unknown gadet type */
#define AN_BadGadget     0x04000001  /* recovery form of AN_GadgetType */
#define AN_CreatePort    0x84010002  /* create port, no memory */
#define AN_ItemAlloc     0x04010003  /* item plane alloc, no memory */
```

```
#define AN_SubAlloc     0x04010004  /* sub alloc, no memory */
#define AN_PlaneAlloc   0x84010005  /* plane alloc, no memory */
#define AN_ItemBoxTop   0x84000006  /* item box top < RelZero */
#define AN_OpenScreen   0x84010007  /* open screen, no memory */
#define AN_OpenScrnRast 0x84010008  /* open screen, raster alloc, no memory */
#define AN_SysScrnType  0x84000009  /* open sys screen, unknown type */
#define AN_AddSWGadget  0x8401000A  /* add SW gadgets, no memory */
#define AN_OpenWindow   0x8401000B  /* open window, no memory */
#define  AN_BadState    0x8400000C  /* bad state return entering Intuition */
#define  AN_BadMessage  0x8400000D  /* bad message received by IDCMP */
#define  AN_WeirdEcho   0x8400000E  /* weird echo causing incomprehension */
#define  AN_NoConsole   0x8400000F  /* couldn't open the console device */


/*------ math.library */
#define AN_MathLib      0x05000000

/*------ clist.library */
#define AN_CListLib     0x06000000

/*------ dos.library */
#define AN_DOSLib       0x07000000
#define AN_StartMem     0x07010001  /* no memory at startup */
#define AN_EndTask      0x07000002  /* EndTask didn't */
#define AN_QPktFail     0x07000003  /* Qpkt failure */
#define AN_AsyncPkt     0x07000004  /* unexpected packet received */
#define AN_FreeVec      0x07000005  /* Freevec failed */
#define AN_DiskBlkSeq   0x07000006  /* disk block sequence error */
#define AN_BitMap       0x07000007  /* bitmap corrupt */
#define AN_KeyFree      0x07000008  /* key already free */
#define AN_BadChkSum    0x07000009  /* invalid checksum */
#define AN_DiskError    0x0700000A  /* disk Error */
#define AN_KeyRange     0x0700000B  /* key out of range */
#define AN_BadOverlay   0x0700000C  /* bad overlay */

/*------ ramlib.library */
#define AN_RAMLib       0x08000000

/*------ ramlib.library */
#define AN_IconLib      0x09000000

/*------ audio.device */
#define AN_AudioDev     0x10000000

/*------ console.device */
#define AN_ConsoleDev   0x11000000

/*------ gameport.device */
#define AN_GamePortDev  0x12000000

/*------ keyboard.device */
#define AN_KeyboardDev  0x13000000

/*------ trackdisk.device */
#define AN_TrackDiskDev 0x14000000
#define AN_TDCalibSeek  0x14000001  /* calibrate: seek error */
```

```
#define AN_TDDelay      0x14000002  /* delay: error on timer wait */

/*------ timer.device */
#define AN_TimerDev     0x15000000
#define AN_TMBadReq     0x15000001  /* bad request */

/*------ cia.resource */
#define AN_CIARsrc      0x20000000

/*------ disk.resource */
#define AN_DiskRsrc     0x21000000
#define AN_DRHasDisk    0x21000001  /* get unit: already has disk */
#define AN_DRIntNoAct   0x21000002  /* interrupt: no active unit */

/*------ misc.resource */
#define AN_MiscRsrc     0x22000000

/*------ bootstrap */
#define AN_BootStrap    0x30000000
#define AN_BootError    0x30000001  /* boot code returned an error */

/*------ Workbench */
#define AN_Workbench    0x31000000

#endif !EXEC_ALERTS_H
```

Jan 3 22:47 1986 exec/devices.h Page 1

```
#ifndef  EXEC_DEVICES_H
#define  EXEC_DEVICES_H
/*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*  $Header: devices.h,v 1.0 85/08/28 15:06:50 carl Exp $
*
*  $Locker:  $
*
********************************************************************/

#ifndef EXEC_LIBRARIES_H
#include "exec/libraries.h"
#endif !EXEC_LIBRARIES_H

#ifndef EXEC_PORTS_H
#include "exec/ports.h"
#endif !EXEC_PORTS_H


/****** Device *****************************************************/

struct Device {
    struct  Library dd_Library;
};


/****** Unit *******************************************************/

struct Unit {
    struct  MsgPort *unit_MsgPort;  /* queue for unprocessed messages */
    UBYTE   unit_flags;
    UBYTE   unit_pad;
    UWORD   unit_OpenCnt;     /* number of active opens */
};


#define UNITF_ACTIVE (1<<0)
#define UNITF_INTASK (1<<1)

#endif
```

Jan 3 22:47 1986 exec/errors.h Page 1

```
/*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*  $Header: errors.h,v 1.0 85/08/28 15:07:14 carl Exp $
*
*  $Locker:  $
*
********************************************************************/


#define  IOERR_OPENFAIL  -1 /* device/unit failed to open */
#define  IOERR_ABORTED   -2 /* request aborted */
#define  IOERR_NOCMD     -3 /* command not supported */
#define  IOERR_BADLENGTH -4 /* not a valid length */
```

```

#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/interrupts.h"
#include "exec/memory.h"
#include "exec/ports.h"
#include "exec/tasks.h"
#include "exec/libraries.h"
#include "exec/devices.h"
#include "exec/io.h"

```

```
#ifndef  EXEC_EXECBASE_H
#define  EXEC_EXECBASE_H
/*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*  $Header: execbase.h,v 1.1 85/11/12 16:10:26 carl Exp $
*
*  $Locker: carl $
*
********************************************************************/

#ifndef EXEC_LISTS_H
#include "exec/libraries.h"
#endif !EXEC_LISTS_H

#ifndef EXEC_INTERRUPTS_H
#include "exec/interrupts.h"
#endif !EXEC_INTERRUPTS_H

#ifndef EXEC_LIBRARIES_H
#include "exec/libraries.h"
#endif !EXEC_LIBRARIES_H

#ifndef EXEC_TASKS_H
#include "exec/tasks.h"
#endif !EXEC_TASKS_H


struct ExecBase {
   struct Library  LibNode;

   UWORD SoftVer;       /* kickstart release number    */
   WORD  LowMemChkSum;
   ULONG ChkBase;       /* system base pointer complement */
   APTR  ColdCapture;   /* coldstart soft vector    */
   APTR  CoolCapture;
   APTR  WarmCapture;
   APTR  SysStkUpper;   /* system stack base (upper bound) */
   APTR  SysStkLower;   /* top of system stack (lower bound) */
   ULONG MaxLocMem;
   APTR  DebugEntry;
   APTR  DebugData;
   APTR  AlertData;
   APTR  RsvdExt;

   UWORD ChkSum;

/****** Interrupt Related ***************************************/

   struct   IntVector IntVects[16];

```

Jan 3 22:47 1986 exec/execbase.h Page 2

```
/****** System Variables *****************************************/

   struct   Task *ThisTask;    /* pointer to current task */
   ULONG IdleCount;            /* idle counter */
   ULONG DispCount;            /* dispatch counter */
   UWORD Quantum;              /* time slice quantum */
   UWORD Elapsed;              /* current quantum ticks */
   UWORD SysFlags;             /* misc system flags */
   BYTE  IDNestCnt;            /* interrupt disable nesting count */
   BYTE  TDNestCnt;            /* task disable nesting count */

   UWORD AttnFlags;            /* special attention flags */
   UWORD AttnResched;          /* rescheduling attention */
   APTR  ResModules;           /* resident module array pointer */

   APTR  TaskTrapCode;
   APTR  TaskExceptCode;
   APTR  TaskExitCode;
   ULONG TaskSigAlloc;
   UWORD TaskTrapAlloc;


/****** System Lists *********************************************/

   struct   List MemList;
   struct   List ResourceList;
   struct   List DeviceList;
   struct   List IntrList;
   struct   List LibList;
   struct   List PortList;
   struct   List TaskReady;
   struct   List TaskWait;

   struct   SoftIntList SoftInts[5];

/****** Other Globals ********************************************/

   LONG  LastAlert[4];

   LONG  ExecBaseReserved[8];
};

#define SYSBASESIZE  sizeof( struct   ExecBase)

/****** AttnFlags ******/
/*  Processors and Co-processors: */
#define  AFB_68010   0  /* (will remain set for 68020 as well) */
#define  AFB_68020   1
#define  AFB_68881   4
#define  AFB_PAL     8   PAL/NTSC
#define  AFB_50HZ    9   Clock Rate
#endif
```

Jan 3 22:47 1986 exec/execname.h Page 1

```
/* Commodore-Amiga, Inc. */
#define EXECNAME "exec.library"
```

Apr 1 14:54 1986 exec/interrupts.h Page 1

```
#ifndef  EXEC_INTERRUPTS_H
#define  EXEC_INTERRUPTS_H
/*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*  $Header: interrupts.h,v 1.0 85/08/28 15:09:53 carl Exp $
*
*  $Locker:  $
*
*********************************************************************/

#ifndef EXEC_NODES_H
#include "exec/nodes.h"
#endif !EXEC_NODES_H

#ifndef EXEC_LISTS_H
#include "exec/lists.h"
#endif !EXEC_LISTS_H


struct Interrupt {
    struct  Node is_Node;
    APTR    is_Data;            /* server data segment  */
    VOID    (*is_Code)();       /* server code entry    */
};


struct IntVector {              /* For Exec use ONLY! */
    APTR    iv_Data;
    VOID    (*iv_Code)();
    struct  Node *iv_Node;
};


struct SoftIntList {            /* For Exec use ONLY! */
    struct  List sh_List;
    UWORD   sh_Pad;
};

#define SIH_PRIMASK (0xf0)

#endif
```

Jan 3 22:47 1986 exec/io.h Page 1

```
#ifndef  EXEC_IO_H
#define  EXEC_IO_H
/*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*  $Header: io.h,v 1.0 85/08/28 15:10:30 carl Exp $
*
*  $Locker:  $
*
*********************************************************************/

#ifndef EXEC_PORTS_H
#include "exec/ports.h"
#endif !EXEC_PORTS_H


struct IORequest {
    struct  Message io_Message;
    struct  Device  *io_Device;     /* device node pointer  */
    struct  Unit    *io_Unit;       /* unit (driver private)*/
    UWORD   io_Command;             /* device command */
    UBYTE   io_Flags;
    BYTE    io_Error;               /* error or warning num */
};

struct IOStdReq {
    struct  Message io_Message;
    struct  Device  *io_Device;     /* device node pointer  */
    struct  Unit    *io_Unit;       /* unit (driver private)*/
    UWORD   io_Command;             /* device command */
    UBYTE   io_Flags;
    BYTE    io_Error;               /* error or warning num */
    ULONG   io_Actual;              /* actual number of bytes transferred */
    ULONG   io_Length;              /* requested number bytes transferred*/
    APTR    io_Data;                /* points to data area */
    ULONG   io_Offset;              /* offset for block structured devices *
/*  ULONG   io_Reserved1;
    ULONG   io_Reserved2;
*/
};


#define IOB_QUICK 0
#define IOF_QUICK (1<<0)


#define CMD_INVALID  0
#define CMD_RESET    1
#define CMD_READ     2
#define CMD_WRITE    3
#define CMD_UPDATE   4
```

```
#define CMD_CLEAR    5
#define CMD_STOP     6
#define CMD_START    7
#define CMD_FLUSH    8

#define CMD_NONSTD   9

#endif
```

```
#ifndef  EXEC_LIBRARIES_H
#define  EXEC_LIBRARIES_H
/*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*   $Header: libraries.h,v 1.0 85/08/28 15:10:56 carl Exp $
*
*   $Locker:  $
*
*********************************************************************/

#ifndef EXEC_NODES_H
#include "exec/nodes.h"
#endif !EXEC_NODES_H


#define LIB_VECTSIZE    6
#define LIB_RESERVED    4
#define LIB_BASE        (-LIB_VECTSIZE)
#define LIB_USERDEF     (LIB_BASE-(LIB_RESERVED*LIB_VECTSIZE))
#define LIB_NONSTD      (LIB_USERDEF)

#define LIB_OPEN        (-6)
#define LIB_CLOSE       (-12)
#define LIB_EXPUNGE     (-18)
#define LIB_EXTFUNC     (-24)


extern struct Library {
    struct  Node lib_Node;
    UBYTE   lib_Flags;
    UBYTE   lib_pad;
    UWORD   lib_NegSize;        /* number of bytes before library */
    UWORD   lib_PosSize;        /* number of bytes after library */
    UWORD   lib_Version;
    UWORD   lib_Revision;
    APTR    lib_IdString;
    ULONG   lib_Sum;            /* the checksum itself */
    UWORD   lib_OpenCnt;        /* number of current opens */
};

#define LIBF_SUMMING (1<<0)     /* we are currently checksumming */
#define LIBF_CHANGED (1<<1)     /* we have just changed the lib */
#define LIBF_SUMUSED (1<<2)     /* set if we should bother to sum */
#define LIBF_DELEXP  (1<<3)     /* delayed expunge */

/* Temporary Compatibility */
#define lh_Node      lib_Node
#define lh_Flags   lib_Flags
#define lh_pad     lib_pad
#define lh_NegSize   lib_NegSize
```

Jan 3 22:47 1986 exec/libraries.h Page 2

```
#define lh_PosSize   lib_PosSize
#define lh_Version   lib_Version
#define lh_Revision  lib_Revision
#define lh_IdString  lib_IdString
#define lh_Sum    lib_Sum
#define lh_OpenCnt   lib_OpenCnt

#endif
```

Jan 3 22:47 1986 exec/lists.h Page 1

```
#ifndef  EXEC_LISTS_H
#define  EXEC_LISTS_H
/*******************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
********************************************************************
*
*   Source Control:
*
*  $Header: lists.h,v 1.0 85/08/28 15:11:23 carl Exp $
*
*  $Locker:  $
*
*******************************************************************/

#ifndef EXEC_NODES_H
#include "exec/nodes.h"
#endif !EXEC_NODES_H


struct List {
    struct  Node *lh_Head;
    struct  Node *lh_Tail;
    struct  Node *lh_TailPred;
    UBYTE   lh_Type;
    UBYTE   l_pad;
};

#endif
```

```
#ifndef  EXEC_MEMORY_H
#define  EXEC_MEMORY_H
/*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*  $Header: memory.h,v 1.0 85/08/28 15:11:49 carl Exp $
*
*  $Locker:  $
*
*********************************************************************/

#ifndef EXEC_NODES_H
#include "exec/nodes.h"
#endif !EXEC_NODES_H


/****** MemChunk ****************************************************/

struct  MemChunk {
    struct  MemChunk *mc_Next;    /* pointer to next chunk */
    ULONG   mc_Bytes;             /* chunk byte size     */
};


/****** MemHeader ***************************************************/

struct   MemHeader {
    struct  Node mh_Node;
    UWORD   mh_Attributes;        /* characteristics of this region */
    struct  MemChunk *mh_First;   /* first free region          */
    APTR    mh_Lower;             /* lower memory bound         */
    APTR    mh_Upper;             /* upper memory bound+1       */
    ULONG   mh_Free;              /* total number of free bytes */
};


/****** MemEntry ****************************************************/

struct   MemEntry {
union {
    ULONG   meu_Reqs;             /* the AllocMem requirements */
    APTR    meu_Addr;             /* the address of this memory region */
    } me_Un;
    ULONG   me_Length;            /* the length of this memory region */
};

#define me_un          me_Un      /* compatibility */
#define me_Reqs        me_Un.meu_Reqs
#define me_Addr        me_Un.meu_Addr


```

```
/****** MemList *****************************************************/

struct   MemList {
    struct  Node ml_Node;
    UWORD   ml_NumEntries;        /* number of entries in this struct */
    struct  MemEntry ml_ME[1];    /* the first entry   */
};

#define ml_me  ml_ME              /* compatibility */


/*----- Memory Requirement Types ---------------------------*/

#define MEMF_PUBLIC (1<<0)
#define MEMF_CHIP   (1<<1)
#define MEMF_FAST   (1<<2)

#define MEMF_CLEAR  (1<<16)
#define MEMF_LARGEST (1<<17)

#define MEM_BLOCKSIZE   8
#define MEM_BLOCKMASK   7

#endif
```

Jan 3 22:47 1986 exec/nodes.h Page 1

```
#ifndef  EXEC_NODES_H
#define  EXEC_NODES_H
/*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*  $Header: nodes.h,v 1.1 85/11/12 18:22:53 carl Exp $
*
*  $Locker:  $
*
*********************************************************************/


struct Node {
    struct  Node *ln_Succ;
    struct  Node *ln_Pred;
    UBYTE   ln_Type;
    BYTE    ln_Pri;
    char    *ln_Name;
};

/*----- Node Types --------*/
#define NT_UNKNOWN   0
#define NT_TASK      1
#define NT_INTERRUPT 2
#define NT_DEVICE    3
#define NT_MSGPORT   4
#define NT_MESSAGE   5
#define NT_FREEMSG   6
#define NT_REPLYMSG  7
#define NT_RESOURCE  8
#define NT_LIBRARY   9
#define NT_MEMORY    10
#define NT_SOFTINT   11
#define NT_FONT      12
#define NT_PROCESS   13
#define NT_SEMAPHORE 14

#endif
```

Apr 1 14:56 1986 exec/ports.h Page 1

```
#ifndef  EXEC_PORTS_H
#define  EXEC_PORTS_H
/*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*  $Header: ports.h,v 1.1 85/11/12 18:11:45 carl Exp $
*
*  $Locker:  $
*
*********************************************************************/

#ifndef EXEC_NODES_H
#include "exec/nodes.h"
#endif !EXEC_NODES_H

#ifndef EXEC_LISTS_H
#include "exec/lists.h"
#endif !EXEC_LISTS_H

#ifndef EXEC_TASKS_H
#include "exec/tasks.h"
#endif !EXEC_TASKS_H


/****** MsgPort *****************************************************/

struct MsgPort {
    struct  Node mp_Node;
    UBYTE   mp_Flags;
    UBYTE   mp_SigBit;              /* signal bit number    */
    struct  Task *mp_SigTask;       /* task to be signaled */
    struct  List mp_MsgList;        /* message linked list  */
};

#define mp_SoftInt mp_SigTask

#define PF_ACTION    3

#define PA_SIGNAL    0
#define PA_SOFTINT   1
#define PA_IGNORE    2


/****** Message *****************************************************/

struct Message {
    struct  Node mn_Node;
    struct  MsgPort *mn_ReplyPort; /* message reply port */
    UWORD   mn_Length;             /* message len in bytes */
};

```

```
/****** Semaphore ***************************************************/

struct Semaphore {
    struct MsgPort sm_MsgPort;
    WORD   sm_Bids;
};

#define  sm_LockMsg  mp_SigTask
#endif
```

```
#ifndef  EXEC_RESIDENT_H
#define  EXEC_RESIDENT_H
/*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*  $Header: resident.h,v 1.0 85/08/28 15:13:28 carl Exp $
*
*  $Locker:  $
*
*********************************************************************/

#ifndef EXEC_NODES_H
#include "exec/nodes.h"
#endif !EXEC_NODES_H

struct Resident {
    UWORD rt_MatchWord;     /* word to match on (ILLEGAL)     */
    struct Resident *rt_MatchTag; /* pointer to the above    */
    APTR  rt_EndSkip;       /* address to continue scan    */
    UBYTE rt_Flags;         /* various tag flags        */
    UBYTE rt_Version;       /* release version number  */
    UBYTE rt_Type;          /* type of module (NT_mumble)     */
    BYTE  rt_Pri;           /* initialization priority */
    char  *rt_Name;         /* pointer to node name     */
    char  *rt_IdString;     /* pointer to ident string    */
    APTR  rt_Init;          /* pointer to init code     */
};

#define  RTC_MATCHWORD  0x4AFC

#define  RTF_AUTOINIT   (1<<7)
#define RTF_COLDSTART   (1<<0)

/* Compatibility: */
#define  RTM_WHEN       3
#define  RTW_NEVER      0
#define  RTW_COLDSTART  1

#endif
```

```
#ifndef  EXEC_TASKS_H
#define  EXEC_TASKS_H
/*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*   $Header: tasks.h,v 1.0 85/08/28 15:14:19 carl Exp $
*
*   $Locker:  $
*
*********************************************************************/

#ifndef EXEC_NODES_H
#include "exec/nodes.h"
#endif !EXEC_NODES_H

#ifndef EXEC_LISTS_H
#include "exec/lists.h"
#endif !EXEC_LISTS_H


extern struct Task {
    struct  Node tc_Node;
    UBYTE   tc_Flags;
    UBYTE   tc_State;
    BYTE    tc_IDNestCnt;           /* intr disabled nesting*/
    BYTE    tc_TDNestCnt;           /* task disabled nesting*/
    ULONG   tc_SigAlloc;            /* sigs allocated */
    ULONG   tc_SigWait;             /* sigs we are waiting for */
    ULONG   tc_SigRecvd;            /* sigs we have received */
    ULONG   tc_SigExcept;           /* sigs we will take excepts for */
    UWORD   tc_TrapAlloc;           /* traps allocated */
    UWORD   tc_TrapAble;            /* traps enabled */
    APTR    tc_ExceptData;          /* points to except data */
    APTR    tc_ExceptCode;          /* points to except code */
    APTR    tc_TrapData;            /* points to trap code */
    APTR    tc_TrapCode;            /* points to trap data */
    APTR    tc_SPReg;               /* stack pointer      */
    APTR    tc_SPLower;             /* stack lower bound    */
    APTR    tc_SPUpper;             /* stack upper bound + 2*/
    VOID    (*tc_Switch) ();        /* task losing CPU    */
    VOID    (*tc_Launch) ();        /* task getting CPU  */
    struct  List tc_MemEntry;       /* allocated memory */
    APTR    tc_UserData;            /* per task data */
};

/*----- Flag Bits ------------------------------------------*/
#define TB_PROCTIME  0
#define TB_STACKCHK  4
#define TB_EXCEPT    5
#define TB_SWITCH    6
#define TB_LAUNCH    7
```

```

/*----- Task States ----------------------------------------*/
#define TS_INVALID   0
#define TS_ADDED     1
#define TS_RUN       2
#define TS_READY     3
#define TS_WAIT      4
#define TS_EXCEPT    5
#define TS_REMOVED   6

/*----- Predefined Signals ---------------------------------*/

#define SIGF_ABORT    (1<<0)
#define SIGF_CHILD    (1<<1)
#define SIGF_BLIT     (1<<4)
#define SIGF_DOS      (1<<8)

#endif

```

```
#ifndef  EXEC_TYPES_H
#define  EXEC_TYPES_H
/*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*  $Header: types.h,v 1.2 85/11/15 17:43:37 carl Exp $
*
*  $Locker:  $
*
*********************************************************************/


#define GLOBAL extern               /* the declaratory use of an external */
#define IMPORT extern               /* reference to an external */
#define STATIC static               /* a local static variable */
#define REGISTER register           /* a (hopefully) register variable */

#define VOID      void              /* typedef does not seem to work here */

typedef long             LONG;      /* signed 32-bit quantity */
typedef unsigned long    ULONG;     /* unsigned 32-bit quantity */
typedef unsigned long    LONGBITS;  /* 32 bits manipulated individually */
typedef short            WORD;      /* signed 16-bit quantity */
typedef unsigned short   UWORD;     /* unsigned 16-bit quantity */
typedef unsigned short   WORDBITS;  /* 16 bits manipulated individually */
typedef char             BYTE;      /* signed 8-bit quantity */
typedef unsigned char    UBYTE;     /* unsigned 8-bit quantity */
typedef unsigned char    BYTEBITS;  /* 8 bits manipulated individually */
typedef unsigned char    *STRPTR;   /* string pointer */
typedef STRPTR           *APTR;     /* absolute memory pointer */

/* For compatability only: (don't use in new code) */
typedef short            SHORT;     /* signed 16-bit quantity (WORD) */
typedef unsigned short   USHORT;    /* unsigned 16-bit quantity (UWORD) */


/* Types with specific semantics */
typedef float            FLOAT;
typedef double           DOUBLE;
typedef short            COUNT;
typedef unsigned short   UCOUNT;
typedef short            BOOL;
typedef unsigned char    TEXT;

#define TRUE      1
#define FALSE     0
#define NULL      0

#define BYTEMASK  0xFF

#define  LIBRARY_VERSION   31
```

```

#endif
```

```
    IFND EXEC_ABLES_I
EXEC_ABLES_I  SET 1
*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*   $Header: ables.i,v 1.0 85/08/28 15:05:30 carl Exp $
*
*   $Locker:  $
*
*********************************************************************

    IFND EXEC_TYPES_I
    INCLUDE "exec/types.i"
    ENDC !EXEC_TYPES_I

    IFND EXEC_EXECBASE_I
    INCLUDE "exec/execbase.i"
    ENDC !EXEC_EXECBASE_I


*----------------------------------------------------------------
*
*   Interrupt Exclusion Macros
*
*----------------------------------------------------------------

INT_ABLES   MACRO                       * externals for dis/enable
            XREF    _intena
            ENDM


DISABLE     MACRO   * [scratchReg]
        IFC     '\1',''
            MOVE.W  #$04000,_intena     *(NOT IF_SETCLR)+IF_INTEN
            ADDQ.B  #1,IDNestCnt(A6)
        ENDC
        IFNC    '\1',''
        MOVE.L  4,\1
            MOVE.W  #$04000,_intena     *(NOT IF_SETCLR)+IF_INTEN
            ADDQ.B  #1,IDNestCnt(\1)
        ENDC
            ENDM


ENABLE      MACRO   * [scratchReg]
        IFC     '\1',''
            SUBQ.B  #1,IDNestCnt(A6)
        BGE.S   ENABLE\@
            MOVE.W  #$0C000,_intena     *IF_SETCLR+IF_INTEN
ENABLE\@:
        ENDC
```

```
        IFNC    '\1',''
            MOVE.L  4,\1
            SUBQ.B  #1,IDNestCnt(\1)
            BGE.S   ENABLE\@
            MOVE.W  #$0C000,_intena
ENABLE\@:
        ENDC
            ENDM


*----------------------------------------------------------------
*
*   Tasking Exclusion Macros
*
*----------------------------------------------------------------

TASK_ABLES  MACRO
*       INCLUDE 'execbase.i' for TDNestCnt offset
            XREF    _LVOPermit
            ENDM


FORBID      MACRO
            ADDQ.B  #1,TDNestCnt(A6)
            ENDM


PERMIT      MACRO
            JSR     _LVOPermit(A6)
            ENDM

    ENDC !EXEC_ABLES_I
    IFND EXEC_ALERTS_I
EXEC_ALERTS_I SET 1
***********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
***********************************************************************
*
*   Source Control:
*
*  $Header: alerts.i,v 1.0 85/08/28 15:05:58 carl Exp $
*
*  $Locker:  $
*
***********************************************************************

    BITDEF  S,ALERTWACK,1  * in ExecBase.SysFlags


***********************************************************************
*
*  Format of the alert error number:
*
*    +---------------+---------------+-------------------------------+
*    |D|  SubSysId   |  General Error |    SubSystem Specific Error    |
*    +---------------+---------------+-------------------------------+
*
*          D:  DeadEnd alert
*       SubSysId:  indicates ROM subsystem number.
*    General Error:  roughly indicates what the error was
*   Specific Error:  indicates more detail
***********************************************************************

*
*  Use this macro for causing an alert.  THIS MACRO MAY CHANGE!
*  It is very sensitive to memory corruption.... like stepping on
*  location 4!  But it should work for now.
*
ALERT    macro    (alertNumber, paramArray, scratch)
      movem.l d7/a5/a6,-(sp)
      move.l   #\1,d7
      IFNC  '\2',''
      lea   \2,a5
      ENDC
      move.l  4,a6       ; (use proper name!!!)
      jsr   _LVOAlert(a6)
      movem.l  (sp)+,d7/a5/a6
      endm


***********************************************************************
*
*  General Dead-End Alerts
*
*  For example:  timer.device cannot open math.library:
*
```

```
*       ALERT  (AN_TimerDev!AG_OpenLib!AO_MathLib),(A0),A1
*
***********************************************************************

;------ alert types
AT_DeadEnd      equ $80000000
AT_Recovery     equ $00000000

;------ general purpose alert codes
AG_NoMemory     equ $00010000
AG_MakeLib      equ $00020000
AG_OpenLib      equ $00030000
AG_OpenDev      equ $00040000
AG_OpenRes      equ $00050000
AG_IOError      equ $00060000

;------ alert objects:
AO_ExecLib      equ $00008001
AO_GraphicsLib equ $00008002
AO_LayersLib    equ $00008003
AO_Intuition    equ $00008004
AO_MathLib      equ $00008005
AO_CListLib     equ $00008006
AO_DOSLib       equ $00008007
AO_RAMLib       equ $00008008
AO_IconLib      equ $00008009
AO_AudioDev     equ $00008010
AO_ConsoleDev   equ $00008011
AO_GamePortDev equ $00008012
AO_KeyboardDev equ $00008013
AO_TrackDiskDev equ $00008014
AO_TimerDev     equ $00008015
AO_CIARsrc      equ $00008020
AO_DiskRsrc     equ $00008021
AO_MiscRsrc     equ $00008022
AO_BootStrap    equ $00008030
AO_Workbench    equ $00008031


***********************************************************************
*
*   Specific Dead-End Alerts:
*
*   For example:   exec.library -- corrupted memory list
*
*      ALERT  AN_MemCorrupt,(A0),A1
*
***********************************************************************

;------ exec.library
AN_ExecLib      equ $01000000
AN_ExcptVect    equ $81000001    ; 68000 exception vector checksum
AN_BaseChkSum   equ $81000002   ; execbase checksum
AN_LibChkSum    equ $81000003   ; library checksum failure
AN_LibMem       equ $81000004   ; no memory to make library
AN_MemCorrupt   equ $81000005   ; corrupted memory list
```

```
AN_IntrMem      equ $81000006  ; no memory for interrupt servers
AN_InitAPtr     equ $81000007  ; InitStruct() of an APTR source

;------ graphics.library
AN_GraphicsLib equ $02000000
AN_CopDisplay  equ $82010001  ; copper display list, no memory
AN_CopInstr    equ $82010002  ; copper instruction list, no memory
AN_CopListOver equ $82000003  ; copper list overload
AN_CopIListOver equ $82000004 ; copper intermediate list overload
AN_CopListHead equ $82010005  ; copper list head, no memory
AN_LongFrame   equ $82010006  ; long frame, no memory
AN_ShortFrame  equ $82010007  ; short frame, no memory
AN_FloodFill   equ $82010008  ; flood fill, no memory
AN_TextTmpRas  equ $02010009  ; text, no memory for TmpRas
AN_BltBitMap   equ $8201000A  ; BltBitMap, no memory

;------ layers.library
AN_LayersLib   equ $03000000

;------ intuition.library
AN_Intuition   equ $04000000
AN_GadgetType  equ $84000001  ; unknown gadet type
AN_BadGadget   equ $04000001  ; cecovery form of AN_GadgetType
AN_CreatePort  equ $84010002  ; create port, no memory
AN_ItemAlloc   equ $04010003  ; item plane alloc, no memory
AN_SubAlloc    equ $04010004  ; sub alloc, no memory
AN_PlaneAlloc  equ $84010005  ; plane alloc, no memory
AN_ItemBoxTop  equ $84000006  ; item box top < RelZero
AN_OpenScreen  equ $84010007  ; open screen, no memory
AN_OpenScrnRast equ $84010008 ; open screen, raster alloc, no memory
AN_SysScrnType equ $84000009  ; open sys screen, unknown type
AN_AddSWGadget equ $8401000A  ; add SW gadgets, no memory
AN_OpenWindow  equ $8401000B  ; open window, no memory
AN_BadState    equ $8400000C  ; bad state return entering Intuition
AN_BadMessage  equ $8400000D  ; bad message received by IDCMP
AN_WeirdEcho   equ $8400000E  ; weird echo causing incomprehension
AN_NoConsole   equ $8400000F  ; couldn't open the console device

;------ math.library
AN_MathLib     equ $05000000

;------ clist.library
AN_CListLib    equ $06000000

;------ dos.library
AN_DOSLib      equ $07000000
AN_StartMem    equ $07010001  ; no memory at startup
AN_EndTask     equ $07000002  ; EndTask didn't
AN_QPktFail    equ $07000003  ; Qpkt failure
AN_AsyncPkt    equ $07000004  ; unexpected packet received
AN_FreeVec     equ $07000005  ; Freevec failed
AN_DiskBlkSeq  equ $07000006  ; disk block sequence error
AN_BitMap      equ $07000007  ; bitmap corrupt
AN_KeyFree     equ $07000008  ; key already free
AN_BadChkSum   equ $07000009  ; invalid checksum
AN_DiskError   equ $0700000A  ; disk error
```

```
AN_KeyRange    equ $0700000B  ; key out of range
AN_BadOverlay  equ $0700000C  ; bad overlay

;------ ramlib.library
AN_RAMLib      equ $08000000

;------ icon.library
AN_IconLib     equ $09000000

;------ audio.device
AN_AudioDev    equ $10000000

;------ console.device
AN_ConsoleDev  equ $11000000

;------ gameport.device
AN_GamePortDev equ $12000000

;------ keyboard.device
AN_KeyboardDev equ $13000000

;------ trackdisk.device
AN_TrackDiskDev equ $14000000
AN_TDCalibSeek equ $14000001  ; calibrate: seek error
AN_TDDelay     equ $14000002  ; delay: error on timer wait

;------  timer.device
AN_TimerDev    equ $15000000
AN_TMBadReq    equ $15000001  ; bad request

;------ cia.resource
AN_CIARsrc     equ $20000000

;------  disk.resource
AN_DiskRsrc    equ $21000000
AN_DRHasDisk   equ $21000001  : get unit: already has disk
AN_DRIntNoAct  equ $21000002  ; interrupt: no active unit

;------ misc.resource
AN_MiscRsrc    equ $22000000

;------ bootstrap
AN_BootStrap   equ $30000000
AN_BootError   equ $30000001  ; boot code returned an error

;------ workbench
AN_Workbench   equ $31000000

    ENDC !EXEC_ALERTS_I
```

Jan 3 22:47 1986 exec/devices.i Page 1

```
    IFND EXEC_DEVICES_I
EXEC_DEVICES_I SET 1
**********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
**********************************************************************
*
*   Source Control:
*
*  $Header: devices.i,v 1.0 85/08/28 15:07:02 carl Exp $
*
*  $Locker:  $
*
**********************************************************************

    IFND EXEC_LIBRARIES_I
    INCLUDE "exec/libraries.i"
    ENDC !EXEC_LIBRARIES_I

    IFND EXEC_PORTS_I
    INCLUDE "exec/ports.i"
    ENDC !EXEC_PORTS_I


*----------------------------------------------------------------
*
*   Device Data Structure
*
*----------------------------------------------------------------

 STRUCTURE  DD,LIB_SIZE
    LABEL   DD_SIZE             * identical to library


*----------------------------------------------------------------
*
*   Suggested Unit Structure
*
*-------------------------------- ------------------------------

 STRUCTURE  UNIT,MP_SIZE        * queue for requests
    UBYTE   UNIT_FLAGS
    UBYTE   UNIT_pad
    UWORD   UNIT_OPENCNT
    LABEL   UNIT_SIZE


*------ UNIT_FLAG definitions:

    BITDEF  UNIT,ACTIVE,0       * driver is active
    BITDEF  UNIT,INTASK,1       * running in driver's task

    ENDC !EXEC_DEVICES_I
```

Jan 3 22:47 1986 exec/errors.i Page 1

```
    IFND EXEC_ERRORS_I
EXEC_ERRORS_I SET 1
**********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
**********************************************************************
*
*   Source Control:
*
*  $Header: errors.i,v 1.0 85/08/28 15:07:26 carl Exp $
*
*  $Locker:  $
*
**********************************************************************

*------ Standard IO Errors:

IOERR_OPENFAIL  EQU  -1    * device/unit failed to open
IOERR_ABORTED   EQU  -2    * request aborted
IOERR_NOCMD EQU      -3    * command not supported
IOERR_BADLENGTH EQU  -4    * not a valid length


ERR_OPENDEVICE EQU  IOERR_OPENFAIL  * REMOVE !!!

    ENDC !EXEC_ERRORS_I
```

Jan 3 22:47 1986 exec/exec.i Page 1

```

    INCLUDE "exec/nodes.i"
    INCLUDE "exec/lists.i"
    INCLUDE "exec/interrupts.i"
    INCLUDE "exec/memory.i"
    INCLUDE "exec/ports.i"
    INCLUDE "exec/tasks.i"
    INCLUDE "exec/libraries.i"
    INCLUDE "exec/devices.i"
    INCLUDE "exec/io.i"

```

Jan 3 22:47 1986 exec/exec_lib.i Page 1

```
*** Commodore-Amiga, Inc.
*** This file generated on Tue Nov 12 16:50:46 1985
*** $Header: gen-lib.ml,v 2.5 85/10/08 18:09:47 carl Exp $
*** DO NOT EDIT: FILE BUILT AUTOMATICALLY
   FUNCDEF  Supervisor
   FUNCDEF  ExitIntr
   FUNCDEF  Schedule
   FUNCDEF  Reschedule
   FUNCDEF  Switch
   FUNCDEF  Dispatch
   FUNCDEF  Exception
   FUNCDEF  InitCode
   FUNCDEF  InitStruct
   FUNCDEF  MakeLibrary
   FUNCDEF  MakeFunctions
   FUNCDEF  FindResident
   FUNCDEF  InitResident
   FUNCDEF  Alert
   FUNCDEF  Debug
   FUNCDEF  Disable
   FUNCDEF  Enable
   FUNCDEF  Forbid
   FUNCDEF  Permit
   FUNCDEF  SetSR
   FUNCDEF  SuperState
   FUNCDEF  UserState
   FUNCDEF  SetIntVector
   FUNCDEF  AddIntServer
   FUNCDEF  RemIntServer
   FUNCDEF  Cause
   FUNCDEF  Allocate
   FUNCDEF  Deallocate
   FUNCDEF  AllocMem
   FUNCDEF  AllocAbs
   FUNCDEF  FreeMem
   FUNCDEF  AvailMem
   FUNCDEF  AllocEntry
   FUNCDEF  FreeEntry
   FUNCDEF  Insert
   FUNCDEF  AddHead
   FUNCDEF  AddTail
   FUNCDEF  Remove
   FUNCDEF  RemHead
   FUNCDEF  RemTail
   FUNCDEF  Enqueue
   FUNCDEF  FindName
   FUNCDEF  AddTask
   FUNCDEF  RemTask
   FUNCDEF  FindTask
   FUNCDEF  SetTaskPri
   FUNCDEF  SetSignal
   FUNCDEF  SetExcept
   FUNCDEF  Wait
   FUNCDEF  Signal
   FUNCDEF  AllocSignal
   FUNCDEF  FreeSignal
```

```
        FUNCDEF  AllocTrap
        FUNCDEF  FreeTrap
        FUNCDEF  AddPort
        FUNCDEF  RemPort
        FUNCDEF  PutMsg
        FUNCDEF  GetMsg
        FUNCDEF  ReplyMsg
        FUNCDEF  WaitPort
        FUNCDEF  FindPort
        FUNCDEF  AddLibrary
        FUNCDEF  RemLibrary
        FUNCDEF  OldOpenLibrary
        FUNCDEF  CloseLibrary
        FUNCDEF  SetFunction
        FUNCDEF  SumLibrary
        FUNCDEF  AddDevice
        FUNCDEF  RemDevice
        FUNCDEF  OpenDevice
        FUNCDEF  CloseDevice
        FUNCDEF  DoIO
        FUNCDEF  SendIO
        FUNCDEF  CheckIO
        FUNCDEF  WaitIO
        FUNCDEF  AbortIO
        FUNCDEF  AddResource
        FUNCDEF  RemResource
        FUNCDEF  OpenResource
        FUNCDEF  RawIOInit
        FUNCDEF  RawMayGetChar
        FUNCDEF  RawPutChar
        FUNCDEF  RawDoFmt
        FUNCDEF  GetCC
        FUNCDEF  TypeOfMem
        FUNCDEF  Procure
        FUNCDEF  Vacate
        FUNCDEF  OpenLibrary
```

```
    IFND EXEC_EXECBASE_I
EXEC_EXECBASE_I SET 1
**********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
**********************************************************************
*
*   Source Control:
*
*  $Header: execbase.i,v 1.1 85/11/12 16:10:51 carl Exp $
*
*  $Locker: carl $
*
**********************************************************************

    IFND EXEC_LISTS_I
    INCLUDE "exec/lists.i"
    ENDC !EXEC_LISTS_I

    IFND EXEC_INTERRUPTS_I
    INCLUDE "exec/interrupts.i"
    ENDC !EXEC_INTERRUPTS_I

    IFND EXEC_LIBRARIES_I
    INCLUDE "exec/libraries.i"
    ENDC !EXEC_LIBRARIES_I


******* Static System Variables *************************************

 STRUCTURE  ExecBase,LIB_SIZE     ; standard library node

            UWORD SoftVer          ; kickstart release number
            WORD  LowMemChkSum     ; checksum of 68000 trap vectors
            ULONG ChkBase          ; system base pointer complement
            APTR  ColdCapture      ; cold soft capture vector
            APTR  CoolCapture      ; cool soft capture vector
            APTR  WarmCapture      ; warm soft capture vector
            APTR  SysStkUpper      ; system stack base   (upper bound)
            APTR  SysStkLower      ; top of system stack (lower bound)
            ULONG MaxLocMem        ; last calculated local memory max
            APTR DebugEntry        ; global debugger entry point
            APTR DebugData         ; global debugger data segment
            APTR AlertData         ; alert data segment
            APTR RsvdExt           ; reserved

            WORD  ChkSum           ; for all of the above


******* Interrupt Related ********************************************

            LABEL  IntVects
            STRUCT   IVTBE,IV_SIZE
            STRUCT   IVDSKBLK,IV_SIZE
            STRUCT   IVSOFTINT,IV_SIZE
            STRUCT   IVPORTS,IV_SIZE
            STRUCT   IVCOPER,IV_SIZE
            STRUCT   IVVERTB,IV_SIZE
            STRUCT   IVBLIT,IV_SIZE
            STRUCT   IVAUD0,IV_SIZE
            STRUCT   IVAUD1,IV_SIZE
            STRUCT   IVAUD2,IV_SIZE
            STRUCT   IVAUD3,IV_SIZE
            STRUCT   IVRBF,IV_SIZE
            STRUCT   IVDSKSYNC,IV_SIZE
            STRUCT   IVEXTER,IV_SIZE
            STRUCT   IVINTEN,IV_SIZE
            STRUCT   IVNMI,IV_SIZE


******* Dynamic System Variables *************************************

            APTR  ThisTask        ; pointer to current task
            ULONG IdleCount       ; idle counter
            ULONG DispCount       ; dispatch counter
            UWORD Quantum         ; time slice quantum
            UWORD Elapsed         ; current quantum ticks
            UWORD SysFlags        ; misc system flags
            BYTE  IDNestCnt       ; interrupt disable nesting count
            BYTE  TDNestCnt       ; task disable nesting count

            UWORD AttnFlags       ; special attention flags
            UWORD AttnResched     ; rescheduling attention
            APTR  ResModules      ; pointer to resident module array

            APTR TaskTrapCode     ; default task trap routine
            APTR TaskExceptCode   ; default task exception code
            APTR TaskExitCode     ; default task exit code
            ULONG TaskSigAlloc    ; preallocated signal mask
            UWORD TaskTrapAlloc   ; preallocated trap mask


******* System List Headers ***********************************************

            STRUCT   MemList,LH_SIZE
            STRUCT   ResourceList,LH_SIZE
            STRUCT   DeviceList,LH_SIZE
            STRUCT   IntrList,LH_SIZE
            STRUCT   LibList,LH_SIZE
            STRUCT   PortList,LH_SIZE
            STRUCT   TaskReady,LH_SIZE
            STRUCT   TaskWait,LH_SIZE

            STRUCT   SoftInts,SH_SIZE*5

            STRUCT  LastAlert,4*4

            LONG ExecBaseReserved,4*8

            LABEL SYSBASESIZE

```

```
******* AttnFlags
*  Processors and Co-processors:
AFB_68010   EQU      0  (will remain set for 68020 as well)
AFB_68020   EQU      1
AFB_68881   EQU      4
AFB_PAL     EQU      8  PAL/NTSC
AFB_50HZ    EQU      9  Clock Rate
    ENDC !EXEC_EXECBASE_I
```

Jan 3 22:47 1986 exec/execname.i Page 1

```
    IFND EXEC_EXECNAME_I
EXEC_EXECNAME_I SET 1
* Commodore-Amiga, Inc.
EXECNAME macro
      dc.b  'exec.library',0
      ds.w  0
      endm

    ENDC !EXEC_EXECNAME_I

```

Jan 3 22:47 1986 exec/initializers.i Page 1

```
    IFND EXEC_INITIALIZERS_I
EXEC_INITIALIZERS_I SET 1
**********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
**********************************************************************
*
*   Source Control:
*
*  $Header: initializers.i,v 1.0 85/08/28 15:09:29 carl Exp $
*
*  $Locker:  $
*
**********************************************************************


INITBYTE        MACRO   * &offset,&value
                DC.B    $e0
                DC.B    0
                DC.W    \1
                DC.B    \2
                DC.B    0
                ENDM

INITWORD        MACRO   * &offset,&value
                DC.B    $d0
                DC.B    0
                DC.W    \1
                DC.W    \2
                ENDM

INITLONG        MACRO   * &offset,&value
                DC.B    $c0
                DC.B    0
                DC.W    \1
                DC.L    \2
                ENDM

INITSTRUCT  MACRO   * &size,&offset,&value,&count
                DS.W    0
            IFC     '\4',''
COUNT\@         SET     0
            ENDC
            IFNC    '\4',''
COUNT\@         SET     \4
            ENDC
CMD\@           SET     (((\1)<<4)!COUNT\@)
            IFLE    (\2)-255
                DC.B    (CMD\@)!$80
                DC.B    \2
           MEXIT
           ENDC
                DC.B    CMD\@!$0C0
                DC.B    (((\2)>>16)&$0FF)
                DC.W    ((\2)&$0FFFF)
            ENDM

    ENDC !EXEC_INITIALIZERS_I
```

```
    IFND EXEC_INTERRUPTS_I
EXEC_INTERRUPTS_I SET 1
*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*  $Header: interrupts.i,v 1.0 85/08/28 15:10:16 carl Exp $
*
*  $Locker:  $
*
*********************************************************************

    IFND EXEC_NODES_I
    INCLUDE "exec/nodes.i"
    ENDC !EXEC_NODES_I

    IFND EXEC_LISTS_I
    INCLUDE "exec/lists.i"
    ENDC !EXEC_LISTS_I


*----------------------------------------------------------------
*
*   Interrupt Structure
*
*----------------------------------------------------------------

 STRUCTURE  IS,LN_SIZE
    APTR    IS_DATA
    APTR    IS_CODE
    LABEL   IS_SIZE


*---------------------------------------------------------------------
*
*   Exec Internal Interrupt Vectors
*
*---------------------------------------------------------------------

 STRUCTURE  IV,0
    APTR    IV_DATA
    APTR    IV_CODE
    APTR    IV_NODE
    LABEL   IV_SIZE


*------ System Flag bits (in SysBase.SysFlags )

    BITDEF  S,SAR,15        * scheduling attention required
    BITDEF  S,TQE,14        * time quantum expended -- time to resched
    BITDEF  S,SINT,13

```

```

*----------------------------------------------------------------------
*
*   Software Interrupt List Headers
*
*----------------------------------------------------------------------

 STRUCTURE  SH,LH_SIZE
    UWORD   SH_PAD
    LABEL   SH_SIZE

SIH_PRIMASK  EQU  $0F0
SIH_QUEUES   EQU  5

    ENDC !EXEC_INTERRUPTS_I
```

```
    IFND EXEC_IO_I
EXEC_IO_I SET 1
***********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
***********************************************************************
*
*   Source Control:
*
*  $Header: io.i,v 1.0 85/08/28 15:10:43 carl Exp $
*
*  $Locker:  $
*
***********************************************************************

    IFND EXEC_PORTS_I
    INCLUDE "exec/ports.i"
    ENDC !EXEC_PORTS_I

    IFND EXEC_LIBRARIES_I
    INCLUDE "exec/libraries.i"
    ENDC !EXEC_LIBRARIES_I


*------------------------------------------------------------------
*
*   IO Request Structures
*
*------------------------------------------------------------------

*------ Required portion of IO request:

 STRUCTURE  IO,MN_SIZE
    APTR    IO_DEVICE           * device node pointer
    APTR    IO_UNIT             * unit (driver private)
    UWORD   IO_COMMAND          * device command
    UBYTE   IO_FLAGS            * special flags
    BYTE    IO_ERROR            * error or warning code
    LABEL   IO_SIZE


*------ Standard IO request extension:

    ULONG   IO_ACTUAL           * actual # of bytes transfered
    ULONG   IO_LENGTH           * requested # of bytes transfered
    APTR    IO_DATA             * pointer to data area
    ULONG   IO_OFFSET           * offset for seeking devices
*   ULONG   IO_RESERVED1
*   ULONG   IO_RESERVED2
    LABEL   IOSTD_SIZE


*------ IO_FLAGS bit definitions:

    BITDEF  IO,QUICK,0          * complete IO quickly
```

```


*----------------------------------------------------------------
*
*   Standard Device Library Functions
*
*----------------------------------------------------------------

        LIBINIT

        LIBDEF  DEV_BEGINIO     * process IO request
        LIBDEF  DEV_ABORTIO     * abort IO request


*----------------------------------------------------------------
*
*   IO Function Macros
*
*----------------------------------------------------------------

BEGINIO     MACRO
        LINKLIB DEV_BEGINIO,IO_DEVICE(A1)
        ENDM

ABORTIO     MACRO
        LINKLIB DEV_ABORTIO,IO_DEVICE(A1)
        ENDM


*----------------------------------------------------------------
*
*   Standard Device Command Definitions
*
*----------------------------------------------------------------

*------ Command definition macro:
DEVINIT     MACRO   * [baseOffset]
        IFC         '\1',''
CMD_COUNT   SET     CMD_NONSTD
        ENDC
        IFNC      '\1',''
CMD_COUNT   SET         \1
        ENDC
            ENDM

DEVCMD      MACRO   * cmdname
\1          EQU       CMD_COUNT
CMD_COUNT   SET        CMD_COUNT+1
        ENDM


*------ Standard device commands:

        DEVINIT 0

        DEVCMD  CMD_INVALID    * invalid command
```

```
        DEVCMD  CMD_RESET       * reset as if just inited
        DEVCMD  CMD_READ        * standard read
        DEVCMD  CMD_WRITE       * standard write
        DEVCMD  CMD_UPDATE      * write out all buffers
        DEVCMD  CMD_CLEAR       * clear all buffers
        DEVCMD  CMD_STOP        * hold current and queued
        DEVCMD  CMD_START       * restart after stop
        DEVCMD  CMD_FLUSH       * abort entire queue


*------ First nonstandard device command value:

        DEVCMD  CMD_NONSTD

    ENDC !EXEC_IO_I
    IFND EXEC_LIBRARIES_I
EXEC_LIBRARIES_I SET 1
*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*  $Header: libraries.i,v 1.0 85/08/28 15:11:09 carl Exp $
*
*  $Locker:  $
*
*********************************************************************

    IFND EXEC_NODES_I
    INCLUDE "exec/nodes.i"
    ENDC !EXEC_NODES_I


*------ Special Constants ---------------------------------------

LIB_VECTSIZE    EQU   6
LIB_RESERVED    EQU   4
LIB_BASE        EQU   $FFFFFFFA   * (-LIB_VECTSIZE)
LIB_USERDEF     EQU   LIB_BASE-(LIB_RESERVED*LIB_VECTSIZE)
LIB_NONSTD      EQU   LIB_USERDEF

*----------------------------------------------------------------
*
*   Library Definition Macros
*
*----------------------------------------------------------------

*------ LIBINIT sets base offset for library function definitions:

LIBINIT     MACRO    * [baseOffset]
         IFC        '\1',''
COUNT_LIB   SET     LIB_USERDEF
            ENDC
         IFNC    '\1',''
COUNT_LIB   SET        \1
         ENDC
            ENDM


*------ LIBDEF is used to define each library function entry:

LIBDEF      MACRO    * libraryFunctionSymbol
\1          EQU      COUNT_LIB
COUNT_LIB   SET      COUNT_LIB-LIB_VECTSIZE
            ENDM


*----------------------------------------------------------------
```

```
*
*   Standard Library Functions
*
*----------------------------------------------------------------

    LIBINIT LIB_BASE

    LIBDEF  LIB_OPEN
    LIBDEF  LIB_CLOSE
    LIBDEF  LIB_EXPUNGE
    LIBDEF  LIB_EXTFUNC        * reserved *


*----------------------------------------------------------------
*
*   Standard Library Data Structure
*
*----------------------------------------------------------------

 STRUCTURE LIB,LN_SIZE
    UBYTE   LIB_FLAGS
    UBYTE   LIB_pad
    UWORD   LIB_NEGSIZE              * number of bytes before LIB
    UWORD   LIB_POSSIZE              * number of bytes after LIB
    UWORD   LIB_VERSION              * major
    UWORD   LIB_REVISION             * minor
    APTR    LIB_IDSTRING             * identification
    ULONG   LIB_SUM                  * the checksum itself
    UWORD   LIB_OPENCNT              * number of current opens
    LABEL   LIB_SIZE


*------ LIB_FLAGS bit definitions:

    BITDEF  LIB,SUMMING,0            * we are currently checksumming
    BITDEF  LIB,CHANGED,1            * we have just changed the lib
    BITDEF  LIB,SUMUSED,2            * set if we should bother to sum
    BITDEF  LIB,DELEXP,3             * delayed expunge


*----------------------------------------------------------------
*
*   Function Invocation Macros
*
*----------------------------------------------------------------

*------ CALLLIB for calling functions where A6 is already correct:

CALLLIB    MACRO   * functionOffset
  IFGT NARG-1
     FAIL    !!! CALLLIB MACRO - too many arguments !!!
  ENDC
           JSR     \1(A6)
           ENDM

```

```
*------ LINKLIB for calling functions where A6 is incorrect:

LINKLIB     MACRO   * functionOffset,libraryBase
   IFGT NARG-2
      FAIL    !!! LINKLIB MACRO - too many arguments !!!
   ENDC
            MOVE.L  A6,-(SP)
            MOVE.L  \2,A6
            CALLLIB \1
            MOVE.L  (SP)+,A6
            ENDM

    ENDC !EXEC_LIBRARIES_I
```

```
    IFND EXEC_LISTS_I
EXEC_LISTS_I SET 1
*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*  $Header: lists.i,v 1.1 85/09/06 15:49:56 carl Exp $
*
*  $Locker:  $
*
*********************************************************************

    IFND EXEC_NODES_I
    INCLUDE "exec/nodes.i"
    ENDC !EXEC_NODES_I


*-------------------------------------------------------------------
*
*   List Structures
*
*-------------------------------------------------------------------

 STRUCTURE  LH,0
    APTR    LH_HEAD
    APTR    LH_TAIL
    APTR    LH_TAILPRED
    UBYTE   LH_TYPE
    UBYTE   LH_pad
    LABEL   LH_SIZE


NEWLIST       MACRO   * list
          MOVE.L  \1,(\1)
          ADDQ.L  #LH_TAIL,(\1)
          CLR.L   LH_TAIL(\1)
          MOVE.L  \1,(LH_TAIL+LN_PRED)(\1)
          ENDM

TSTLIST       MACRO   * [list]
       IFC      '\1',''
          CMP.L   LH_TAIL+LN_PRED(A0),A0
       ENDC
       IFNC     '\1',''
          CMP.L   LH_TAIL+LN_PRED(\1),\1
       ENDC
       ENDM

SUCC          MACRO   * node,succ
          MOVE.L  (\1),\2
       ENDM

```

```
PRED          MACRO   * node,pred
          MOVE.L  LN_PRED(\1),\2
       ENDM

IFEMPTY       MACRO   * list,label
          CMP.L   LH_TAIL+LN_PRED(\1),\1
          BEQ     \2
       ENDM

IFNOTEMPTY  MACRO    * list,label
          CMP.L   LH_TAIL+LN_PRED(\1),\1
          BNE     \2
       ENDM

TSTNODE     MACRO    * node,next
          MOVE.L  (\1),\2
          TST.L   (\2)
       ENDM

NEXTNODE    MACRO    * next,current,exit_label (DX,AX,DISP16)
          MOVE.L  \1,\2
          MOVE.L  (\2),\1
          BEQ     \3
       ENDM

ADDHEAD       MACRO
          MOVE.L  (A0),D0
          MOVE.L  A1,(A0)
          MOVEM.L D0/A0,(A1)
          MOVE.L  D0,A0
          MOVE.L  A1,LN_PRED(A0)
       ENDM

ADDTAIL       MACRO
          LEA      LH_TAIL(A0),A0
          MOVE.L  LN_PRED(A0),D0
          MOVE.L  A1,LN_PRED(A0)
          MOVE.L  A0,(A1)
          MOVE.L  D0,LN_PRED(A1)
          MOVE.L  D0,A0
          MOVE.L  A1,(A0)
       ENDM

REMOVE        MACRO
          MOVE.L  (A1),A0
          MOVE.L  LN_PRED(A1),A1
          MOVE.L  A0,(A1)
          MOVE.L  A1,LN_PRED(A0)
       ENDM

REMHEAD       MACRO
          MOVE.L  (A0),A1
          MOVE.L  (A1),D0
          BEQ.S   REMHEAD\@
          MOVE.L  D0,(A0)
          EXG.L   D0,A1
```

```
          MOVE.L  A0,LN_PRED(A1)
REMHEAD\@
       ENDM

*-----------------------------------------------------------------
*
*  REMHEADQ -- remove-head quickly
*
*  Useful when a scratch register is available and
*  list is known to contain at least one node.
*
*-----------------------------------------------------------------

REMHEADQ     MACRO    * head,node,scratchReg
          MOVE.L  (\1),\2
          MOVE.L  (\2),\3
          MOVE.L  \3,(\1)
          MOVE.L  \1,LN_PRED(\3)
       ENDM

REMTAIL        MACRO
          MOVE.L  LH_TAIL+LN_PRED(A0),A1
          MOVE.L  LN_PRED(A1),D0
          BEQ.S   REMTAIL\@
          MOVE.L  D0,LH_TAIL+LN_PRED(A0)
          EXG.L   D0,A1
          MOVE.L  A0,(A1)
          ADDQ.L  #4,(A1)
REMTAIL\@
       ENDM

    ENDC !EXEC_LISTS_I
    IFND EXEC_MEMORY_I
EXEC_MEMORY_I SET 1
***********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
***********************************************************************
*
*   Source Control:
*
*   $Header: memory.i,v 1.0 85/08/28 15:12:02 carl Exp $
*
*   $Locker:  $
*
***********************************************************************

    IFND EXEC_NODES_I
    INCLUDE "exec/nodes.i"
    ENDC !EXEC_NODES_I


*---------------------------------------------------------------------
*
*   Memory List Structures
*
*---------------------------------------------------------------------
*
*   A memory list appears in two forms:  One is a requirements list;
*   the other is a list of already allocated memory.  The format is
*   the same, with the requirements/address field occupying the same
*   position.
*
*   The format is a linked list of ML structures, each of which has
*   an array of ME entries.
*
*---------------------------------------------------------------------

 STRUCTURE ML,LN_SIZE
    UWORD   ML_NUMENTRIES           * The number of ME structures that follow
    LABEL   ML_ME                   * where the ME structures begin
    LABEL   ML_SIZE


 STRUCTURE ME,0
    LABEL   ME_REQS                 * the AllocMem requirements
    APTR    ME_ADDR                 * the address of this block (an alias
*                                   *   for the same location as ME_REQS)
    ULONG   ME_LENGTH               * the length of this region
    LABEL   ME_SIZE


*------ memory options:

    BITDEF  MEM,PUBLIC,0
    BITDEF  MEM,CHIP,1
    BITDEF  MEM,FAST,2
```

```
    BITDEF  MEM,CLEAR,16
    BITDEF  MEM,LARGEST,17


*------ alignment rules for a memory block:

MEM_BLOCKSIZE   EQU 8
MEM_BLOCKMASK   EQU (MEM_BLOCKSIZE-1)


*---------------------------------------------------------------------
*
*   Memory Region Header
*
*---------------------------------------------------------------------

 STRUCTURE  MH,LN_SIZE
    UWORD   MH_ATTRIBUTES           * characteristics of this region
    APTR    MH_FIRST                * first free region
    APTR    MH_LOWER                * lower memory bound
    APTR    MH_UPPER                * upper memory bound+1
    ULONG   MH_FREE                 * number of free bytes
    LABEL   MH_SIZE


*---------------------------------------------------------------------
*
*   Memory Chunk
*
*---------------------------------------------------------------------

 STRUCTURE  MC,0
    APTR    MC_NEXT                 * ptr to next chunk
    ULONG   MC_BYTES                * chunk byte size
    APTR    MC_SIZE

    ENDC !EXEC_MEMORY_I
    IFND EXEC_NODES_I
EXEC_NODES_I SET 1
***********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
***********************************************************************
*
*   Source Control:
*
*  $Header: nodes.i,v 1.1 85/11/12 18:23:08 carl Exp $
*
*  $Locker:  $
*
***********************************************************************


*----------------------------------------------------------------
*
*   List Node Structure
*
*----------------------------------------------------------------

 STRUCTURE  LN,0
    APTR    LN_SUCC
    APTR    LN_PRED
    UBYTE   LN_TYPE
    BYTE    LN_PRI
    APTR    LN_NAME
    LABEL   LN_SIZE


*------ Node Types:

NT_UNKNOWN      EQU     0
NT_TASK         EQU     1
NT_INTERRUPT    EQU     2
NT_DEVICE       EQU     3
NT_MSGPORT      EQU     4
NT_MESSAGE      EQU     5
NT_FREEMSG      EQU     6
NT_REPLYMSG     EQU     7
NT_RESOURCE     EQU     8
NT_LIBRARY      EQU     9
NT_MEMORY       EQU     10
NT_SOFTINT      EQU     11
NT_FONT         EQU     12
NT_PROCESS      EQU     13
NT_SEMAPHORE    EQU     14

    ENDC !EXEC_NODES_I
```

```
    IFND EXEC_PORTS_I
EXEC_PORTS_I SET 1
***********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
***********************************************************************
*
*   Source Control:
*
*  $Header: ports.i,v 1.1 85/11/12 18:12:24 carl Exp $
*
*  $Locker:  $
*
***********************************************************************

    IFND EXEC_NODES_I
    INCLUDE "exec/nodes.i"
    ENDC !EXEC_NODES_I

    IFND EXEC_LISTS_I
    INCLUDE "exec/lists.i"
    ENDC !EXEC_LISTS_I


*----------------------------------------------------------------
*
*   Message Port Structure
*
*----------------------------------------------------------------

 STRUCTURE  MP,LN_SIZE
    UBYTE   MP_FLAGS
    UBYTE   MP_SIGBIT               * signal bit number
    APTR    MP_SIGTASK              * task to be signalled
    STRUCT  MP_MSGLIST,LH_SIZE      * message linked list
    LABEL   MP_SIZE


*------ unions:

MP_SOFTINT      EQU  MP_SIGTASK


*------ flags fields:

PF_ACTION       EQU  3


*------ PutMsg actions:

PA_SIGNAL       EQU  0
PA_SOFTINT      EQU  1
PA_IGNORE       EQU  2


```

```
*-----------------------------------------------------------------------
*
*   Message Structure
*
*-----------------------------------------------------------------------

 STRUCTURE  MN,LN_SIZE
    APTR    MN_REPLYPORT            * message reply port
    UWORD   MN_LENGTH               * message len in bytes
    LABEL   MN_SIZE


*-----------------------------------------------------------------------
*
*   Semaphore Message Port
*
*-----------------------------------------------------------------------

 STRUCTURE  SM,MP_SIZE
    WORD    SM_BIDS     * number of bids for lock
    LABEL   SM_SIZE


*------ unions:

SM_LOCKMSG    EQU  MP_SIGTASK

    ENDC !EXEC_PORTS_I
```

```
    IFND EXEC_RESIDENT_I
EXEC_RESIDENT_I SET 1
***********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
***********************************************************************
*
*   Source Control:
*
*  $Header: resident.i,v 1.0 85/08/28 15:13:41 carl Exp $
*
*  $Locker:  $
*
***********************************************************************


*-----------------------------------------------------------------------
*
*   Resident Module Tag
*
*-----------------------------------------------------------------------

 STRUCTURE RT,0
    UWORD RT_MATCHWORD          * word to match
    APTR  RT_MATCHTAG           * pointer to structure base
    APTR  RT_ENDSKIP            * address to continue scan
    UBYTE RT_FLAGS              * various tag flags
    UBYTE RT_VERSION            * release version number
    UBYTE RT_TYPE               * type of module
    BYTE  RT_PRI                * initialization priority
    APTR  RT_NAME               * pointer to node name
    APTR  RT_IDSTRING           * pointer to id string
    APTR  RT_INIT               * pointer to init code
    LABEL RT_SIZE


*------ Match word definition:

RTC_MATCHWORD  EQU   $4AFC     * (ILLEGAL instruction)


*------ RT_FLAGS bit and field definitions:

    BITDEF RT,COLDSTART,0
    BITDEF RT,AUTOINIT,7        * RT_INIT points to data

* Compatibility:
RTM_WHEN       EQU   1          * field position in RT_FLAGS
RTW_NEVER      EQU   0          * never ever init
RTW_COLDSTART  EQU   1          * init at coldstart time

    ENDC !EXEC_RESIDENT_I
    IFND EXEC_STRINGS_I
EXEC_STRINGS_I SET 1
*********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*********************************************************************
*
*   Source Control:
*
*  $Header: strings.i,v 1.0 85/08/28 15:14:06 carl Exp $
*
*  $Locker:  $
*
*********************************************************************


*------  Terminal Control:

EOS             EQU     0
BELL            EQU     7
LF              EQU     10
CR              EQU     13
BS              EQU     8
DEL             EQU     $7F
NL              EQU     LF


*----------------------------------------------------------------
*
*   String Support Macros
*
*----------------------------------------------------------------

STRING      MACRO
            DC.B    \1
            DC.B    0
            CNOP    0,2
            ENDM


STRINGL     MACRO
            DC.B    13,10
            DC.B    \1
            DC.B    0
            CNOP    0,2
            ENDM


STRINGR     MACRO
            DC.B    \1
            DC.B    13,10,0
            CNOP    0,2
            ENDM


```

```
STRINGLR    MACRO
            DC.B    13,10
            DC.B    \1
            DC.B    13,10,0
            CNOP    0,2
            ENDM

        ENDC !EXEC_STRINGS_I
```

```
    IFND EXEC_TASKS_I
EXEC_TASKS_I SET 1
**********************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
**********************************************************************
*
*   Source Control:
*
*  $Header: tasks.i,v 1.0 85/08/28 15:14:32 carl Exp $
*
*  $Locker:  $
*
**********************************************************************

    IFND EXEC_NODES_I
    INCLUDE "exec/nodes.i"
    ENDC !EXEC_NODES_I

    IFND EXEC_LISTS_I
    INCLUDE "exec/lists.i"
    ENDC !EXEC_LISTS_I


*----------------------------------------------------------------
*
*   Task Control Structure
*
*----------------------------------------------------------------

 STRUCTURE  TC,LN_SIZE
    UBYTE   TC_FLAGS
    UBYTE   TC_STATE
    BYTE    TC_IDNESTCNT            * intr disabled nesting
    BYTE    TC_TDNESTCNT            * task disabled nesting
    ULONG   TC_SIGALLOC             * sigs allocated
    ULONG   TC_SIGWAIT              * sigs we are waiting for
    ULONG   TC_SIGRECVD             * sigs we have received
    ULONG   TC_SIGEXCEPT            * sigs we take as exceptions
    UWORD   TC_TRAPALLOC            * traps allocated
    UWORD   TC_TRAPABLE             * traps enabled
    APTR    TC_EXCEPTDATA           * data for except proc
    APTR    TC_EXCEPTCODE           * exception procedure
    APTR    TC_TRAPDATA             * data for proc trap proc
    APTR    TC_TRAPCODE             * proc trap procedure
    APTR    TC_SPREG                * stack pointer
    APTR    TC_SPLOWER              * stack lower bound
    APTR    TC_SPUPPER              * stack upper bound + 2
    APTR    TC_SWITCH               * task losing CPU
    APTR    TC_LAUNCH               * task getting CPU
    STRUCT  TC_MEMENTRY,LH_SIZE     * allocated memory
    APTR    TC_Userdata
    LABEL   TC_SIZE


```

```
*------ Flag Bits:

    BITDEF  T,PROCTIME,0
    BITDEF  T,STACKCHK,4
    BITDEF  T,EXCEPT,5
    BITDEF  T,SWITCH,6
    BITDEF  T,LAUNCH,7


*------ Task States:
TS_INVALID  EQU     0
TS_ADDED    EQU     TS_INVALID+1
TS_RUN      EQU     TS_ADDED+1
TS_READY    EQU     TS_RUN+1
TS_WAIT     EQU     TS_READY+1
TS_EXCEPT   EQU     TS_WAIT+1
TS_REMOVED  EQU     TS_EXCEPT+1


*------ System Task Signals:

SIGF_ABORT  EQU   $0001
SIGF_CHILD  EQU   $0002
SIGF_BLIT   EQU   $0010
SIGF_DOS    EQU   $0100

SIGB_ABORT  EQU   0
SIGB_CHILD  EQU   1
SIGB_BLIT   EQU   4
SIGB_DOS    EQU   8


SYS_SIGALLOC   EQU   $0FFFF      ; preallocated signals
SYS_TRAPALLOC  EQU   $08000      ; preallocated traps

    ENDC !EXEC_TASKS_I
    IFND EXEC_TYPES_I
EXEC_TYPES_I SET 1
*************************************************************************
*
* Commodore-Amiga, Inc. -- ROM Operating System Executive Include File
*
*************************************************************************
*
*   Source Control:
*
*  $Header: types.i,v 1.2 85/11/15 17:44:08 carl Exp $
*
*  $Locker:  $
*
*************************************************************************

EXTERN_LIB  MACRO
            XREF    _LVO\1
            ENDM

STRUCTURE   MACRO
\1          SET     0          * for assembler's sake
SOFFSET     SET     \2
            ENDM

BOOL        MACRO
\1          EQU      SOFFSET
SOFFSET     SET      SOFFSET+2
            ENDM

BYTE        MACRO
\1          EQU     SOFFSET
SOFFSET     SET     SOFFSET+1
            ENDM

UBYTE       MACRO
\1          EQU     SOFFSET
SOFFSET     SET     SOFFSET+1
            ENDM

WORD        MACRO
\1          EQU     SOFFSET
SOFFSET     SET     SOFFSET+2
            ENDM

UWORD       MACRO
\1          EQU     SOFFSET
SOFFSET     SET     SOFFSET+2
            ENDM

SHORT        MACRO
\1          EQU     SOFFSET
SOFFSET     SET     SOFFSET+2
            ENDM

USHORT      MACRO
```

```
\1          EQU     SOFFSET
SOFFSET     SET     SOFFSET+2
            ENDM

LONG        MACRO
\1          EQU     SOFFSET
SOFFSET     SET     SOFFSET+4
            ENDM

ULONG       MACRO
\1          EQU     SOFFSET
SOFFSET     SET     SOFFSET+4
            ENDM

FLOAT       MACRO
\1          EQU     SOFFSET
SOFFSET     SET     SOFFSET+4
            ENDM

APTR        MACRO
\1          EQU     SOFFSET
SOFFSET     SET     SOFFSET+4
            ENDM

RPTR        MACRO
\1          EQU     SOFFSET
SOFFSET     SET     SOFFSET+2
            ENDM

STRUCT      MACRO
\1          EQU     SOFFSET
SOFFSET     SET     SOFFSET+\2
            ENDM

LABEL       MACRO
\1          EQU     SOFFSET
            ENDM

*------ bit definition macro ------------------------------------
*
*  Given:
*
*      BITDEF  MEM,CLEAR,16
*
* Yields:
*
*      MEMB_CLEAR  EQU 16
*      MEMF_CLEAR  EQU (1.SL.MEMB_CLEAR)
*

BITDEF      MACRO    * prefix,&name,&bitnum
            BITDEF0 \1,\2,B_,\3
\@BITDEF    SET     1<<\3
            BITDEF0 \1,\2,F_,\@BITDEF
            ENDM
```

```
BITDEF0       MACRO    * prefix,&name,&type,&value
\1\3\2        EQU      \4
              ENDM

LIBRARY_VERSION   EQU   31

      ENDC EXEC_TYPES_I
```

Appendix B

OTHER ROUTINES

The *Amiga ROM Kernel* manuals have become more than just documents that describe the ROM Kernel. The name "ROM kernel" has been kept for historical reasons even though the "ROM" software loads into kickstart RAM.

Certain topics were not available as of the most recent revision of the AmigaDOS manuals. These topics are included in this appendix to ensure that Amiga developers will have the latest available information on these topics. These DOS-related topics are the Exec support library, AmigaDOS general information, and the Interchange File Format.

Exec Support Library

The Exec support library *exec_support.lib* contains Exec support functions. Included are the assembly language routines **_NewList** and **_BeginIO** and C language routines to create and delete standard and extended I/O requests.

Debug.lib Routines

This section contains summaries for the following routines in the **debug.lib** library:

KDoFmt
KGetChar
KMayGetChar
KPutChar
KPutFmt
KPutStr

Amiga.lib Routines

This section contains a description of utility functions built into **amiga.lib**. These functions are provided specifically for C-language binding in which each parameter to a function is pushed onto the stack before calling the routine and the stack appears as follows:

```
parameter.n
    ...
parameter.2      8(SP)
parameter.1      4(SP)
return.address   (SP)
```

The stack is formed in this way typically by a C-language function call such as:

result = Function(parameter.1, parameter.2 .... parameter.n);

Amiga.lib contains such functions as **printf**, **getc**, **putchar** and so on. It also contains assembly code fragments that convert C-language calling conventions—that is, stack-based variable passing converted to register-based variable passing used by the Kickstart routines.

If you wish to use the **amiga.lib** versions of the utility routines contained herein instead of those with equivalent names in the runtime library for a different C compiler, simply specify **amiga.lib** to the left on the command line for ALINK of that other runtime library and use **Astartup.obj**. The following is an example command:

```
        ALINK with myprogram.with
```

The *#myprogram.with* file contains:

```
        FROM lib:Astartup.obj, myprogram.o
        TO myprogram
        LIBRARY lib:amiga.lib lib:lc.lib
```

The advantage for certain routines (notably **printf**) is that you may have a smaller object module as a result of using the equivalent **amiga.lib** functions. The disadvantage is that there are some limitations on the formatting capabilities (outlined here). Also, be aware that you cannot mix functions from the runtime libraries of different compilers when I/O is used. That is, a "file handle" obtained from an AmigaDOS **Open()** routine cannot be passed as a "stream" to the Amiga C function **fprintf**. Thus, you should be aware of the parameters that each function expects.

AmigaDOS General Information

This part describes certain topics that are likely to be of interest to advanced developers who may wish to create new devices to be added to the Amiga or who may wish their code to run on Amiga computers that have been expanded beyond a 512K memory size.

Interchange File Format

The material in this appendix was developed jointly by Commodore-Amiga and Electronic Arts. It includes source code for routines that will both read and write this data format. Commodore-Amiga has adopted this data interchange file format for internal use, and we recommend that our developers adopt it as well.

Apr 1 15:54 1986 appxD.part1/NewList.asm Page 1

```
*************************************************************************
*
*       Exec Support Functions -- NewList
*
*************************************************************************

        INCLUDE "exec/types.i"
        INCLUDE "exec/nodes.i"
        INCLUDE "exec/lists.i"

        section _NewList
        xdef    _NewList

_NewList:
                move.l  4(sp),a0
                NEWLIST a0
                rts

        end
```

Apr 1 15:54 1986 appxD.part1/BeginIO.asm Page 1

```
        INCLUDE "exec/types.i"
        INCLUDE "exec/nodes.i"
        INCLUDE "exec/lists.i"
        INCLUDE "exec/libraries.i"
        INCLUDE "exec/ports.i"
        INCLUDE "exec/io.i"


        section _BeginIO
        xdef    _BeginIO

_BeginIO:
                move.l  4(sp),a1
                BEGINIO
                rts

        end
```

```
/**************************************************************************
*
*       Exec Support Functions -- Ports and Messages
*
**************************************************************************/

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/ports.h"
#include "exec/libraries.h"
#include "exec/tasks.h"
#include "exec/execbase.h"


extern APTR AllocMem();
extern UBYTE AllocSignal();
extern struct Task *FindTask();


struct MsgPort *CreatePort (name, pri)
    char *name;
    BYTE  pri;
{
    BYTE sigBit;
    struct MsgPort *port;

    if ((sigBit = AllocSignal (-1)) == -1)
        return ((struct MsgPort *) 0);

    port = AllocMem ((ULONG) sizeof (*port), MEMF_CLEAR | MEMF_PUBLIC);

    if (port == 0) {
        FreeSignal (sigBit);
        return ((struct MsgPort *) (0));
    }

    port -> mp_Node.ln_Name = name;
    port -> mp_Node.ln_Pri = pri;
    port -> mp_Node.ln_Type = NT_MSGPORT;

    port -> mp_Flags = PA_SIGNAL;
    port -> mp_SigBit = sigBit;
    port -> mp_SigTask = FindTask (0);

    if (name != 0)
        AddPort (port);
    else
        NewList (&(port -> mp_MsgList));

    return (port);
}


DeletePort(port)
    struct MsgPort *port;
{
    if ((port -> mp_Node.ln_Name) != 0)
        RemPort (port);

    port -> mp_Node.ln_Type = 0xff;
    port -> mp_MsgList.lh_Head = (struct Node  *) - 1;

    FreeSignal (port -> mp_SigBit);

    FreeMem (port, (ULONG) sizeof (*port));
}
```

```
/***********************************************************************
*
*       Exec Support Functions -- Standard I/O Requests
*
***********************************************************************/

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/ports.h"
#include "exec/libraries.h"
#include "exec/io.h"
#include "exec/tasks.h"
#include "exec/execbase.h"


/****** exec_support/CreateStdIO ************************************
*
*   NAME
*       CreateStdIO() -- create a standard I/O request
*
*   SYNOPSIS
*       ioStdReq = CreateStdIO( ioReplyPort );
*
*   FUNCTION
*       Allocates memory for and initializes a new I/O request block.
*
*   INPUTS
*       ioReplyPort - a pointer to an already initialized
*               message port to be used for this I/O request's
*               reply port.
*
*   RESULT
*       Returns a pointer to the new io request block.  A NULL
*       indicates that there was not enough memory for the I/O Request,
*       or that the reply port was not a valid port.
*
*   EXAMPLE
*       struct IOStdReq *myBlock;
*       struct MsgPort *port;
*
*       myBlock = CreateStdIO( port );
*       if( myBlock == NULL) {
*               printf( "Insufficient memory" );
*       }
*
*   SEE ALSO
*       DeleteStdIO, CreateExtIO
*
***********************************************************************/

struct IOStdReq *
CreateStdIO( ioReplyPort )
```

```
    struct MsgPort *ioReplyPort;
{
    return( (struct IOStdReq *)
        CreateExtIO( ioReplyPort, sizeof( struct IOStdReq ) ) );
}

/****** exec_support/DeleteStdIO ****************************************
*
*   NAME
*       DeleteStdIO( ioStdReq ) - return memory allocated for I/O request
*
*   SYNOPSIS
*       DeleteStdIO(ioStdReq);
*
*   FUNCTION
*       Free memory allocate for the io request.
*
*   INPUTS
*       A pointer to the IOStdReq block whose resources are to be freed.
*
*   RESULT
*       no defined return value
*
*   EXAMPLE
*       struct IOStdReq *ioRequest;
*       DeleteStdIO( ioRequest );
*
*   SEE ALSO
*       CreateStdIO, DeleteExtIO
*
***********************************************************************/

DeleteStdIO( ioStdReq )
    struct IOStdReq *ioStdReq;
{
    return( DeleteExtIO( ioStdReq ) );
}
```

```
#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/ports.h"
#include "exec/libraries.h"
#include "exec/tasks.h"
#include "exec/execbase.h"

extern APTR AllocMem();
extern struct Task *FindTask();

/*
 *  Create a task with given name, priority, and stack size.
 *  It will use the default exception and trap handlers for now.
 */

/* the template for the mementries.  Unfortunately, this is hard to
 * do from C: mementries have unions, and they cannot be statically
 * initialized...
 *
 * In the interest of simplicity I recreate the mem entry structures
 * here with appropriate sizes.  We will copy this to a local
 * variable and set the stack size to what the user specified,
 * then attempt to actually allocate the memory.
 */

#define ME_TASK         0
#define ME_STACK        1
#define NUMENTRIES      2

struct FakeMemEntry {
    ULONG fme_Reqs;
    ULONG fme_Length;
};

struct FakeMemList {
    struct Node fml_Node;
    UWORD       fml_NumEntries;
    struct FakeMemEntry fml_ME[NUMENTRIES];
} TaskMemTemplate = {
    { 0 },                                          /* Node */
    NUMENTRIES,                                     /* num entries */
    {                                               /* actual entries: */
        { MEMF_PUBLIC | MEMF_CLEAR, sizeof( struct Task ) },    /* task */
        { MEMF_CLEAR,   0 }                                     /* stack */
    }
};


struct Task *
```

```
CreateTask( name, pri, initPC, stackSize )
    char *name;
    UBYTE pri;
    APTR  initPC;
    ULONG stackSize;
{
    struct Task *newTask;
    ULONG memSize;
    struct FakeMemList fakememlist;
    struct MemList *ml;

    /* round the stack up to longwords... */
    stackSize = (stackSize +3) & ~3;

    /*
     * This will allocate two chunks of memory: task of PUBLIC
     * and stack of PRIVATE
     */
    fakememlist = TaskMemTemplate;
    fakememlist.fml_ME[ME_STACK].fme_Length = stackSize;

    ml = (struct MemList *) AllocEntry( &fakememlist );

    if( ! ml ) {
        return( NULL );
    }

    /* set the stack accounting stuff */
    newTask = (struct Task *) ml->ml_ME[ME_TASK].me_Addr;

    newTask->tc_SPLower = ml->ml_ME[ME_STACK].me_Addr;
    newTask->tc_SPUpper = (APTR)((ULONG)(newTask->tc_SPLower) + stackSize);
    newTask->tc_SPReg = newTask->tc_SPUpper;

    /* misc task data structures */
    newTask->tc_Node.ln_Type = NT_TASK;
    newTask->tc_Node.ln_Pri = pri;
    newTask->tc_Node.ln_Name = name;

    /* add it to the tasks memory list */
    NewList( &newTask->tc_MemEntry );
    AddHead( &newTask->tc_MemEntry, ml );

    /* add the task to the system -- use the default final PC */
    AddTask( newTask, initPC, 0 );
    return( newTask );
}


DeleteTask( tc )
    struct Task *tc;
{
    /* because we added a MemList structure to the tasks's TC_MEMENTRY
     * structure, all the memory will be freed up for us!
     */
    RemTask( tc );
}
```

```
/*********************************************************************
*
*       Exec Support Function -- Extended I/O Request
*
*********************************************************************/

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/ports.h"
#include "exec/libraries.h"
#include "exec/io.h"
#include "exec/tasks.h"
#include "exec/execbase.h"

extern APTR AllocMem();

/****** exec_support/CreateExtIO *************************************
*
*   NAME
*       CreateExtIO() -- create an I/O request
*
*   SYNOPSIS
*       ioReq = CreateExtIO( ioReplyPort, size );
*
*   FUNCTION
*       Allocates memory for and initializes a new I/O request block
*       of a user-specified number of bytes.  The number of bytes
*       MUST be greater than the length of an Exec message, or some
*       very nasty things will happen.
*
*   INPUTS
*       ioReplyPort - a pointer to an already initialized
*               message port to be used for this I/O request's reply port.
*       size - the size of the I/O request to be created.
*
*   RESULT
*       Returns a pointer to the new I/O Request block, or NULL if
*               the request failed.
*
*   EXAMPLE
*       struct IORequest *myBlock;
*       struct MsgPort *port;
*
*       myBlock = CreateExtIO( port, sizeof(struct IOExtTD) );
*       if( myBlock == NULL ) {
*               exit( NO_MEM_OR_SIGNALS );
*       }
*
*       example used to allocate space for IOExtTD (e.g. a trackdisk driver
*       I/O Request block for extended I/O operations).
*
*   SEE ALSO
*       DeleteExtIO
```

```
 *
 ***************************************************************************/

struct IORequest *
CreateExtIO( ioReplyPort, size )
    struct MsgPort *ioReplyPort;
    LONG size;
{
    struct IORequest *ioReq;

    if( ioReplyPort == NULL ) {
        return ( NULL );
    }

    ioReq = AllocMem( size, MEMF_CLEAR | MEMF_PUBLIC );

    if( ioReq == NULL ) {
        return( NULL );
    }

    ioReq->io_Message.mn_Node.ln_Type = NT_MESSAGE;
    ioReq->io_Message.mn_Length = size;

    ioReq->io_Message.mn_ReplyPort = ioReplyPort;

    return( ioReq );
}

/****** exec_support/DeleteExtIO ****************************************
 *
 *   NAME
 *       DeleteExtIO() - return memory allocated for extended I/O request
 *
 *   SYNOPSIS
 *       DeleteExtIO( ioReq );
 *
 *   FUNCTION
 *       Frees up an IO request as allocated by CreateExtIO().
 *
 *   INPUTS
 *       ioReq - A pointer to the IORequest block to be freed.
 *
 *   RESULTS
 *       no return value
 *
 *   EXAMPLE
 *       struct IORequest *ioReq;
 *       DeleteExtIO( ioReq );
 *
 *   SEE ALSO
 *       CreateExtIO
 *
 ***************************************************************************/

DeleteExtIO( ioExt )
    struct IORequest *ioExt;
```

```
{
    /* just in case the user did not check things properly... */
    if( ! ioExt ) return;

    /* try to make it hard to reuse the request by accident */
    ioExt->io_Message.mn_Node.ln_Type = 0xff;
    ioExt->io_Device = (struct Device *) -1;
    ioExt->io_Unit = (struct Unit *) -1;

    FreeMem( ioExt, ioExt->io_Message.mn_Length );
}
```

INTRODUCTION

This section outlines the routines available in the debug.lib.

The debug.lib is a linked library rather than a shared library. You link to it at compile/link time rather than opening a library. This linked library becomes a physical part of your program code.

Routines that are described in the Amiga ROM Kernel Reference Manual as well as those listed here, can be called directly from tasks. Routines that are listed in the Lattice C manual, such as printf and scanf for example, should be called only from a process, rather than from a task, in that they require a process model in order to function. A process is started when you ask AmigaDOS to run your program. Any part of main() or any routines it calls become part of that process. If your program spawns any tasks on its own, to have those tasks execute any process-dependent code, the tasks should either send messages back to the main program, which in turn executes the appropriate code, or use these debug routines if the programmer has connected a 9600 baud device to the Amiga's serial port.

To start the debug mode correctly, you can set the serial port parameters by calling

```
        romwack
```

from a CLI. Then, from the external terminal, type

```
        go
```

The port is now set up and ready to go. The debug print and get-character routines will function as described below.

```
debug.lib/KDoFmt                                                debug.lib/KDoFmt

   NAME
        KDoFmt -- format data into a character stream.

   SYNOPSIS
        KDoFmt(FormatString, DataStream, PutChProc, PutChData);
               A0            A1          A2         A3

   FUNCTION
        perform "C"-language-like formatting of a data stream,
        outputting the result a character at a time

   INPUTS
        FormatString - a "C"-language-like null terminated format
                string, with the following supported % types:
        DataStream - a stream of data that is interpreted according to
                the format string.
        PutChProc - the procedure to call with each character to be
                output, called as:
            PutChProc(Char,  PutChData);
                      D0-0:8 A3
                the procedure is called with a null Char at the end of
                the format string.
        PutChData - an address register that passes thru to PutChProc.
```

```
debug.lib/KGetChar                                              debug.lib/KGetChar

   NAME
        KGetChar - get a character from the debug console (a 9600 baud
                   device attached to the serial port of the Amiga.)

   SYNOPSIS
        char = KGetChar();
        D0


   FUNCTION
        get the next character from the debug console device.
```

```
debug.lib/KMayGetChar                                   debug.lib/KMayGetChar

   NAME
        KMayGetChar - return a char iff present, but don't block.
                      (device attached serial port).

   SYNOPSIS
        flagChar = KMayGetChar();
        D0

   FUNCTION
        return either a -1, saying that there is no char present, or
        the char that was waiting


debug.lib/KPutChar                                         debug.lib/KPutChar

   NAME
        KPutChar - put a character to the debug console
                   (device attached to Amiga serial port).

   SYNOPSIS
        char = KPutChar(char);
        D0              D0

   FUNCTION
        put a character to the debug console device.


debug.lib/KPutFmt                                           debug.lib/KPutFmt

   NAME
        KPutFmt - print formatted data to the debug console
                  (device attached to Amiga serial port).

   SYNOPSIS
        KPutFmt(format,values);
                A0     A1

   FUNCTION
        print formatted data to the debug console device
```

```
debug.lib/KPutStr                                           debug.lib/KPutStr

   NAME
        KPutStr - put a string to the debug console
                  (attached to the Amiga serial port).

   SYNOPSIS
        KPutStr(string)
                A0

   FUNCTION
        put a null terminated string to the debug console device.
```

TABLE OF CONTENTS

NAME

AddTOF - add a task to the TopOfFrame Interrupt server chain.

SYNOPSIS

```
AddTOF(i,p,a);
```

FUNCTION

To add a task to the vertical-blanking interval interrupt server chain.

INPUTS

```
i - pointer to structure Isrvstr.
p - pointer to the C-code routine that this server is to call each
    time TOF happens.
a - pointer to the first longword in an array of longwords that
    is to be used as the arguments passed to your routine
    pointed to by p.
```

BUGS

Once added, the task stays in the queue! This means that the parameters that are passed to AddTOF must remain memory resident until the next system reset.

SEE ALSO

RemTOF

```
FastRand                                                    FastRand

NAME

        FastRand - quickly generate a pseudo-random number.

SYNOPSIS

        number = FastRand(seed);


FUNCTION

        C-implementation only.   Seed value is taken from stack, shifted
        left one position, exclusive-or'ed with hex value $1D872B41 and
        returned (D0).

INPUTS

        seed - a 32-bit integer

RESULT

        number - a 32-bit value

SEE ALSO

        FastRand
```

```
fgetc                                                                      fgetc

NAME

        fgetc - Get a character from a specified stream.

SYNOPSIS

        c = fgetc(fh);

FUNCTION

        NOTE:  This is a "patch" function allowing a user to specify, in
        source text, a call to a standard Unix-like function, but call
        a user-customized underlying ReadChar routine.  The amiga.lib
        implementation of fgetc is "jmp _ReadChar", which means that
        to use this function, you must also provide your own custom
        version of ReadChar(fh) with which the amiga.lib function can
        link to perform the fgetc operation.  The normal implementation
        should use the AmigaDOS file handle pointer (fh) to retrieve the next
        character from the file to which the file handle is open.  Retrieves
        a value of -1 if EOF or other error is encountered.

INPUTS

        fh - an AmigaDOS FileHandle pointer (a BCPL pointer).

RESULT

        c = integer in the range of 0 to 255, or -1 if error.

EXAMPLE
        c = fgetc(fh);
        ...
        ReadChar(fh)
        {
            struct FileHandle *fh;
            char x;
            int actual;
            actual = Read(fh,&x,1);
            if(actual == 1)
                return(x);      /* ok */
            else
                return(-1);     /* error */
        }

BUGS
        If you don't provide a ReadChar(fh) function, you will get an
        unresolved reference error when you run Alink.

SEE ALSO
        getchar
```

```
fprintf                                                          fprintf

NAME

      fprintf - print formatted output to a specified output stream.

SYNOPSIS

      fprintf( fh, formatstring [,value [,values] ] );

FUNCTION

      perform printf for a specified output stream.

INPUTS

      fh - a pointer to an AmigaDOS file handle (BCPL pointer).
      formatstring - pointer to a null terminated string describing the
                     desired output formatting.
      value(s) - numeric information to be formatted into the output
                     stream.

SEE ALSO

      printf
```

```
fputc                                                              fputc

NAME

      fputc - output a character to a specified stream.

SYNOPSIS

      fputc(fh);

FUNCTION

      Output a character through AmigaDOS to the file which is
      open to this AmigaDOS file handle (fh).

INPUTS

      fh - an AmigaDOS file handle pointer (BCPL pointer).

BUGS

      none known.

SEE ALSO

      putchar
```

```
getchar                                                         getchar

NAME

        getchar - get the next character from the standard input (stdin).

SYNOPSIS

        c = getchar();

FUNCTION

        Use the underlying AmigaDOS Read function to wait for the next
        character to be typed at the keyboard (if stdin is connected
        to a virtual terminal) or retrieve the next character from a
        file (if stdin has been redirected from a file).

INPUTS

        none

RESULT

        character is an integer (32-bit) value.  If an error is noted, a
        value of -1 is returned.  Otherwise the value should range from
        0 to 255.

BUGS

        none known.

SEE ALSO
```

```
IDivS                                                           IDivS

NAME

        IDivS - Integer Divide Signed (Short)

SYNOPSIS

        result = IDivS(dividend,divisor);

FUNCTION

        Divide a signed 32-bit value by a signed 16-bit value
        and return a signed 16-bit value as a result.

INPUTS

        dividend - signed 32-bit value
        divisor - signed 16-bit value

RESULT

        result - a signed word (16-bit) return.

BUGS

        none known.

SEE ALSO

        IDivU
```

```
IDivU                                                                IDivU

NAME

        IDivU - Integer Divide Unsigned (Short)

SYNOPSIS

        result = IDivU(dividend,divisor);

FUNCTION

        Divide an unsigned 32-bit value by an unsigned 16-bit value
        and return an unsigned 16-bit value as a result.

INPUTS

        dividend - unsigned 32-bit value
        divisor - unsigned 16-bit value

RESULT

        result - an unsigned word (16-bit) return.

BUGS

        none known.

SEE ALSO

        IDivS
```

```
IMulS                                                                IMulS

NAME

        IMulS - Integer Multiply Signed (Short)

SYNOPSIS

        result = IMulS(x,y);

FUNCTION

        Multiply two signed 16-bit values and return a signed
        32-bit value as a result.

INPUTS

        x, y - signed 16-bit values

RESULT

        result - a signed int (32-bit) return.

BUGS

        none known.

SEE ALSO

        IMulU
```

NAME

        IMulU - Integer Multiply Unsigned (Short)

SYNOPSIS

        result = IMulU(x,y);

FUNCTION

        Multiply two unsigned 16-bit values and return an unsigned
        32-bit value as a result.

INPUTS

        x, y - unsigned 16-bit values

RESULT

        result - an unsigned int (32-bit) return.

BUGS

        none known.

SEE ALSO

        IMulS

NAME

    printf - print a formatted output line to the standard output.

SYNOPSIS

    printf(formatstring [,value [,values]]);

FUNCTION

    Format the output in accordance with specifications in the format
    string:

INPUTS

    formatstring - a pointer to a null-terminated string describing the
                   format of the output data, including format conversions
                   as shown below.

    value(s) - numeric variables or addresses of null-terminated strings
               to be added to the format information.

    The function printf can handle the following format conversions, in
    common with the normal call to printf:

        %lc - the next long word in the array is to be formatted
              as a character (8-bit) value

        %ld - the next long word in the array is to be formatted
              as a decimal number

        %lx - the next long word in the array is to be formatted
              as a hexadecimal number

        %ls - the next long word is the starting address of a
              null-terminated string of characters

    The "l" (small-L) character must be used as part of the format if called
    from C since the calling sequence for C can only pass longword values.

    Notice also that the ROM resident version of a print formatter does
    NOT support any form of floating point output formatting. You'll have
    to use the Amiga C form of printf and the corresponding floating point
    functions if you wish to output floating point information.

    Following the %l, you may also specify:

    o   an optional minus (-) sign that tells the formatter
            to left-justify the formatted item within the field
            width

    o   an optional field-width specifier... that is, how
            many spaces to allot for the full width of this

```
        item.  If the field width specifier begins with
        a zero (0), it means that leading spaces, ahead of
        the formatted item (usually a number) are to be
        zero-filled instead of blank-filled

    o   an optional period (.) that separates the width
        specifier from a maximum number of characters
        specifier

    o   an optional digit string (for %ls specifications
        only) that specifies the maximum number of characters
        to print from a string.

    See other books on C language programming for examples of the use
    of these formatting options (see "printf" in other books).

BUGS

        none known
```

```
putchar                                                          putchar

NAME

        putchar - Output a single character to the current stdout

SYNOPSIS

        putchar(c);

FUNCTION

        write a single character.

INPUTS

        c - integer value ranging from 0 to 255.

BUGS

        none known.
```

```
RangeRand                                                    RangeRand

NAME

        RangeRand - To obtain a random number within a specific integer range
                    of 0 to value.

SYNOPSIS

        number = RangeRand(value);

FUNCTION

        RangeRand accepts a value from 1 to 65535, and returns a value
        within that range. (16-bit integer).  Note:  C-language implementation.
        Value is passed on stack as a 32-bit integer but used as though
        it is only a 16-bit integer.

INPUTS

        value - integer in the range of 1 to 65535.

RESULT

        number - pseudo random integer in the range of 1 to <value>.

BUGS

        none known

SEE ALSO

        FastRand
```

```
RemTOF                                                          RemTOF

NAME

        RemTOF - Remove a task from the TopOfFrame interrupt server chain.

SYNOPSIS

        RemTOF(i);

FUNCTION

        To remove a task from the vertical-blanking interval interrupt server
        chain.

INPUTS

        i - pointer to structure Isrvstr.

BUGS

        As of this writing, this routine does nothing.  A TOF server, once
        installed, remains active until the next system reset.

SEE ALSO

        AddTOF
```

```
sprintf                                                              sprintf

NAME

        sprintf - format a string into a string buffer

SYNOPSIS

        sprintf( destination, formatstring [,value [, values] ] );

FUNCTION

        perform string formatting identical to printf, but direct the output
        into a specific destination in memory.

INPUTS

        destination - the address of an area in memory into which the
                        formatted output is to be placed.
        formatstring - pointer to a null terminated string describing the
                       desired output formatting.
        value(s) - numeric information to be formatted into the output
                        stream.

BUGS

        none known.

SEE ALSO

        printf
```

AmigaDOS General Information

This part of the appendix describes certain topics that are likely to be of interest to the advanced developer, who may wish to create new devices to be added to the Amiga or who may wish their code to run with Amiga computers that have been expanded beyond a 512K memory size.

The following topics are covered here:

Overlay Hunk Description

For developers putting together large programs

ATOM utility

Works on a new binary file format to allow a developer to set the appropriate load bits. Assures that program code and data that must be resident in CHIP memory (the lowest 512K of the system) for the program to function will indeed be placed there by AmigaDOS when it is loaded. Otherwise the program code may not work on an extended memory machine.

Linking in a new DISK-device to AmigaDOS

Lets a developer add a hard disk or disklike device as a name-addressable part of the filing system.

Linking in a new non-disk-device to AmigaDOS

Lets a developer add such things as additional serial ports, parallel ports, graphics tablets, ram-disks, or what-have-you to AmigaDOS (not related to the file system).

Using AmigaDOS without using Intuition

For developers who may prefer to install and use their own screen handling in place of that provided by Intuition.

Overlay Hunk Description

When overlays are used, the linker basically produces one large file containing all of the object modules as hunks of relocatable code. The hunk overlay table contains a data structure that describes the hunks and their relationships to one another.

When you are designing a program to use overlays, you must keep in mind how the overlay manager (also called the overlay supervisor) handles the interaction between the various segments of the file. What you must do, basically, is build a tree that reflects the relationships between the various code modules of the overall program and tell the linker how this tree should be constructed.

The hunk overlay table is generated as a set of eight longwords, each describing a particular overlay node that is part of the overall file. Each eight-longword entry is comprised of the following data:

HUNK OVERLAY SYMBOL TABLE-ENTRY DATA STRUCTURE:

```
long seekOffset;      /* where in the file to find this node */
long dummy1;          /* a value of 0 ... compatibility item */
long dummy2;          /* a value of 0 ... compatibility item */
long level;           /* level in the tree */
long ordinate;        /* item number at that level */
long firstHunk;       /* hunk number of the first hunk containing
                       * this node. */
long symbolHunk;      /* the hunk number in which this symbol is
                       * located */
long symbolOffsetX;   /* (offset + 4), where offset is the offset
                       * within the symbol hunk at which this
                       * symbol's entry is located. */
```

Each of these items is explained further in the sections that follow.

DESIGNING AN OVERLAY TREE

Suppose you have the following files: main, a, b, c, d, e, f, g, h, i, and j. Say that main can call a,b,c, and d and that each of these files can call main. In addition, assume that routine e can be called from a, b, c, d, or main, but has no relationship to routine f. Thus, if a routine in e is to be run, a, b, c, and d need to be memory-resident as well. Routine f is like e; that is, it needs nothing in e to be present, but can be called from a, b, c, or d. This means that the overlay manager can share the memory space between routines e and f, since neither need ever be memory coresident with the other in order to run.

If you consider routine g as sharing the same space as the combination of a,b,c, and d and routines h,i and j sharing the same space, you have the basis for constructing the overlay tree for this program structure:

```
                  main (root level of the tree)
                  |                            |
        a,b,c,d (1,1)                  -----g (1,2)-----
        |          |                   |        |       |
    e (2,1)     f (2,2)            h (2,1)  i (2,2)  j(2,3)
```

We have drawn the tree and labeled its branches to match the hunk overlay (level,ordinate) numbers that are found in the hunk overlay table that matches the nodes to which they are assigned.

From the description above, you can see that if main is to call any routine in program segment a-d, all of those segments should be resident in memory at the same time. Thus, they have all been assigned to a single node by the linker. While a-d are resident, if you call routines in e, the linker will automatically load routine e from disk, and reinitialize the module (each time it is again brought in) so that its subroutines will be available to be run. If any segment a-d calls a routine in f, the linker replaces e with the contents of f and initializes it. Thus, a-d are at level 1 in the overlay tree, and routines e and f are at level 2, requiring that a-d be loaded before e or f can be accessed and loaded for execution.

Note: A routine can perform calls only to routines in other nodes that are currently memory resident (the ancestors of the node in which the routine now in use is located) or to routines in a direct child node. That is, main cannot call e directly, but e can call routines in main, because main is an ancestor.

Note also that within each branch of each subnode, for a given level, the ordinate numbers begin again with number 1.

DESCRIBING THE TREE

You create the tree by telling the overlay linker about its structure The numerical values, similar to those noted in the figure above, are assigned sequentially by the linker itself and appear in the hunk node table. Here is the sequence of overlay link statements that cause the figure above to be built:

```
OVERLAY
a,b,c,d
*e
*f
g
*h
*i
```

This description tells the linker that a, b, c, and d are part of a single node at a given level (in this case level 1), and the asterisks in front of e and f say that these are both on the next level down from a-d, and accessible only through a-d or anything closer to the root of the tree. The name g has no asterisk, so it is considered on the same level as a-d, telling the linker that either a-d or g will be memory resident, but not both simultaneously. Names h and i are shown to be related to g, one level down.

The above paragraphs have explained the origin of the hunk node level and the hunk ordinate in the hunk overlay symbol table.

SEEK OFFSET AMOUNT

The first value for each node in the overlay table is the seek offset As specified earlier, the overlay linker builds a large single file containing all of the overlay nodes. The seek offset number is that value that can be given to the seek(file, byte_offset) routine to point to the first byte of the hunk header of a node.

INITIALHUNK

The initialHunk value in the overlay symbol table is used by the overlay manager when unloading a node. It specifies the initial hunk that must have been loaded in order to have loaded the node that contains this symbol. When a routine is called at a different level and ordinate (unless it is a direct, next-level child of the current node), it will become necessary to free the memory used by invalid hunks, to make room to overlay with the hunk(s) containing the desired symbol.

SYMBOLHUNK AND SYMBOLOFFSETX

These table entries for the symbols are used by the overlay manager to actually locate the entry point once it either has determined it is already loaded or has loaded it. The symbolHunk shows in which hunk to locate the symbol. SymbolOffsetX-4 shows the offset from the start of that hunk at which the entry point is actually located.

ATOM: (Alink Temporary Object Modifier)

This document describes the ATOM utility, including its development history, the manner in which it has been implemented, and alternatives to its use.

Programmers want to be able to specify that parts of their program go into CHIP memory (the first 512K) so that the custom chips can access it. They also want to treat this data just like any other data in their program and therefore have it link and load normally.

PREVIOUS SOLUTIONS

Previously, the recommended way of dealing with this was to do an AllocMem with the chip memory bit set and copy data from where it was loaded ("fast" memory) to where it belonged (chip memory) and then use pointers to get to it. This involved having two copies of your data in memory-- the first loaded with your program, the second copied into the first 512K of memory.

The other solution was to have the program not run in machines with more than 512K. This will quickly become an unacceptable solution.

THE ATOM SOLUTION

The ATOM solution is as follows:

1) Compile or assemble normally

2) Pass the object code through a post- (or pre-) processor called "ATOM." ATOM will interact with the user and the object file(s). It will flag the desired hunks (or all hunks) as "for chip memory" by changing the hunk type.

3) The linker will now take nine hunk types instead of three. The old types were hunk_code, hunk_data, and hunk_bss. The new ones are:

```
hunk_code_chip  = hunk_code + bit 30 set
hunk_code_fast  = hunk_code + bit 31 set
hunk_data_chip  = hunk_data + bit 30 set
hunk_data_fast  = hunk_data + bit 31 set
```

```
hunk_bss_chip   = hunk_bss  + bit 30 set
hunk_bss_fast   = hunk_bss  + bit 31 set
```

The linker will pass all hunk types through to the LOADER, (coagulating if necessary). The LOADER uses the hunk header information when loading.

You may recall from the information provided in the linker documentation that CODE hunks contain executable (68000) machine language, DATA hunks contain initialized data (constants,...), and BSS hunks contained uninitialized data (arrays, variable declarations, ...).

4) The LOADER will load according to information from step 3) above. Hunks will go into the designated memory type.

5) Old versions of the LOADER will interpret the new hunk types as a VERY large hunk and not load (error 103, not enough memory)

FUTURE SOLUTIONS

The assembler and Lattice "C" may be changed to generate the new hunk types under programmer control to provide other solutions.

HOW THE BITS WORK

The hunk size is a word containing the number of words in the hunk. Therefore, for the foreseeable future, including 32-bit address space machines, the upper two bits are unused. The bits have been redefined as follows:

```
+------ Bit31   MEMF_FAST
|  +--- Bit30   MEMF_CHIP
|  |
v  v
0  0    If neither bit is set then get whatever memory
        is available, this is "backwards" compatible.
        Preference is given to FAST memory.
1  0    Loader must get FAST memory, or fail.
0  1    Loader must get CHIP memory, or fail.
1  1    If Bit31 and Bit30 are both set, there is
        extra information available following this long
        word. This is reserved for future expansion, as
        needed.  It is not currently used.
```

PERCEIVED IMPACT

Old programs, programs that have not been compiled or assembled with the new options, and programs that have not been run through ATOM will run as well as ever. This includes crashing in extended memory, if poorly programmed. The previous solutions mentioned at the beginning of this section still hold.

Program development and testing on a 512K machine could follow EXACTLY the same loop you have now--edit, compile, link, execute, test, edit, --UNTIL you are about to release. Then you edit, compile, ATOM, Alink, add external memory (>512K) and test. This works well for all three environments (Amiga, IBM, and Sun).

For native (Amiga) development on a >512K machine, you may want to ATOM the few required object files so you can both run your linked program in an extended memory machine and take advantage of a large RAM: disk. The development cycle then becomes: edit, compile, optionally ATOM (if this code or data contains items needed by the blitter), link, execute, test, edit...

"New programs" will not load in a V1.0 Kickstart environment. The result will be error 103--not enough memory. Old (V1.0 and before) versions of dumpobj and OMD will not work on files after ATOM has been run on them.

WORKING ENVIRONMENT

To get all of this to work together, you need Release 1.1-compatible copies of:

```
ATOM (Version 1.0 or later)
Alink (Version 3.30 or later)
Kickstart (Release 1.1 or later) for DOS LOADER.
DumpObj (Version 2.1) Needed if you wish to examine
  programs modified by ATOM.
```

ATOM COMMAND LINE SYNTAX

The command line syntax is:

```
ATOM <infile> <outfile> [-I]
        -- or --
ATOM <infile> <outfile> [-C[C|D|B]] [-F[C|D|B]] [-P[C|D|B]]
```

where:

```
<infile>        Represents an object file, just compiled,
                assembled or ATOMed (yes, you can re-ATOM
                an object file.
<outfile>       The destination for the converted file
-C              Change memory to CHIP
-F              Change memory to FAST
-P              Change memory to "public" (any type of memory
                available)
C               Change CODE hunks
D               Change DATA hunks
B               Change BSS hunks
```

COMMAND LINE EXAMPLES

Example #1:

In most cases there is no need to place CODE hunks in chip memory. Sometimes DATA and BSS hunks do need to be placed in chip memory; therefore, the following is a fairly common usage of ATOM. To cause all code hunks go into public RAM and data and BSS hunks to go into chip RAM type the following:

```
        ATOM infile.obj outfile.obj -pc -cdb
```

Example #2:

To cause all the hunks in object file to be loaded into chip memory type the following:

```
        ATOM infile.obj outfile.obj -c
```

Example #3:

To set all data hunks to load into chip memory type:

```
        ATOM myfile.o myfile.set.o -cd
```

Example #4:

This is an interactive example. User input is in lower case, computer output is in upper case. In this example the code hunk is set to FAST, the data hunk is set to CHIP. There were no BSS hunks. Note that help was requested in the beginning.

```
        2> atom from.o from.set -i
        AMIGA OBJECT MODIFIER V1.0

        UNIT NAME FROM
        HUNK NAME NONE
        HUNK TYPE CODE          {Note: code hunk}
        MEMORY ALLOCATION PUBLIC

        DISPLAY SYMBOLS? [Y/N] y
        __base..
        __xcovf.
        _CXD22..
        _printf.
        _main...

        MEMORY TYPE? [F|C|P] ?  {Note: request for help}

        Please enter F for fast        -
                     C for Chip --  Memory type.
                     P for Public      -
                     Q to quit     {cancels the operation, no
                                    output file is created}
                     W to windup   {does not change the rest
                                    of the file, just passes
                                    it through}
                     N for Next hunk  {skip this hunk,
                                       show next}

        MEMORY TYPE? [F|C|P] f

        UNIT NAME 0000
        HUNKNAME NONE
        HUNK TYPE DATA          {Note: data hunk}
        MEMORY ALLOCATION PUBLIC

        DISPLAY SYMBOLS? [Y/N] n

        MEMORY TYPE? [F|C|P] c

        UNIT NAME 0000
        HUNKNAME NONE
        HUNK TYPE BSS
        MEMORY ALLOCATION PUBLIC

        DISPLAY SYMBOLS? [Y/N] y

        MEMORY TYPE? [F|C|P] p
        2>_
```

ERROR MESSAGES

The error messages generated by ATOM are as follows:

Error Bad Args:
- a) an option does not start with a "-"
- b) wrong number of parameters
- c) "-" not followed by I, C, F or P.
- d) -x supplied in addition to -I etcetera.

Error Bad Infile:
File not found.

Error Bad Outfile:
File can not be created.

Error Bad Type ##: ATOM has detected a hunk type that it does not recognize. The object file may be corrupt.

Error Empty Input: Input file does not contain any data.

Error ReadExternals: External reference or definition if of an undefined type. Object file may be corrupt.

Error premature end of file: An end of file condition (out of data) was detected while ATOM was still expecting input. Object file may be corrupt.

Error This utility can only be used on files that have NOT been passed through ALINK:
The input file you specified has already been processed by the linker. External symbols have been removed and hunks coagulated. You need to run ATOM on the object files produced by the C compiler or Macro Assembler before they are linked.

Making New Disk Devices

To create a new disk device, you must construct a new device node as described in section 3.3.1 of the AmigaDos Technical manual. You must also write a device driver for the new disk device.

A device driver for a new disk device must mimic the calls that are performed by the trackdisk device (described in the Amiga ROM Kernel manual). It must include the ability to respond to commands such as read, write, seek, and return status information in the same way as described for the trackdisk driver.

For the following description, note that most pointers are of the type BPTR (as described in the AmigaDOS Technical Reference Manual), a machine pointer to some longword-aligned memory location (such as returned by AllocMem), shifted right by two.

Construct the new node with the following fields:

0	Next
0	dt_device
0	Task
0	Lock
0	Handler
210	Stacksize
10	Priority
BPTR to startup info	
Seglist	
0	Global vector
BSTR to name	

The BSTR to a name is a BCPL pointer to the name of your new device (such as HD0:) represented as the length of the string in the first byte, and the characters following.

The Seglist must be the segment list of the filing system task. To obtain this, you must access a field in the process base of one of the filing system tasks.

The code that follows can be used for this purpose:

```
UBYTE *port;

port = DeviceProc( "DF0:");   /* Returns msg port of filesystem task */
task = (struct Task *)(port-sizeof(struct Task);
                             /* Task structure is below port */
list = ( task.pr_Seglist ) << 2;  /* make machine ptr from SegArray */
segl = list[3];  /* third element in SegArray is filesystem seglist */
```

Next, you must set up the start-up information (again, remember to use BPTRs where needed). This information consists of a BPTR to three longwords that contain the following:

- Unit number (do not use unit zero)
- Device driver name, stored as a BPTR to the device driver name, which must be terminated by a null byte that is included in the count (e.g., 4/'H'/'D'/'0'/0)
- BPTR to disk information

The disk size information contains the following longword fields:

11	Size of table
128	Disk block size in long words (assuming 512-byte blocksize)
0	Sector origin (i.e., first sector is sector zero)
Number of surfaces	(E.g., 2 for floppy disk)
1	Number of sectors per block
Number of blocks per track	(E.g., 11 for floppy disk)
2	(Or more, indicating number of blocks to be reserved at start)
0	Preallocation factor
0	Interleave factor
Lowest cylinder number	(Commonly 0)
Highest cylinder number	(E.g., 79 for floppy disk)
5	(Or more, indicating number of cache blocks)

Finally, the device node must be attached to the end of the list (note the Next fields are all BPTRs) of device nodes within the Info substructure. Warning: the list to which this refers is not the same kind of list that is referenced in the Exec portion of the Amiga ROM Kernel Reference Manual, but is instead the kind of list described in the AmigaDOS Technical Reference Manual.

To partition a hard disk, you make two or more device nodes and set the lowest and highest cylinder numbers to partition the disk as desired.

Creating a New Device to Run under AmigaDOS

This section provides information about adding devices that are not part of the DOS filing system. The next section provides information about adding file-system-related devices (hard-disks, floppy disks)--that is, devices that DOS can use to read and write files with their associated directories. You would want to use this information to add a new device, such as a new serial port or a new parallel port. In this case, you may be creating a device named "SER2:" which is to act just like "SER:" as far as DOS is concerned.

Two steps are involved here. First, you must create a suitable device, which is a process that is not addressed here. Note: The code for creating a skeleton disk-resident device is contained in the Amiga ROM Kernel Reference Manual: Libraries and Devices.

Second, you must make this new device available as an AmigaDOS device. This process involves writing a suitable device handler (see Amiga ROM Kernel Reference Manual: Libraries and Devices) and installing it into the AmigaDOS structures.

This installation is handled by creating a suitable device node structure for your new device. This is similar to creating a DevInfo slot for a new disk device, except that the start-up argument can be anything you want, the Segment list slot is zero, and the file name of your disk-resident device handler is placed in the Filename slot.

```
        0       Next
        0       dt_device
        0       Task  (or process id - see below)
        0       Lock
        BSTR    Filename of handler code
        NNN     Stacksize required
        NN      Priority required
        XXX     Startup information
        0       SegList (nonzero if you load the code)
        0       Global vector required
        BSTR    Device Name
```

The device handler is the interface between your device and an application program. This is normally written in BCPL, and the AmigaDOS kernel will attempt to load the code of the handler and create a new process for it when it is first referenced. This is handled automatically when the kernel notices that the Task field in the DevInfo structure is zero. If the code is already loaded, the code segment pointer is placed in the SegList field. If this field is zero, the kernel loads the code from the filename given in the Filename field and updates the SegList field.

If you want this automatic loading and process initialization to work, you must create a code module, which is written in BCPL or is written in assembly language to look like a BCPL module. This ensures that the dynamic linking used by the kernel will work correctly.

If you are writing in assembly language, the format of the code section must be as shown below. Note that you may use DATA and BSS sections, but each section must have the same format as described here.

```
StartModule  DC.L  (EndModule-StartModule)/4   Size of module in lwords
EntryPoint   ....
             ....    (your code)
             CNOP    0,4                          Align to lword boundary
             DC.L    0                            End marker
             DC.L    1                            Define Global 1
             DC.L    EntryPoint-StartModule       Offset of entry point
             DC.L    1                            Highest global used
             END
```

In assembly language, you will be started with register D1 holding a BCPL pointer to the initial packet passed from the kernel.

If you are writing in BCPL, a skeleton routine such as the following will appear. The main job of the device handler is to convert Open, Read, Write, and Close requests into the device read and write requests. Other packet types are marked as an error.

```
// "Include files containing useful constants"

GET "LIBHDR"
GET "IOHDR"
GET "MANHDR"
GET "EXECHDR"

// This is a handler for a skeleton Task.

// When the task is created, the parameter packet contains
// the following.
// parm.pkt!pkt.arg1 = BPTR to BCPL string of device name, i.e.  ("SKEL:")
// parm.pkt!pkt.arg2 = extra info (if needed)
// parm.pkt!pkt.arg3 = BPTR to device info node

MANIFEST
$(
   IO.blocksize = 30 // size of devices IO block
$)

LET start(parm.pkt) BE
$(
   LET extrainfo = parm.pkt!pkt.arg2
   LET read.pkt = 0
   LET write.pkt = 0

   LET openstring = parm.pkt!pkt.arg1

   LET inpkt  = VEC pkt.res1
   LET outpkt = VEC pkt.res1
   LET IOB    = VEC IO.blocksize
   LET IOBO   = VEC IO.blocksize

   LET error   = FALSE
   LET devname = "serial.device*X00"

   LET open = FALSE  // flag to show whether device has been "opened"
                     // with act.findinput or act.findoutput

   LET node = parm.pkt!pkt.arg3

   // Zero the block first so that we can see what goes
   // into it when we call Opendevice
FOR i=0 TO IO.blocksize DO IOB!i := 0

IF OpenDevice( IOB, devname, 0, 0 ) = 0 THEN error := TRUE

IF error THEN
$( returnpkt(parm.pkt,FALSE,error,objectinuse)
   return
$)

// Copy all the necessary info to the Output buffer too.

FOR i=0 TO IO.blocksize DO IOBO!i := IOB!i

outpkt!pkt.type := act.write
inpkt!pkt.type  := act.read
node!dev.task := taskid()     // Insert process id into device node

// Finished with parameter packet...send back...

returnpkt( parm.pkt, TRUE )

// This is the main repeat loop waiting for an event

$( LET p = taskwait()

SWITCHON p!pkt.type INTO
$(

CASE act.findinput:      // Open
CASE act.findoutput:
$( LET scb = p!pkt.arg1
   open := TRUE
   scb!scb.id   := TRUE  // Interactive
   returnpkt(p,TRUE)
   LOOP
$)

CASE act.end:            // Close
   node!dev.task := 0    // Remove process id from device node
   open := FALSE
   returnpkt(p,TRUE)
   LOOP

CASE act.read:           // Read request returning
   inpkt := p
   handle.return(IOBO,read.pkt)
   LOOP

CASE act.write:          // Write request returning
   outpkt := p
   handle.return(IOBO,write.pkt)
   LOOP

CASE 'R':                // A read request
   read.pkt := p
   handle.request(IOB,IOC.read,p,inpkt)
   inpkt := 0
   LOOP

CASE 'W':                // A write request
   write.pkt := p
   handle.request(IOBO,IOC.write,p,outpkt)
   outpkt := 0
   LOOP

     DEFAULT:
   UNLESS open DO node!dev.task := 0     //Remove process id unless open
     $)
   $) REPEATWHILE open | outpkt = 0 | inpkt = 0

   // Termination

   CloseDevice( IOB )

$)

// Handle an IO request.  Passed command, transmission packet (tp)
// and request packet (rp).  rp contains buffer and length in arg2/3.
AND handle.request(IOB, command rp, tp ) BE
$(
   LET buff = rp!pkt.arg2
   LET len  = rp!pkt.arg3
   SetIO( IOB, command, ?, rp!pkt.arg3, 0 )
   putlong ( IOB, IO.data, buff )
   SendIO(IOB, tp )
$)

// Handle a returning IO request.  The user request packet is
// passed as p, and must be returned with success/failure message.
AND handle.return(IOB, p ) BE
$(
   LET errcode = IOB O.error
   LET len = getlong( IOB, IO.actual )

   TEST errcode = 0 THEN       // No error
      returnpkt(p, len )
   ELSE
      returnpkt(p, -1, errcode )
$)
```

If you wish to write your device handler in C, you cannot use the automatic load and process creation provided by the kernel. In this case, you must load the code yourself and use a call to CreateProc to create a process. The result from this call should be stored in the Task field of the DevInfo structure. You must then send a message to the new process to get it started. This message might contain such things as the unit number of the device involved. The handler process should then wait for Open, Read, Write, and Close calls and handle them as described in the example above. C code does not need to insert the process id into the device node, because this is done when code is loaded, as described above.

Using AmigaDOS Without Workbench/Intuition

This information is provided to give developers some information about how AmigaDOS and Intuition interact with each other. As of this writing, it is not possible to fully close down Intuition or the input.device. It is possible to install one's own input handler within the input stream (as is demonstrated in the Amiga ROM Kernel Reference Manual: Libraries and Devices, in the Input Device description) and thereby handle input events yourself, after your program has been loaded and started by AmigaDOS. If, after that point, you take over the machine in some manner, you can prevent AmigaDOS from trying to put up system requesters or otherwise interacting with the screen by modifying DOS as shown below. Basically, your own program must provide alternate ways to handle errors that would normally cause DOS to put up a requester. Another alternative for taking over the machine is to ignore the AmigaDOS filing system altogether, and use the trackdisk.device to boot your code and data on your own. You will find details about the disk boot block and the track formatting in the Amiga ROM Kernel Reference Manual: Libraries and Devices, allowing this alternate means if you so choose.

Here are the details about AmigaDOS and Intuition. AmigaDOS initializes itself and opens Intuition. It then attempts to open the configuration file (created by Preferences) and passes this to Intuition. It then opens the initial CLI window via Intuition and attempts to run the first CLI command. This is commonly a loadwb (load Workbench), followed by an endcli on the initial CLI.

An application program can be made to behave like Workbench in that it spawns off a new process. The next CLI command is then endcli, which closes everything down, leaving only the new process running (along with the file-system processes). This process would set the pr_WindowPtr field to -1, which indicates that the DOS should report errors quietly. Note that the application must handle all errors. There are further details on this in the AmigaDOS Technical Reference Manual, chapter 3. AmigaDOS will also have initialized the TrapHandler field of the user task to point to code that will display a requester after an error; this should be replaced by a user-provided routine. This will stop all uses of Intuition from the user task, provided no serious memory corruption problems are found, in which case AmigaDOS will call Exec Alert directly.

There is still the problem that the file-system processes may ask for a requester, in the event of a disk error or if the file-system task crashes due to memory corruption. To stop this, the pr_WindowPtr and tc_TrapHandler fields of the file-system tasks must be set to -1 and a private Trap handler must be provided in the same way as was done for the user task. This is easily done as shown below.

Find the message port for each file-system task by calling DeviceProc(), passing DF0, DF1, etc. An error indicates that the device is not present. From the message port you can find the task base for each file-system task and then patch these two slots. This should be repeated for each disk unit.

The application program can now close Intuition. Workbench has, of course, never been invoked. Note that as of this writing, it is not possible to stop AmigaDOS from opening Intuition.

Note that if the application wants to use any other device, such as SER:, the handler process must be patched in exactly the same way as the file-system processes. The application obviously should not attempt to open the CON: or RAW: once Intuition has become inactive.

"EA IFF 85" Standard for Interchange Format Files

Document Date:	January 14, 1985
From:	Jerry Morrison, Electronic Arts
Status of Standard:	Released and in use

INTRODUCTION

Standards Are Good for Software Developers

As home computer hardware improves, the demand increases for higher-quality, more detailed data. Data development becomes more expensive, requires more expertise and better tools, and has to be shared across projects. Think about several ports of a product on one CD-ROM with 500M bytes of common data!

Development tools need standard interchange file formats. Imagine scanning in images of "player" shapes, moving them to a paint program for editing, then incorporating them into a game. Or writing a theme song with a Macintosh score editor and incorporating it into an Amiga game. The data must at times be transformed, clipped, filled out, and moved across machine kinds. Media projects will depend on data transfer from graphic, music, sound effect, animation, and script tools.

Standards Are Good for Software Users

Customers should be able to move their own data between independently developed software products. They should also be able to buy data libraries usable across many such products. The types of data objects to exchange are open-ended and include plain and formatted text, raster and structured graphics, fonts, music, sound effects, musical instrument descriptions, and animation.

The problem with expedient file formats, typically memory dumps, is that they are too provincial. Designing data for one particular use (e.g., a screen snapshot), programmers preclude future expansion (would you like a full-page picture? a multipage document?). In neglecting the possibility that other programs might read their data, they fail to save contextual information (how many bit planes? what resolution?). Ignoring the possibility that other programs might create such files, they write programs that are intolerant of extra data (texture palette for a picture editor), missing data (no color map), or minor variations (smaller image). In practice, a filed representation should rarely mirror an in-memory representation. The former should be designed for longevity; the latter to optimize the manipulations of a particular program. The same filed data will be read into different memory formats by different programs.

The IFF philosophy: "A little behind-the-scenes conversion when programs read and write files is far better than NxM explicit conversion utilities for highly specialized formats." So we need some standardization for data interchange among development tools and products. The more developers that adopt a standard, the better for all of us and our customers.

EA IFF 1985

Here is our offering: Electronic Arts' IFF standard for Interchange File Format. Alternatives and justifications are included for certain choices. Public domain subroutine packages and utility programs are available to make it easy to write and use IFF-compatible programs.

References and Trademarks

American National Standard Additional Control Codes for Use with ASCII, ANSI standard 3.64-1979 for an 8-bit character set. See also ISO standard 2022 and ISO/DIS standard 6429.2.

Amiga[tm] is a trademark of Commodore-Amiga, Inc.

C, A Reference Manual, Samuel P. Harbison and Guy L. Steele Jr., Tartan Laboratories. Prentice-Hall, Englewood Cliffs, NJ, 1984.

Compiler Construction, An Advanced Course, edited by F. L. Bauer and J. Eickel (Springer-Verlag, 1976). This book is one of many sources for information on recursive descent parsing.

DIF Technical Specification (Software Arts, Inc., 1981) DIF[tm] is the format for spreadsheet data interchange developed by Software Arts, Inc. DIF[tm] is a trademark of Software Arts, Inc.

Electronic Arts[tm] is a trademark of Electronic Arts.

"FTXT" IFF Formatted Text, from Electronic Arts. IFF supplement document for a text format.

Inside Macintosh (Apple Computer, Inc., 1985). This is a programmer's reference manual. Apple(R) is a trademark of Apple Computer, Inc. Macintosh[tm] is a trademark licensed to Apple Computer, Inc.

"ILBM" IFF Interleaved Bitmap, from Electronic Arts. IFF supplement document for a raster image format.

M68000 16/32-Bit Microprocessor Programmer's Reference Manual (Motorola, Inc., 1984).

PostScript Language Manual (Adobe Systems Inc., 1984). PostScript[tm] is a trad emark of Adobe Systems, Inc. Times and Helvetica(R) are trademarks of Allied Corporation.

InterScript: A Proposal for a Standard for the Interchange of Editable Documents (Xerox Corporation, 1984).

Introduction to InterScript (Xerox Corporation, 1985).

BACKGROUND

What Do We Need?

A standard should be long on prescription and short on overhead. It should give lots of rules for designing programs and data files for synergy. But neither the programs nor the files should cost too much more than the expedient variety. Although we are looking to a future with CD-ROMs and perpendicular recording, the standard must work well on floppy disks.

For program portability, simplicity, and efficiency, formats should be designed with more than one implementation style in mind. (In practice, pure stream I/O is adequate, although random access makes it easier to write files.) It ought to be possible to read one of many objects in a file without scanning all the preceding data. Some programs need to read and play out their data in real time, so we need good compromises between generality and efficiency.

As much as we need standards, they cannot hold up product schedules. So we also need a kind of decentralized extensibility that allows any software developer to define and refine new object types without some "standards authority" in the loop. Developers must be able to extend existing formats in a forward- and backward-compatible way. A central repository for design information and example programs can help us take full advantage of the standard.

For convenience, data formats should heed the restrictions of various processors and environments. For example, word-alignment greatly helps 68000 access at insignificant cost to 8088 programs.

Other goals include the ability to share common elements over a list of objects and the ability to construct composite objects containing other data objects with structural information such as directories.

And finally, "Simple things should be simple and complex things should be possible." Alan Kay.

Think Ahead

Let's think ahead and build programs that read and write files for programs, including those others yet to be designed. Let's build data formats to last for future computers, so long as the overhead is acceptable. This extends the usefulness and life of today's programs and data.

To maximize interconnectivity, the standard file structure and the specific object formats must all be general and extensible. Think ahead when designing an object. Objects should serve many purposes and allow many programs to store and read back all the information they need; even squeeze in custom data. Then a programmer can store the available data and is encouraged to include fixed contextual details. Recipient programs can read the needed parts, skip unrecognized stuff, default missing data, and use the stored context to help transform the data as needed.

Scope

IFF addresses these needs by defining a standard file structure, some initial data object types, ways to define new types, and rules for accessing these files. We can accomplish a great deal by writing programs according to this standard, but do not expect direct compatibility with existing software. We will need conversion programs to bridge the gap from the old world.

IFF is geared for computers that readily process information in 8-bit bytes. It assumes a "physical layer" of data storage and transmission that reliably maintains "files" as strings of 8-bit bytes. The standard treats a "file" as a container of data bytes and is independent of how to find a file and whether it has a byte count.

This standard does not by itself implement a clipboard for cutting and pasting data between programs. A clipboard needs software to mediate access, to maintain a "contents version number" so programs can detect updates, and to manage the data in "virtual memory."

Data Abstraction

The basic problem is how to represent information in a way that's program-independent, compiler-independent, machine-independent, and device-independent.

The computer science approach is "data abstraction," also known as "objects," "actors," and "abstract data types." A data abstraction has a "concrete representation" (its storage format), an "abstract representation" (its capabilities and uses), and access procedures that isolate all the calling software from the concrete representation. Only the access procedures touch the data storage. Hiding mutable details behind an interface is called "information hiding". What data abstraction does is abstract from details of implementing the object, namely the selected storage representation and algorithms for manipulating it.

The power of this approach is modularity. By adjusting the access procedures we can extend and restructure the data without impacting the interface or its callers. Conversely, we can extend and restructure the interface and callers without making existing data obsolete. It's great for interchange!

But we seem to need the opposite: fixed file formats for all programs to access. Actually, we could file data abstractions ("filed objects") by storing the data and access procedures together. We'd have to encode the access procedures in a standard machine-independent programming language la PostScript. Even still, the interface can't evolve freely since we can't update all copies of the access procedures. So we'll have to design our abstract representations for limited evolution and occasional revolution (conversion).

In any case, today's microcomputers can't practically store data abstractions. They can do the next best thing: store arbitrary types of data in "data chunks", each with a type identifier and a length count. The type identifier is a reference by name to the access procedures (any local implementation). The length count enables storage-level object operations like "copy" and "skip to next" independent of object type.

Chunk writing is straightforward. Chunk reading requires a trivial parser to scan each chunk and dispatch to the proper access/conversion procedure. Reading chunks nested inside other chunks requires recursion, but no lookahead or backup.

That's the main idea of IFF. There are, of course, a few other details.

Previous Work

Where our needs are similar, we borrow from existing standards.

Our basic need to move data between independently developed programs is similar to that addressed by the Apple Macintosh desk scrap or "clipboard" [Inside Macintosh chapter "Scrap Manager"]. The Scrap Manager works closely with the Resource Manager, a handy filer and swapper for data objects (text strings, dialog window templates, pictures, fonts, etc.), including types yet to be designed [Inside Macintosh chapter "Resource Manager"]. The Resource Manager is akin to Smalltalk's object swapper.

We will probably write a Macintosh desk accessory that converts IFF files to and from the Macintosh clipboard for quick and easy interchange with programs like MacPaint and Resource Mover.

Macintosh uses a simple and elegant scheme of four-character "identifiers" to identify resource types, clipboard format types, file types, and file creator programs. Alternatives are unique ID numbers assigned by a central authority or by hierarchical authorities, unique ID numbers generated by algorithm, other fixed length character strings, and variable length strings. Character string identifiers double as readable signposts in data files and programs. The choice of four characters is a good tradeoff between storage space, fetch/compare/store time, and name space size. We'll honor Apple's designers by adopting this scheme.

"PICT" is a good example of a standard structured graphics format (including raster images) and its many uses [Inside Macintosh chapter "QuickDraw"]. Macintosh provides QuickDraw routines in ROM to create, manipulate, and display PICTs. Any application can create a PICT by simply asking QuickDraw to record a sequence of drawing commands. Because it is just as easy to ask QuickDraw to render a PICT to a screen or a printer, it is very effective to pass them between programs, say from an illustrator to a word processor. An important feature is the ability to store "comments" in a PICT that QuickDraw will ignore. Actually, it passes them to your optional custom "comment handler."

PostScript, Adobe's print file standard, is a more general way to represent any print image (which is a specification for putting marks on paper) [PostScript Language Manual]. In fact, PostScript is a full-fledged programming language. To interpret a PostScript program is to render a document on a raster output device. The language is defined in layers: a lexical layer of identifiers, constants, and operators; a layer of reverse polish semantics including scope rules and a way to define new subroutines; and a printing-specific layer of built-in identifiers and operators for rendering graphic images. It is clearly a powerful (Turing equivalent) image definition language. PICT and a subset of PostScript are candidates for structured graphics standards.

A PostScript document can be printed on any raster output device (including a display) but cannot generally be edited. That is because the original flexibility and constraints have been discarded. Besides, a PostScript program may use arbitrary computation to supply parameters such as placement and size to each operator. A QuickDraw PICT, in comparison, is a more restricted format of graphic primitives parameterized by constants. So a PICT can be edited at the level of the primitives--for example, a line can be moved or thickened. It cannot be edited at the higher level of, say, the bar chart data that generated the picture.

PostScript has another limitation: Not all kinds of data amount to marks on paper. A musical instrument description is one example. PostScript is just not geared for such uses.

"DIF" is another example of data being stored in a general format usable by future programs [DIF Technical Specification]. DIF is a format for spreadsheet data interchange. DIF and PostScript are both expressed in plain ASCII text files. This is very handy for printing, debugging, experimenting, and transmitting across modems. It can have substantial cost in compaction and read/write work, depending on use. We will not store IFF files this way; but we could define an ASCII alternate representation with a converter program.

InterScript is Xerox' standard for interchange of editable documents [Introduction to InterScript]. It approaches a harder problem: how to represent editable word processor documents that may contain formatted text, pictures, cross-references such as figure numbers, and even highly specialized objects such as mathematical equations. InterScript aims to define one standard representation for each kind of information. Each InterScript-compatible editor is supposed to preserve the objects it does not understand and even maintain nested cross-references. So a simple word processor would let you edit the text of a fancy document without discarding the equations or disrupting the equation numbers.

Our task is similarly to store high level information and preserve as much content as practical while moving it between programs. But we need to span a larger universe of data types and cannot expect to centrally define them all. Fortunately, we do not need to make programs preserve information that they do not understand. And for better or worse, we do not have to tackle general-purpose cross-references yet.

PRIMITIVE DATA TYPES

Atomic components, such as integers and characters, that are interpretable directly by the CPU are specified in one format for all processors. We chose a format that's most convenient for the Motorola MC68000 processor [M68000 16/32-Bit Microprocessor Programmer's Reference Manual].

This section dictates the format for "primitive" data types when and only when they are used in the overall file structure and in standard kinds of chunks (see also "Chunks"). The number of such occurrences will be small enough that the costs of conversion, storage, and management of processor-specific files would far exceed the costs of conversion during I/O by "foreign" programs. A particular data chunk may be specified with a different format for its internal primitive types or with processor- or environment-specific variants if necessary to optimize local usage. Because that hurts data interchange, it is not recommended. (see "Designing New Data Sections" for more information.)

Alignment

All data objects larger than a byte are aligned on even byte addresses relative to the start of the file. This may require padding. Pad bytes are to be written as zeros, but do not count on that when reading. This means that every odd-length "chunk" (see below) must be padded so that the next one will fall on an even boundary. Also, designers of structures to be stored in chunks should include pad fields where needed to align every field larger than a byte. Zeros should be stored in all the pad bytes.

Justification: Even-alignment causes a little extra work for files that are used only on certain processors, but it allows 68000 programs to construct and scan the data in memory and do block I/O. You just add an occasional pad field to data structures that you are going to block read/write or stream read/write an extra byte. And the same source code works on all processors. Unspecified alignment, on the other hand, would force 68000 programs to (dis)assemble word and long-word data one byte at a time. Pretty cumbersome in a high-level language. And if you do not conditionally compile that out for other processors, you will not gain anything.

Numbers

Numeric types supported are two's-complement binary integers in the format used by the MC68000 processor--high byte first, high word first --the reverse of 8088 and 6502 format. These types could potentially include signed and unsigned 8, 16, and 32 bit integers, but the standard only uses the following:

UBYTE	8 bits unsigned
WORD	16 bits signed
UWORD	16 bits unsigned
LONG	32 bits signed

The actual type definitions depend on the CPU and the compiler. In this document, we'll express data type definitions in the C programming language. [See C, A Reference Manual.] In 68000 Lattice C:

```
typedef unsigned char   UBYTE;  /*  8 bits unsigned */
typedef short    WORD;  /* 16 bits signed */
typedef unsigned short  UWORD;  /* 16 bits unsigned */
typedef long     LONG;  /* 32 bits signed */
```

Characters

The following character set is assumed wherever characters are used, for example, in text strings, IDs, and TEXT chunks (see below). Characters are encoded in 8-bit ASCII. Characters in the range NUL (hex 0) through DEL (hex 7F) are well defined by the 7-bit ASCII standard. IFF uses the graphic group RJS (SP, hex 20) through R~S (hex 7E).

Most of the control character group hex 01 through hex 1F has no standard meaning in IFF. The control character LF (hex 0A) is defined as a "newline" character. It denotes an intentional line break--that is, a paragraph or line terminator. (There is no way to store an automatic line break. That is strictly a function of the margins in the environment the text is placed.) The control character ESC (hex 1B) is a reserved escape character under the rules of ANSI standard 3.64-1979 American National Standard Additional Control Codes for Use with ASCII, ISO standard 2022, and ISO/DIS standard 6429.2.

Characters in the range hex 7F through hex FF are not globally defined in IFF. They are best left reserved for future standardization. But note that the FORM type FTXT (formatted text) defines the meaning of these characters within FTXT forms. In particular, character values hex 7F through hex 9F are control codes, and characters hex A0 through hex FF are extended graphic characters such as A, as per the ISO and ANSI standards cited above. [See the "FTXT" IFF Formatted Text document in this appendix.]

Dates

A "creation date" is defined as the date and time a stream of data bytes was created. (Some systems call this a "last modified date.") Editing some data changes its creation date. Moving the data between volumes or machines does not. The IFF standard date format will be one of those used in MS-DOS, Macintosh, or Amiga DOS (probably a 32-bit unsigned number of seconds since a reference point). Issue: Investigate these three date formats.

Type IDs

A "type ID," "property name," "FORM type," or any other IFF identifier is a 32-bit value: the concatenation of four ASCII characters in the range R S (SP, hex 20) through R~S (hex 7E). Spaces (hex 20) should not precede printing characters; trailing spaces are permitted. Control characters are forbidden.

```
typedef CHAR ID[4];
```

IDs are compared using a simple 32-bit case-dependent equality test.

Data section type IDs (aka FORM types) are restricted IDs (see "Data Sections."). Because they may be stored in filename extensions (see

"Single Purpose Files"), lower-case letters and punctuation marks are forbidden. Trailing spaces are permitted.

Carefully choose those four characters when you pick a new ID. Make them mnemonic so that programmers can look at an interchange format file and figure out what kind of data it contains. The name space makes it possible for developers scattered around the globe to generate ID values with minimal collisions so long as they choose specific names like "MUS4" instead of general ones like "TYPE" and "FILE". EA will "register" new FORM type IDs and format descriptions as they are devised, but collisions will be improbable so there will be no pressure on this "clearinghouse" process. A list of currently defined IDs is provided at the end of this appendix.

Sometimes it is necessary to make data format changes that are not backward-compatible. Because IDs are used to denote data formats in IFF, new IDs are chosen to denote revised formats. Because programs will not read chunks whose IDs they do not recognize (see "Chunks" below), the new IDs keep old programs from stumbling over new data. The conventional way to chose a "revision" ID is to increment the last character if it is a digit or to change the last character to a digit. For example, first and second revisions of the ID "XY" would be "XY1" and "XY2." Revisions of "CMAP" would be "CMA1" and "CMA2."

Chunks

Cnunks are the building blocks in the IFF structure. The form expressed as a C typedef is:

```
typedef struct {
        ID      ckID;
        LONG    ckSize; /* sizeof(ckData) */
        UBYTE   ckData[/* ckSize */];
        } Chunk;
```

We can diagram an example chunk, a "CMAP" chunk containing 12 data bytes, like this:

```
                ----------------
        ckID:   |     'CMAP'    |
        ckSize: |       12      |
        ckData: | 0, 0, 0, 32   |   --------
                | 0, 0, 64, 0   |   12 bytes
                | 0, 0, 64, 0   |   ---------
                ----------------
```

The fixed header part means "Here is a type ckID chunk with ckSize bytes of data."

The ckID identifies the format and purpose of the chunk. As a rule, a program must recognize ckID to interpret ckData. It should skip over all unrecognized chunks. The ckID also serves as a format version number as long as we pick new IDs to identify new formats of ckData (see above).

The following ckIDs are universally reserved to identify chunks with particular IFF meanings: "LIST", "FORM", "PROP", "CAT ", and "   ". The special ID "   " (4 spaces) is a ckID for "filler" chunks, that is, chunks that fill space but have no meaningful contents. The IDs "LIS1" through "LIS9", "FOR1" through "FOR9", and "CAT1" through "CAT9" are reserved for future "version number" variations. All IFF-compatible software must account for these 23 chunk IDs. A list of predefined IDs is provided at the end of this document.

The ckSize is a logical block size--the number of data bytes in ckData. If ckData is an odd number of bytes long, a 0 pad byte follows, which is not included in ckSize. (See "Alignment.") A chunk's total physical size is ckSize rounded up to an even number plus the size of the header. Thus the smallest chunk is 8 bytes long with ckSize = 0. For the sake of following chunks, programs must respect every chunk's ckSize as a virtual end-of-file for reading its ckData even if that data is malformed, for example, if nested contents are truncated.

We can describe the syntax of a chunk as a regular expression with "#" representing the ckSize, that is, the length of the following {braced} bytes. The "[0]" represents a sometimes-needed pad byte. (The regular expressions in this document are collected later, along with an explanation of notation.)

```
        Chunk   ::= ID #{ UBYTE* } [0]
```

One chunk output technique is to stream-write a chunk header, stream write the chunk contents, then random-access back to the header to fill in the size. Another technique is to make a preliminary pass over the data to compute the size, then write it out all at once.

Strings, String Chunks, and String Properties

In a string of ASCII text, LF denotes a forced line break (paragraph or line terminator). Other control characters are not used. (See "Characters.") The ckID for a chunk that contains a string of plain, unformatted text is "TEXT." As a practical matter, a text string should probably not be longer than 32767 bytes. The standard allows up to 2**31 - 1 bytes.

When used as a data property (see below), a text string chunk may be 0 to 255 characters long. Such a string is readily converted to a C string or a Pascal STRING[255]. The ckID of a property must be the property name, not "TEXT".

When used as a part of a chunk or data property, restricted C string format is normally used. That means 0 to 255 characters followed by a NUL byte (ASCII value 0).

Data Properties

Data properties specify attributes for following (non-property) chunks. A data property essentially says "identifier = value," for example "XY = (10, 200)," telling something about following chunks. Properties may appear only inside data sections ("FORM" chunks--see "Data Sections") and property sections ("PROP" chunks--see "Group PROP").

The form of a data property is a special case of Chunk. The ckID is a property name as well as a property type. The ckSize should be small, because data properties are intended to be accumulated in RAM when reading a file. (256 bytes is a reasonable upper bound.) Syntactically:

```
        Property        ::= Chunk
```

When designing a data object, use properties to describe context information, such as the size of an image, even if they do not vary in your program. Other programs will need this information.

Think of property settings as assignments to variables in a programming language. Multiple assignments are redundant and local assignments temporarily override global assignments. The order of assignments does not matter as long as they precede the affected chunks. (See "LISTs," "CATs," and "Shared Properties.")

Each object type (FORM type) is a local name space for property IDs. Think of a "CMAP" property in a "FORM ILBM" as the qualified ID "ILBM.CMAP". Property IDs specified when an object type is designed (and therefore known to all clients) are called "standard;" specialized ones added later are "nonstandard."

Links

Issue: A standard mechanism for "links" or "cross references" is very desirable for processes such as combining images and sounds into animations. Perhaps we will define "link" chunks within FORMs that refer to other FORMs or to specific chunks within the same and other FORMs. This needs further work. EA IFF 1985 has no standard link mechanism.

For now, it may suffice to read a list of, say, musical instruments, and then just refer to them within a musical score by index number.

File References

Issue: We may need a standard form for references to other files. A "file ref" could name a directory and a file in the same type of operating system as the ref's originator. The reference would imply that the file was on some mounted volume. In a network environment, a file ref could name a server, too.

Issue: How can we express operating-system independent file refs?

Issue: What about a means to reference a portion of another file? Would this be a "file ref" plus a reference to a "link" within the target file?

DATA SECTIONS

The first thing a file must tell us is whether it contains IFF data and, if so, whether it contains the kind of data we are looking for. This brings up the notion of a "data section". A "data section," or "IFF "FORM " is one self-contained "data object" that might be stored in a file by itself. It is one high-level data object such as a picture or a sound effect. The IFF structure "FORM" makes it self-identifying. It could be a composite object such as a musical score with nested musical instrument descriptions.

Group FORM

A data section is a chunk with ckID "FORM" and this arrangement:

```
FORM         ::= "FORM" #{ FormType (LocalChunk | FORM | LIST | CAT)* }
FormType     ::= ID
LocalChunk   ::= Property | Chunk
```

The ID "FORM" is a syntactic keyword like "struct" in C. Think of a "struct ILBM" containing a field "CMAP". If you see "FORM" you will know to expect a FORM type ID (the structure name, "ILBM", in this example) and a particular contents arrangement or "syntax" (local chunks, FORMs, LISTs, and CATs). (LISTs and CATs are discussed below.) A "FORM ILBM", in particular, might contain a local chunk "CMAP", an "ILBM.CMAP" (to use a qualified name).

So the chunk ID "FORM" indicates a data section. It implies that the chunk contains an ID and some number of nested chunks. In reading a FORM, as when reading any other chunk, programs must respect its ckSize as a virtual end-of-file for reading its contents, even if they are truncated.

The FormType (or FORM type) is a restricted ID that may not contain lower-case letters or punctuation characters. (See "Type IDs" and "Single-purpose Files.")

The type-specific information in a FORM is composed of its "local chunks": data properties and other chunks. Each FORM type is a local name space for local chunk IDs. Thus, "CMAP" local chunks in other FORM types may be unrelated to "ILBM.CMAP". More than that, each FORM type defines semantic scope. If you know what a FORM ILBM is, you will know what an ILBM.CMAP is.

Local chunks defined when the FORM type is designed (and therefore known to all clients of this type) are called "standard," and specialized ones added later are "nonstandard."

Among the local chunks, property chunks give settings for various details such as text font while the other chunks supply the essential information. This distinction is not clear cut. A property setting canceled by a later setting of the same property has effect only on data chunks in between. For example, in the sequence:

```
        prop1 = x   (propN = value)*  prop1 = y
```

where the propNs are not prop1, the setting prop1 = x has no effect.

The following universal chunk IDs are reserved inside any FORM: "LIST", "FORM", "PROP", "CAT ", "JJJJ", "LIS1" through "LIS9", "FOR1" through "FOR9", and "CAT1" through "CAT9". (See "Chunks," "Group LIST", and "Group PROP.") For clarity, these universal chunk names may not be FORM type IDs.

A later section talks about grouping FORMs into LISTs and CATs, which let you group a bunch of FORMs without imposing any particular meaning or constraints on the grouping.

Composite FORMs

A FORM chunk inside a FORM is a full-fledged data section. This means you can build a composite object such as a multiframe animation sequence from available picture FORMs and sound effect FORMs. You can insert additional chunks with information like frame rate and frame count. Using composite FORMs, you leverage on existing programs that create and edit the component FORMs. Those editors may even look into your composite object to copy out its type of component, although few programs are fancy enough to do that. Such editors are not allowed to replace their component objects within your composite object. That is because the IFF standard lets you specify consistency requirements for the composite FORM, such as maintaining a count or a directory of the components. Only programs that are written to uphold the rules of your FORM type should create or modify such FORMs.

Therefore, in designing a program that creates composite objects, you are strongly requested to provide a facility for your users to import and export the nested FORMs. Import and export could move the data through a clipboard or a file.

Here are several existing FORM types and rules for defining new ones.

FTXT

An FTXT data section contains text with character-formatting information such as fonts and faces. It has no paragraph- or document-formatting information such as margins and page headers. FORM FTXT is well matched to the text representation in Amiga's Intuition environment. See the supplemental document "FTXT" IFF Formatted Text.

ILBM

"ILBM" is an InterLeaved BitMap image with color map; a machine-independent format for raster images. FORM ILBM is the standard image file format for the Commodore-Amiga computer and is useful in other environments, too. See the supplemental document "ILBM" IFF Interleaved Bitmap.

PICS

The data chunk inside a "PICS" data section has ID "PICT" and holds a QuickDraw picture. Issue: Should we allow more than one PICT in a PICS? See Inside Macintosh chapter "QuickDraw" for details on PICTs and how to create and display them on the Macintosh computer.

The only standard property for PICS is "XY", an optional property that indicates the position of the PICT relative to "the big picture." The contents of an XY is a QuickDraw Point.

Note: PICT may be limited to Macintosh use, in which case another format will be used for structured graphics in other environments.

Other Macintosh Resource Types

Some other Macintosh resource types could be adopted for use within IFF files; perhaps MWRT, ICN, ICN#, and STR#.

Issue: Consider the candidates and reserve some more IDs.

Designing New Data Sections

Supplemental documents will define additional object types. A supplement needs to specify the object's purpose, its FORM type ID, the IDs and formats of standard local chunks, and rules for generating and interpreting the data. It is a good idea to supply typedefs and an example source program that accesses the new object. See "ILBM" IFF Interleaved Bitmap for a good example.

Anyone can pick a new FORM type ID but should reserve it with Electronic Arts at their earliest convenience. Although decentralized format definitions and extensions are possible in IFF, our preference is to get design consensus by committee, implement a program to read and write it, perhaps tune the format, and then publish the format with example code. Some organization should remain in charge of answering questions and coordinating extensions to the format.

If it becomes necessary to revise the design of some data section, its FORM type ID will serve as a version number (See "Type IDs"). For example, a revised "VDEO" data section could be called "VDE1". But try to get by with compatible revisions within the existing FORM type.

In a new FORM type, the rules for primitive data types and word-alignment (See "Primitive Data Types") may be overridden for the contents of its local chunks but not for the chunk structure itself if your documentation spells out the deviations. If machine-specific type variants are needed--for example, to store vast numbers of integers in reverse bit order--outline the conversion algorithm and indicate the variant inside each file, perhaps via different FORM types. Needless to say, variations should be minimized.

In designing a FORM type, encapsulate all the data that other programs will need to interpret your files. For example, a raster graphics image should specify the image size even if your program always uses 320 x 200 pixels x 3 bitplanes. Receiving programs are then empowered to append or clip the image rectangle, to add or drop bitplanes, etc. This enables a lot more compatibility.

Separate the central data (such as musical notes) from more specialized information (such as note beams) so that simpler programs can extract the central parts during read-in. Leave room for expansion so that other programs can squeeze in new kinds of information (such as lyrics). And remember to keep the property chunks manageably short--say 256 bytes or less.

When designing a data object, try to strike a good compromise between a super-general format and a highly specialized one. Fit the details to at least one particular need--for example, a raster image might as well store pixels in the current machine's scan order. But add the kind of generality that makes it usable with foreseeable hardware and software. For example, use a whole byte for each red, green, and blue color value even if this year's computer has only four-bit video DACs. Think ahead and help other programs whenever the overhead is acceptable. For example, run compress a raster by scan line rather than as a unit so future programs can swap images by scan line to and from secondary storage.

Try to design a general-purpose "least common multiple" format that encompasses the needs of many programs without getting too complicated. Let's coalesce our uses around a few such formats widely separated in the vast design space. Two factors make this flexibility and simplicity practical. First, file storage space is becoming plentiful, so compaction is not a priority. Second, nearly any locally-performed data conversion work during file reading and writing will be cheap compared to the I/O time.

It must be all right to copy a LIST or FORM or CAT intact--for example, to incorporate it into a composite FORM. So any kind of internal references within a FORM must be relative references. They could be relative to the start of the containing FORM, relative from the referencing chunk, or a sequence number in a collection.

With composite FORMs, you leverage on existing programs that create and edit the components. If you write a program that creates composite objects, please provide a facility for your users to import and export the nested FORMs. The import and export functions may move data through a separate file or a clipboard.

Finally, do not forget to specify all implied rules in detail.

LISTS, CATS, AND SHARED PROPERTIES

Data often needs to be grouped together like a list of icons. Sometimes a trick like arranging little images into a big raster works, but generally data will need to be structured as a first class group. The objects "LIST" and "CAT" are IFF-universal mechanisms for this purpose.

Property settings sometimes need to be shared over a list of similar objects. For example, a list of icons may share one color map. LIST provides a means called "PROP" to do this. One purpose of a LIST is to define the scope of a PROP. A "CAT," on the other hand, is simply a concatenation of objects.

Simpler programs may skip LISTs and PROPs altogether and just handle FORMs and CATs. All "fully-conforming" IFF programs also know about "CAT ," "LIST," and "PROP." Any program that reads a FORM inside a LIST must process shared PROPs to correctly interpret that FORM.

Group CAT

A CAT is just an untyped group of data objects.

Structurally, a CAT is a chunk with chunk ID "CAT " containing a "contents type" ID followed by the nested objects. The ckSize of each contained chunk is essentially a relative pointer to the next one.

```
    CAT             ::= "CAT " #{ ContentsType (FORM | LIST | CAT)* }
    ContentsType    ::= ID    -- a hint or an "abstract data type" ID
```

In reading a CAT, as in reading any other chunk, programs must respect its ckSize as a virtual end-of-file for reading the nested objects even if they are malformed or truncated.

The "contents type" following the CAT's ckSize indicates what kind of FORMs are inside. So a CAT of ILBMs would store "ILBM" there. This is just a hint. It may be used to store an "abstract data type." A CAT could just have blank contents ID ("JJJJ") if it contains more than one kind of FORM.

CAT defines only the format of the group. The group's meaning is open to interpretation. This arrangement is like that needed by listing in LISP: the structure of cells is predefined but the meaning of the contents--such as, say, an association list--depends on use. If you need a group with an enforced meaning (an "abstract data type" or Smalltalk "subclass") or one with some consistency constraints or additional data chunks, use a composite FORM instead (see "Composite FORMs").

Because a CAT just means a concatenation of objects, CATs are rarely nested. Programs should really merge CATs rather than nest them.

Group LIST

A LIST defines a group much like CAT does but it also gives a scope for PROPs (see below). Unlike CATs, LISTs should not be merged unless their contents are understood.

Structurally, a LIST is a chunk with ckID "LIST" containing a "contents type" ID, optional shared properties, and the nested contents (FORMs, LISTs, and CATs), in that order. The ckSize of each contained chunk is a relative pointer to the next one. A LIST is not an arbitrary linked list; the cells are simply concatenated.

```
    LIST          ::= "LIST" #{ ContentsType PROP* (FORM | LIST | CAT)* }
    ContentsType  ::= ID
```

Group PROP

PROP chunks may appear in LISTs (not in FORMs or CATs). They supply shared properties for the FORMs in that LIST. This ability to elevate some property settings to shared status for a list of forms is useful for both indirection and compaction. For example, a list of images with the same size and colors can share one "size" property and one "color map" property. Individual FORMs can override the shared settings.

The contents of a PROP is like that of a FORM with no data chunks:

```
        PROP    ::= "PROP" #{ FormType Property* }
```

This means, "Here are the shared properties for FORM type <<FormType>."

A LIST may have at most one PROP of a FORM type, and all the PROPs must appear before any of the FORMs or nested LISTs and CATs. You can have subsequences of FORMs sharing properties by making each subsequence a LIST.

Scoping: Think of property settings as variable bindings in nested blocks of a programming language. Where in C you could write:

```
        TEXT_FONT text_font = Courier;  /* program's global default */

        File(); {
            TEXT_FONT text_font = TimesRoman;   /* shared setting */

                {
                TEXT_FONT text_font = Helvetica;  /* local setting */
                Print("Hello ");   /* uses font Helvetica */
                }

                {
                Print("there.");   /* uses font TimesRoman */
                }
            }
```

an IFF file could contain:

```
    LIST {
        PROP TEXT {
            FONT {TimesRoman}        /* shared setting */
            }

        FORM TEXT {
            FONT {Helvetica}         /* local setting */
            CHRS {Hello }            /* uses font Helvetica */
            }

        FORM TEXT {
            CHRS {there.}    /* uses font TimesRoman */
            }
    }
```

The shared property assignments selectively override the reader's global defaults, but only for FORMs within the group. A FORM's own property assignments selectively override the global and group-supplied values. So when reading an IFF file, keep property settings on a stack. They are designed to be small enough to hold in main memory.

Shared properties are semantically equivalent to copying those properties into each of the nested FORMs right after their FORM type IDs.

Properties for LIST

Optional "properties for LIST" store the origin of the list's contents in a PROP chunk for the fake FORM type "LIST." They are the properties originating program "OPGM", processor family "OCPU", computer type "OCMP", computer serial number or network address "OSN ", and user name "UNAM." In our imperfect world, these could be called upon to distinguish between unintended variations of a data format or to work around bugs in particular originating/receiving program pairs. Issue: Specify the format of these properties.

A creation date could also be stored in a property, but file creating, editing, and transporting programs should maintain the correct date in the local file system. Programs that move files between machine types are expected to copy across the creation dates.

STANDARD FILE STRUCTURE

File Structure Overview

An IFF file is just a single chunk of type FORM, LIST, or CAT. Therefore, an IFF file can be recognized by its first four bytes: "FORM", "LIST", or "CAT ". Any file contents after the chunk's end are to be ignored.

Because an IFF file can be a group of objects, programs that read/write single objects can communicate to an extent with programs that read/write groups. You are encouraged to write programs that handle all the objects in a LIST or CAT. A graphics editor, for example, could process a list of pictures as a multiple page document, one page at a time.

Programs should enforce IFF's syntactic rules when reading and writing files. This ensures robust data transfer. The public domain IFF reader/writer subroutine package does this for you. A utility program "IFFCheck" is available that scans an IFF file and checks it for conformance to IFF's syntactic rules. IFFCheck also prints an outline of the chunks in the file, showing the ckID and ckSize of each. This is quite handy when building IFF programs. Example programs are also available to show details of reading and writing IFF files.

A merge program "IFFJoin" will be available that logically appends IFF files into a single CAT group. It "unwraps" each input file that is a CAT so that the combined file is not made up of nested CATs.

If we need to revise the IFF standard, the three anchoring IDs will be used as "version numbers." That is why IDs "FOR1" through "FOR9", "LIS1" through "LIS9", and "CAT1" through "CAT9" are reserved.

IFF formats are designed for reasonable performance with floppy disks. We achieve considerable simplicity in the formats and programs by relying on the host file system rather than defining universal grouping structures such as directories for LIST contents. On huge storage systems, IFF files could be leaf nodes in a file structure like a B-tree. Let's hope the host file system implements that for us!

There are two kinds of IFF files: single-purpose files and scrap files. They differ in the interpretation of multiple data objects and in the file's external type.

Single-purpose Files

A single purpose IFF file is for normal "document" and "archive" storage. This is in contrast with "scrap files" (see below) and temporary backing storage (non-interchange files). The external file type (or filename extension, depending on the host file system) indicates the file's contents. It is generally the FORM type of the data contained; hence the restrictions on FORM type IDs.

Programmers and users may pick an "intended use" type as the filename extension to make it easy to filter for the relevant files in a filename requester. This is actually a "subclass" or "subtype" that conveniently separates files of the same FORM type that have different uses. Programs cannot demand conformity to its expected subtypes without overly restricting data interchange because they cannot know about the subtypes to be used by future programs with which users will want to exchange data.

Issue: How to generate three-letter MS-DOS extensions from four-letter FORM type IDs?

Most single-purpose files will be a single FORM (perhaps a composite FORM, such as a musical score, containing nested FORMs, such as musical instrument descriptions). If the file is a LIST or a CAT, programs should skip over unrecognized objects to read the recognized ones or the first recognized one. Then a program that can read a single-purpose file can read something out of a scrap file too.

Scrap Files

A scrap file is for maximum interconnectivity in getting data between programs--the core of a clipboard function. Scrap files may have type "IFF " or filename extension ".IFF". A scrap file is typically a CAT containing alternate representations of the same basic information. Include as many alternatives as you can readily generate. This redundancy improves interconnectivity in situations that do not allow all programs to read and write super-general formats. [Inside Macintosh chapter "Scrap Manager"] For example, a graphically-annotated musical score might be supplemented by a stripped-down four-voice melody and by a text (the lyrics).

The originating program should write the alternate representations in order of "preference": most preferred (most comprehensive) type to least preferred (least comprehensive) type. A receiving program should either use the first appearing type that it understands or search for its own "preferred" type.

A scrap file should have at most one alternative of any type. (A LIST of same-type objects is acceptable as one of the alternatives.) But do not count on this when reading; ignore extra sections of a type. Then a program that reads scrap files can read something out of single-purpose files.

Rules for Reader Programs

Here are some notes on building programs that read IFF files. If you use the standard IFF reader module "IFFR.C", many of these rules and details will be automatically handled. (See "Support Software" below) We recommend that you start from the example program "ShowILBM.C". You should also read up on recursive descent parsers. [See, for example, Compiler Construction, An Advanced Course.]

o The standard is flexible so that many programs can exchange data. This implies that a program has to scan the file and react to what is actually there in whatever order it appears. An IFF reader program is a parser.

o For interchange to really work, programs must be willing to do some conversion during read-in. If the data is not exactly what you expect--say, the raster is smaller than those created by your program--then adjust it. Similarly, your program could crop a large picture, add or drop bitplanes, and create/discard a mask plane. The program should give up gracefully on data that it cannot convert.

o If it does not start with "FORM", "LIST", or "CAT ", it is not an IFF-85 file.

o For any chunk you encounter, you must recognize its type ID to understand its contents.

o For any FORM chunk you encounter, you must recognize its FORM type ID to understand the contained "local chunks." Even if you do not recognize the FORM type, you can still scan it for nested FORMs, LISTs, and CATs of interest.

o Do not forget to skip the pad byte after every odd-length chunk.

o Chunk types LIST, FORM, PROP, and CAT are generic groups. They always contain a subtype ID followed by chunks.

o Readers ought to handle a CAT of FORMs in a file. You may treat the FORMs like document pages to sequence through or just use the first FORM.

o Simpler IFF readers completely skip LISTs. "Fully IFF-conforming" readers are those that handle LISTs, even if just to read the first FORM from a file. If you do look into a LIST, you must process shared properties (in PROP chunks) properly. The idea is to get the correct data or none at all.

o The nicest readers are willing to look into unrecognized FORMs for nested FORM types that they do recognize. For example, a musical score may contain nested instrument descriptions, and an animation file may contain still pictures.

Note to programmers: Processing PROP chunks is not simple! You will need some background in interpreters with stack frames. If this is foreign to you, build programs that read/write only one FORM per file. For the more intrepid programmers, the next paragraph summarizes how to process LISTs and PROPs. See the general IFF reader module "IFFR.C" and the example program "ShowILBM.C" for details.

Allocate a stack frame for every LIST and FORM you encounter and initialize it by copying the stack frame of the parent LIST or FORM. At the top level, you will need a stack frame initialized to your program's global defaults. While reading each LIST or FORM, store all encountered properties into the current stack frame. In the example, ShowILBM, each stack frame has a place for a bitmap header property ILBM.BMHD and a color map property ILBM.CMAP. When you finally get to the ILBM's BODY chunk, use the property settings accumulated in the current stack frame.

An alternate implementation would just remember PROPs encountered, forgetting each on reaching the end of its scope (the end of the containing LIST). When a FORM XXXX is encountered, scan the chunks in all remembered PROPs XXXX, in order, as if they appeared before the chunks actually in the FORM XXXX. This gets trickier if you read FORMs inside of FORMs.

Rules for Writer Programs

Here are some notes on building programs that write IFF files, which is much easier than reading them. If you use the standard IFF writer module "IFFW.C" (see "Support Software") many of these rules and details will automatically be enforced. See the example program "Raw2ILBM.C".

o An IFF file is a single FORM, LIST, or CAT chunk.

o Any IFF-85 file must start with the four characters "FORM", "LIST", or "CAT ", followed by a LONG ckSize. There should be no data after the chunk end.

o Chunk types LIST, FORM, PROP, and CAT are generic. They always contain a subtype ID followed by chunks. These three IDs are universally reserved, as are "LIS1" through "LIS9", "FOR1" through "FOR9", "CAT1" through "CAT9", and "   ".

o Do not forget to write a 0 pad byte after each odd-length chunk.

o Four techniques for writing an IFF group: (1) build the data in a file mapped into virtual memory, (2) build the data in memory blocks and use block I/O, (3) stream write the data piecemeal and (do not forget!) random access back to set the group length count, or (4) make a preliminary pass to compute the length count; then stream-write the data.

o Do not try to edit a file that you do not know how to create. Programs may look into a file and copy out nested FORMs of types that they recognize, but do not edit and replace the nested FORMs and do not add or remove them. That could make the containing structure inconsistent. You may write a new file containing items you copied (or copied and modified) from another IFF file, but do not copy structural parts you do not understand.

o You must adhere to the syntax descriptions given at the end of this appendix. For example, PROPs may only appear inside LISTs.

REFERENCE

Type Definitions

The following C typedefs describe standard IFF structures. Declarations to use in practice will vary with the CPU and compiler. For example, 68000 Lattice C produces efficient comparison code if we define ID as a "LONG." A macro "MakeID" builds these IDs at compile time.

```
        /* Standard IFF types, expressed in 68000 Lattice C. */

        typedef unsigned char UBYTE;    /*  8 bits unsigned */
        typedef short WORD;     /* 16 bits signed */
        typedef unsigned short UWORD;   /* 16 bits unsigned */
        typedef long LONG;      /* 32 bits signed */

        typedef char ID[4];     /* 4 chars in ' ' through '~' */

        typedef struct {
                ID      ckID;
                LONG    ckSize; /* sizeof(ckData) */
                UBYTE   ckData[/* ckSize */];
                } Chunk;

        /* ID typedef and builder for 68000 Lattice C. */
        typedef LONG ID;        /* 4 chars in ' ' through '~' */
        #define MakeID(a,b,c,d)  ( (a)<<<<24 | (b)<<<<16 | (c)<<<<8 | (d) )

        /* Globally reserved IDs. */
        #define ID_FORM   MakeID('F','O','R','M')
        #define ID_LIST   MakeID('L','I','S','T')
        #define ID_PROP   MakeID('P','R','O','P')
        #define ID_CAT    MakeID('C','A','T',' ')
        #define ID_FILLER MakeID(' ',' ',' ',' ')
```

Syntax Definitions

Here is a collection of the syntax definitions in this document.

```
   Chunk        ::= ID #{ UBYTE* } [0]
   Property     ::= Chunk

   FORM         ::= "FORM" #{ FormType (LocalChunk | FORM | LIST | CAT)* }
   FormType     ::= ID
   LocalChunk   ::= Property | Chunk

   CAT          ::= "CAT " #{ ContentsType (FORM | LIST | CAT)* }
   ContentsType ::= ID  -- a hint or an "abstract data type" ID

   LIST         ::= "LIST" #{ ContentsType PROP* (FORM | LIST | CAT)* }
   PROP         ::= "PROP" #{ FormType Property* }
```

In this extended regular expression notation, the token "#" represents a ckSize LONG count of the following {braced} data bytes. Literal items are shown in "quotes," [square bracketed items] are optional, and "*" means 0 or more instances. A sometimes-needed pad byte is shown as "[0]".

Defined Chunk IDs

This is a table of currently defined chunk IDs. We will also borrow some Macintosh IDs and data formats.

```
   Group chunk IDs
        FORM, LIST, PROP, CAT.
   Future revision group chunk IDs
        FOR1 I FOR9, LIS1 I LIS9, CAT1 I CAT9.
   FORM type IDs
        (The above group chunk IDs may not be used for FORM type IDs.)
        (Lower case letters and punctuation marks are forbidden in FORM
         type IDs.)
   Data chunk IDs
        "JJJJ", TEXT, PICT.
   PROP LIST property IDs
        OPGM, OCPU, OCMP, OSN, UNAM.
```

The special IDs not specifically outlined above are each defined as follows:

8SVX	8-bit sampled sound voice
ANBM	animated bitmap
FNTR	raster font
FNTV	vector font
FTXT	formatted text
GSCR	general-use musical score
ILBM	interleaved raster bitmap image
PDEF	Deluxe Print page definition
PICS	Macintosh picture
PLBM	(obsolete)
USCR	Uhuru Sound Software musical score
UVOX	Uhuru Sound Software Macintosh voice
SMUS	simple musical score
VDEO	Deluxe Video Construction Set video

Support Software

The following public-domain C source programs are available for use in building IFF-compatible programs. They can be found at the end of this appendix.

IFF.H, IFFR.C, IFFW.C
  IFF reader and writer package. These modules handle many of the details of reliably reading and writing IFF files.

IFFCheck.C
  This handy utility program scans an IFF file, checks that the contents are well formed, and prints an outline of the chunks.

PACKER.H, Packer.C, UnPacker.C
  Run encoder and decoder used for ILBM files.

ILBM.H, ILBMR.C, ILBMW.C
  Reader and writer support routines for raster image FORM ILBM. ILBMR calls IFFR and UnPacker. ILBMW calls IFFW and Packer.

ShowILBM.C
        Example caller of IFFR and ILBMR modules. This
        Commodore-Amiga program reads and displays a FORM ILBM.

Raw2ILBM.C
        Example ILBM writer program. As a demonstration, it
        reads a raw raster image file and writes the image
        as a FORM ILBM file.

ILBM2Raw.C
        Example ILBM reader program. Reads a FORM ILBM file
        and writes it into a raw raster image.

REMALLOC.H, Remalloc.c
        Memory allocation routines used in these examples.

INTUALL.H
        Generic "include almost everything" include-file
        with the sequence of includes correctly specified.

READPICT.H, ReadPict.c
        Given an ILBM file, read it into a bitmap and a color map.

PUTPICT.H, PutPict.c
        Given a bitmap and a color map, save it as
        an ILBM file.

GIO.H, Gio.c
        Generic I/O speedup package. Attempts to speed
        disk I/O by buffering writes and reads.

giocall.c
        Sample call to gio.

ilbmdump.c
        Reads in ILBM file, prints out ascii representation
        for including in C files.

bmprintc.c
        Prints out a C-language representation of data for a bitmap.

Example Diagrams

Here is a box diagram for an example IFF file, a raster image FORM ILBM. This FORM contains a bitmap header property chunk BMHD, a color map property chunk CMAP, and a raster data chunk BODY. This particular raster is 320 x 200 pixels x 3 bit planes uncompressed. The "0" after the CMAP chunk represents a zero pad byte; included because the CMAP chunk has an odd length. The text to the right of the diagram shows the outline that would be printed by the IFFCheck utility program for this particular file.

```
+-----------------------------------+
|'FORM'          24070              |  FORM 24070 IBLM
+-----------------------------------+
|'ILBM'                             |
+-----------------------------------+
| +-------------------------------+ |
| | 'BMHD'       20               | |  .BMHD  20
| | 320, 200, 0, 0, 3, 0, 0, ...  | |
| + ------------------------------+ |
| | 'CMAP'       21               | |  .CMAP  21
| | 0, 0, 0; 32, 0, 0; 64,0,0; .. | |
| +-------------------------------+ |
| 0                                 |
+-----------------------------------+
|'BODY'          24000              |  .BODY 24000
|0, 0, 0, ...                       |
+-----------------------------------+
```

This second diagram shows a LIST of two FORMs ILBM sharing a common BMHD property and a common CMAP property. Again, the text on the right is an outline a la IFFCheck.

```
+-----------------------------------------+
|'LIST'           48114                   |  LIST  48114  AAAA
+-----------------------------------------+
|'AAAA'                                   |  .PROP  62  ILBM
|  +-----------------------------------+  |
|  |'PROP'         62                  |  |
|  +-----------------------------------+  |
|  |'ILBM'                             |  |
|  +-----------------------------------+  |
|  | +-------------------------------+ |  |
|  | | 'BMHD'      20                | |  |  ..BMHD  20
|  | | 320, 200, 0, 0, 3, 0, 0, ...  | |  |
|  | | ------------------------------+ |  |
|  | | 'CMAP'      21                | |  |  ..CMAP  21
|  | | 0, 0, 0; 32, 0, 0; 64,0,0; .. | |  |
|  | +-------------------------------+ |  |
|  | 0                                 |  |
|  +-----------------------------------+  |
|  +-----------------------------------+  |
|  |'FORM'         24012               |  |  .FORM  24012  ILBM
|  +-----------------------------------+  |
|  |'ILBM'                             |  |
|  +-----------------------------------+  |
|  |  +-----------------------------+  |  |
|  |  |'BODY'              24000    |  |  |  ..BODY  24000
|  |  |0, 0, 0, ...                 |  |  |
|  |  +-----------------------------+  |  |
|  +-----------------------------------+  |
|  +-----------------------------------+  |
|  |'FORM'         24012               |  |  .FORM  24012  ILBM
|  +-----------------------------------+  |
|  |'ILBM'                             |  |
|  +-----------------------------------+  |
|  |  +-----------------------------+  |  |
|  |  |'BODY'              24000    |  |  |  ..BODY  24000
|  |  |0, 0, 0, ...                 |  |  |
|  |  +-----------------------------+  |  |
|  +-----------------------------------+  |
+-----------------------------------------+
```

STANDARDS COMMITTEE

The following people contributed to the design of this IFF standard:

Bob "Kodiak" Burns, Commodore-Amiga
R. J. Mical, Commodore-Amiga
Jerry Morrison, Electronic Arts
Greg Riker, Electronic Arts
Steve Shaw, Electronic Arts
Barry Walsh, Commodore-Amiga

"ILBM" IFF Interleaved Bitmap

```
Date:   January 17, 1986
From:   Jerry Morrison, Electronic Arts
Status: Released and in use
```

INTRODUCTION

"EA IFF 85" is Electronic Arts' standard for interchange format files. "ILBM" is a format for a two-dimensional raster graphics image, specifically an InterLeaved bitplane BitMap image with color map. An ILBM is an IFF "data section" or "FORM type", which can be an IFF file or a part of one. (See the IFF reference.)

An ILBM is an archival representation designed for three uses. First, it is a stand-alone image that specifies exactly how to display itself (resolution, size, color map, etc.). Second, an image intended to be merged into a bigger picture that has its own depth, color map, and so on. And third, an empty image with a color map selection or palette for a paint program. ILBM is also intended as a building block for composite IFF FORMs such as animation sequence and structured graphics. Some uses of ILBM will be to preserve as much information as possible across disparate environments. Other uses will be to store data for a single program or highly cooperative programs while maintaining subtle details. So this one format needs to accomplish a great deal.

This memo is the IFF supplement for FORM ILBM. The first two sections define the purpose and format of "standard" chunks: property chunks bitmap header "BMHD", color map "CMAP", hotspot "GRAB", destination merge data "DEST", sprite information "SPRT", Commodore-Amiga viewport mode "CAMG", and standard data chunk "BODY". The next section defines the nonstandard color range data chunk "CRNG". Additional specialized chunks like texture pattern can be added later. Finally, the ILBM syntax is summarized both as a regular expression and as a box diagram, and the optional run encoding scheme is explained.

Details of the raster layout are given in the section entitled "Standard Data Chunk". Some elements are based on the Commodore-Amiga hardware but generalized for use on other computers. An alternative to ILBM would be appropriate for computers with true color data in each pixel.

REFERENCE AND TRADEMARKS

"EA IFF 85" Standard for Interchange Format Files describes the underlying conventions for all IFF files.

Amiga[tm] is a trademark of Commodore-Amiga, Inc.
Electronic Arts[tm] is a trademark of Electronic Arts.
Macintosh[tm] is a trademark licensed to Apple Computer, Inc.
MacPaint[tm] is a trademark of Apple Computer, Inc.

STANDARD PROPERTIES

The required property "BMHD" and any optional properties must appear before any "BODY" chunk. (Because an ILBM has only one BODY chunk, any following properties are superfluous.) Any of these properties may be shared over a LIST of FORMs IBLM by putting them in a PROP ILBM. (See the "EA IFF 85" memo.)

BMHD

The required property "BMHD" holds a BitMapHeader as defined in these C declarations and the following documentation. It describes the dimensions and encoding of the image, including data necessary to understand the BODY chunk to follow.

```
typedef UBYTE Masking;  /* choice of masking technique. */

#define mskNone                 0
#define mskHasMask              1
#define mskHasTransparentColor  2
#define mskLasso                3

typedef UBYTE Compression;
    /* choice of compression algorithm applied to the rows of all
     * source and mask planes. "cmpByteRun1" is the byte run encoding
     * described later in this appendix. Do not compress across rows! */
#define cmpNone 0
#define cmpByteRun1     1

typedef struct {
        UWORD w, h;     /* raster width & height in pixels */
        WORD  x, y;     /* pixel position for this image */
        UBYTE nPlanes;  /* # source bitplanes */
        Masking masking;
        Compression compression;
        UBYTE pad1;     /* unused; for consistency, put 0 here */
        UWORD transparentColor; /* transparent "color number" (sort of) */
        UBYTE xAspect, yAspect; /* pixel aspect, a ratio width : height */
        WORD  pageWidth, pageHeight;    /* source "page" size in pixels */
        } BitMapHeader;
```

Fields are filed in the order shown. The UBYTE fields are byte-packed.

The fields w and h indicate the size of the image rectangle in pixels. Each row of the image is stored in an integral number of 16 bit words. The number of words per row is Ceiling(w/16). The fields x and y indicate the desired position of this image within the destination picture. Some reader programs may ignore x and y. A safe default for writing an ILBM is (x, y) = (0, 0).

The number of source bitplanes in the BODY chunk (see below) is stored in nPlanes. An ILBM with a CMAP but no BODY and nPlanes = 0 is the recommended way to store a color map.

Note: Color numbers are color map index values formed by pixels in the destination bitmap, which may be deeper than nPlanes if a DEST chunk calls for merging the image into a deeper image.

The field masking indicates what kind of masking is to be used for this image. The value mskNone designates an opaque rectangular image. The value mskHasMask means that a mask plane is interleaved with the bitplanes in the BODY chunk (see below). The value mskHasTransparentColor indicates that pixels in the source planes matching transparentColor are to be considered "transparent." (Actually, transparentColor is not a "color number" because it is matched with numbers formed by the source bitmap rather than the possibly deeper destination bitmap. Note that having a transparent color implies ignoring one of the color registers. (See "CMAP," below.) The value mskLasso indicates the reader may construct a mask by lassoing the image as in MacPaint*. To do this, put a 1-pixel border of transparentColor around the image rectangle. Then do a seed fill from this border. Filled pixels are to be transparent.

Issue: Include an algorithm for converting a transparent color to a mask plane and maybe a lasso algorithm.

A code indicating the kind of data compression used is stored in compression. Beware that using data compression makes your data unreadable by programs that do not implement the matching decompression algorithm. So we will employ as few compression encodings as possible. The run encoding byteRun1 is documented later in this appendix.

The field pad1 is a pad byte and must be set to 0 for consistency. This field could get used in the future.

The transparentColor specifies which bit pattern means "transparent." This applies only if masking is mskHasTransparentColor or mskLasso (see above). Otherwise, transparentColor should be 0.

The pixel aspect ratio is stored as a ratio in the two fields xAspect and yAspect. This may be used by programs to compensate for different aspects or to help interpret the fields w, h, x, y, pageWidth, and pageHeight, which are in units of pixels. The fraction xAspect/yAspect represents a pixel's width/height. It is recommended that your programs store proper fractions in BitMapHeaders, but aspect ratios can always be correctly compared with the the test

```
        xAspect%yDesiredAspect = yAspect%xDesiredAspect
```

Typical values for aspect ratio are width : height = 10 : 11 (Amiga 320 x 200 display) and 1 : 1 (Macintosh*).

The size in pixels of the source "page" (any raster device) is stored in pageWidth and pageHeight--for example, (320, 200) for a low-resolution Amiga display. This information might be used to scale an image or to automatically set the display format to suit the image. (The image can be larger than the page.)

CMAP

The optional (but encouraged) property "CMAP" stores color map data as triplets of red, green, and blue intensity values. The n color map entries ("color registers") are stored in the order 0 through n-1, totaling 3n bytes. Thus n is the ckSize/3. Normally, n would equal 2nPlanes.

A CMAP chunk contains a ColorMap array as defined below. (These typedefs assume a C compiler that implements packed arrays of 3-byte elements.)

```
typedef struct {
        UBYTE red, green, blue; /* color intensities 0..255 */
        } ColorRegister;        /* size = 3 bytes */

typedef ColorRegister ColorMap[n];      /* size = 3n bytes */
```

The color components red, green, and blue represent fractional intensity values in the range 0 through 255/256ths. White is (255, 255, 255) and black is (0, 0, 0). If your machine has less color resolution, use the high-order bits. Shift each field right on reading (or left on writing) and assign it to (from) a field in a local packed format such as Color4, below. This achieves automatic conversion of images across environments with different color resolutions. On reading an ILBM, use defaults if the color map is absent or has fewer color registers than you need. Ignore any extra color registers.

The example type Color4 represents the format of a color register in working memory of an Amiga computer, which has 4-bit video DACs. (The ":4" tells the C compiler to pack the field into 4 bits.)

```
typedef struct {
        unsigned pad1 :4, red :4, green :4, blue :4;
        } Color4;       /* Amiga RAM format. Not filed. */
```

Remember that every chunk must be padded to an even length, so a color map with an odd number of entries would be followed by a 0 byte, not included in the ckSize.

GRAB

The optional property "GRAB" locates a "handle" or "hotspot" of the image relative to its upper left corner--for example, when used as a mouse cursor or a "paint brush." A GRAB chunk contains a Point2D.

```
typedef struct {
        WORD x, y;      /* relative coordinates (pixels) */
        } Point2D;
```

DEST

The optional property "DEST" is a way to say how to scatter zero or more source bitplanes into a deeper destination image. Some readers may ignore DEST.

The contents of a DEST chunk is DestMerge structure:

```
typedef struct {
    UBYTE depth;        /* # bitplanes in the original source */
    UBYTE pad1; /* unused; for consistency put 0 here */
    UWORD planePick; /* how to scatter source bitplanes into destination */
    UWORD planeOnOff;   /* default bitplane data for planePick */
    UWORD planeMask;    /* selects which bitplanes to store into */
    } DestMerge;
```

The low-order depth number of bits in planePick, planeOnOff, and planeMask correspond one-to-one with destination bitplanes. Bit 0 with bitplane 0, etc. (Any higher order bits should be ignored.) "1" bits in planePick mean "put the next source bitplane into this bitplane," so the number of "1" bits should equal nPlanes. "0" bits mean "put the corresponding bit from planeOnOff into this bitplane". Bits in planeMask gate writing to the destination bitplane: "1" bits mean "write to this bitplane", while "0" bits mean "leave this bitplane alone". The normal case (with no DEST property) is equivalent to

planePick = planeMask = 2nPlanesJ- 1.

Remember that color numbers are formed by pixels in the destination bitmap (depth planes deep) not in the source bitmap (nPlanes planes deep).

SPRT

The presence of an "SPRT" chunk indicates that this image is intended as a sprite. It is up to the reader program to actually make it a sprite, if possible, and to use or overrule the sprite precedence data inside the SPRT chunk:

```
        typedef UWORD SpritePrecedence;
                        /* relative precedence, 0 is the highest */
```

Precedence 0 is the highest, denoting a sprite that is foremost.

Creating a sprite may imply other set-up. For example, a two-plane Amiga sprite would have transparentColor = 0. Color registers 1, 2, and 3 in the CMAP would be stored into the correct hardware color registers for the hardware sprite number used, and CMAP color register 0 would be ignored.

CAMG

A "CAMG" chunk is specifically for the Commodore Amiga computer. It stores a LONG "viewport mode." This lets you specify Amiga display modes such as "dual playfield" and "hold and modify."

STANDARD DATA CHUNK

Raster Layout

Raster scan proceeds left to right (increasing X) across scan lines, then top to bottom (increasing Y) down columns of scan lines. The coordinate system is in units of pixels, where (0,0) is the upper left corner. The raster is typically organized as bitplanes in memory. The corresponding bits from each plane, taken together, make up an index into the color map that gives a color value for that pixel. The first bitplane, plane 0, is the low-order bit of these color indexes.

A scan line is made of one "row" from each bitplane. A row is one plane's bits for one scan line, but padded out to a word (2-byte) boundary (not necessarily the first word boundary). Within each row, successive bytes are displayed in order, and the most significant bit of each byte is displayed first.

A "mask" is an optional "plane" of data the same size (w, h) as a bitplane. It tells how to "cut out" part of the image when painting it onto another image. A 1 bit in the mask means "copy the corresponding pixel to the destination," and a 0 mask bit means "leave this destination pixel alone." In other words, a 0 bit designate transparent pixels.

The rows of the different bitplanes and mask are interleaved in the file (see below). This localizes all the information pertinent to each scan line. It makes it much easier to transform the data while reading it to adjust the image size or depth. It also makes it possible to scroll a big image by swapping rows directly from the file without random-accessing to all the bitplanes.

BODY

The source raster is stored in a "BODY" chunk. This one chunk holds all bitplanes and the optional mask, interleaved by row.

The BitMapHeader, in a BMHD property chunk, specifies the raster's dimensions w, h, and nPlanes. It also holds the masking field that indicates if there is a mask plane and the compression field that indicates the compression algorithm used. This information is needed to interpret the BODY chunk, so the BMHD chunk must appear first. While reading an ILBM's BODY, a program may convert the image to another size by filling (with transparentColor) or clipping.

The BODY's content is a concatenation of scan lines. Each scan line is a concatenation of one row of data from each plane in order 0 through nPlanes-1 followed by one row from the mask (if masking = hasMask). If the BitMapHeader field compression is cmpNone, all h rows are exactly Ceiling(w/16) words wide. Otherwise, every row is compressed according to the specified algorithm and their stored widths depend on the data compression.

Reader programs that require fewer bitplanes than appear in a particular ILBM file can combine planes or drop the high-order (later) planes. Similarly, they may add bitplanes and/or discard the mask plane.

Do not compress across rows and do not forget to compress the mask just as you do the bitplanes. Remember to pad any BODY chunk that contains an odd number of bytes.

NONSTANDARD DATA CHUNK

The following data chunk was defined after various programs began using FORM ILBM, so it is a "nonstandard" chunk. That means there is some slight chance of name collisions.

CRNG

A "CRNG" chunk contains "color register range" information. It is used by Electronic Arts' Deluxe Paint program to identify a contiguous range of color registers for a "shade range" and color cycling. There can be zero or more CRNG chunks in an ILBM, but all should appear before the BODY chunk. Deluxe Paint normally writes 4 CRNG chunks in an ILBM when the user asks it to "Save Picture".

```
typedef struct {
     WORD  pad1;      /* reserved for future use; store 0 here */
     WORD  rate;      /* color cycle rate */
     WORD  active;    /* nonzero means cycle the colors */
     UBYTE low, high; /* lower and upper color registers selected */
     } CRange;
```

The fields low and high indicate the range of color registers (color numbers) selected by this CRange.

The field active indicates whether color cycling is on or off. Zero means off.

The field rate determines the speed at which the colors will step when color cycling is on. The units are such that a rate of 60 steps per second is represented as 214 = 16384. Slower rates can be obtained by linear scaling: for 30 steps/second, rate = 8192; for 1 step/second, rate = 16384/60 E 273.

CCRT

Commodore's Graphicraft program uses a similar chunk "CCRT" (for Color Cycling Range and Timing). This chunk contains a CycleInfo structure.

```
typedef struct {
       WORD  direction; /* 0 = don't cycle. 1 = cycle forwards (1, 2, 3).
                         * -1 = cycle backwards (3, 2, 1) */
       UBYTE start, end;  /* lower and upper color registers selected */
       LONG  seconds;     /* # seconds between changing colors */
       LONG  microseconds;   /* # microseconds between changing colors */
       WORD  pad;         /* reserved for future use; store 0 here  */
       } CycleInfo;
```

This is pretty similar to a CRNG chunk. A program would probably only use one of these two methods of expressing color cycle data. You could write out both if you want to communicate this information to both Deluxe Paint and Graphicraft.

A CCRT chunk expresses the color cycling rate as a number of seconds plus a number of microseconds.

ILBM Regular Expression

Here's a regular expression summary of the FORM ILBM syntax. This could be an IFF file or a part of one.

```
ILBM ::= "FORM" #{ "ILBM" BMHD [CMAP] [GRAB] [DEST] [SPRT] [CAMG]
                   CRNG* CCRT* [BODY]    }
BMHD ::= "BMHD" #{ BitMapHeader          }
CMAP ::= "CMAP" #{ (red green blue)*     } [0]
GRAB ::= "GRAB" #{ Point2D               }
DEST ::= "DEST" #{ DestMerge             }
SPRT ::= "SPRT" #{ SpritePrecendence     }
CAMG ::= "CAMG" #{ LONG                  }
CRNG ::= "CRNG" #{ CRange                }
CCRT ::= "CCRT" #{ CycleInfo             }
BODY ::= "BODY" #{ UBYTE*                } [0]
```

The token "#" represents a ckSize LONG count of the following {braced} data bytes. For example, a BMHD's "#" should equal sizeof(BitMapHeader). Literal strings are shown in "quotes," [square bracket items] are optional, and "*" means 0 or more repetitions. A sometimes-needed pad byte is shown as "[0]".

The property chunks (BMHD, CMAP, GRAB, DEST, SPRT, and CAMG) and any CRNG and CCRT data chunks may actually be in any order but all must appear before the BODY chunk, because ILBM readers usually stop as soon as they read the BODY. If any of the six property chunks are missing, default values are "inherited" from any shared properties (if the ILBM appears inside an IFF LIST with PROPs) or from the reader program's defaults. If any property appears more than once, the last occurrence before the BODY is the one that counts, because that is the one that modifies the BODY.

ILBM Box Diagram

Here is a box diagram for a simple example: an uncompressed image 320 x 200 pixels x 3 bitplanes. The text to the right of the diagram shows the outline that would be printed by the IFFCheck utility program for this particular file.

```
+-----------------------------------+
|'FORM'          24070              |   FORM 24070 IBLM
+-----------------------------------+
|'ILBM'                             |
+-----------------------------------+
| +-------------------------------+ |
| | 'BMHD'       20               | |   .BMHD  20
| | 320, 200, 0, 0, 3, 0, 0, ...  | |
| + ------------------------------+ |
| | 'CMAP'       21               | |   .CMAP  21
| | 0, 0, 0; 32, 0, 0; 64,0,0; .. | |
| +-------------------------------+ |
| 0                                 |
+-----------------------------------+
|'BODY'          24000              |   .BODY 24000
|0, 0, 0, ...                       |
+-----------------------------------+
```

The "0" after the CMAP chunk is a pad byte.

ByteRun1 Run Encoding

The run-encoding scheme byteRun1 is best described by psuedo-code
for the decoder Unpacker (called UnPackBits in the Macintosh* toolbox):

```
        UnPacker:
            LOOP until produced the desired number of bytes
                Read the next source byte into n
                SELECT n FROM
                    [0..127]     => copy the next n+1 bytes literally
                    [-1..-127]   => replicate the next byte -n+1 times
                    -128         => noop
                    ENDCASE;
            ENDLOOP;
```

In the inverse routine Packer, it is best to encode a 2-byte repeat
run as a replicate run except when preceded and followed by a literal
run, in which case it is best to merge the three into one literal run.
Always encode 3-byte repeats as replicate runs.

Remember that each row of each scan line of a raster is separately packed.

STANDARDS COMMITTEE

The following people contributed to the design of this FORM ILBM standard:

Bob "Kodiak" Burns, Commodore-Amiga
R. J. Mical, Commodore-Amiga
Jerry Morrison, Electronic Arts
Greg Riker, Electronic Arts
Steve Shaw, Electronic Arts
Dan Silva, Electronic Arts
Barry Walsh, Commodore-Amiga

"FTXT" IFF Formatted Text

Date: November 15, 1985
From: Steve Shaw and Jerry Morrison, Electronic Arts and
Bob "Kodiak" Burns, Commodore-Amiga
Status: Draft 2.6

DRAFT DRAFT DRAFT DRAFT DRAFT DRAFT

INTRODUCTION

This memo is the IFF supplement for FORM FTXT. An FTXT is an IFF "data section" or "FORM type" which can be an IFF file or a part of one containing a stream of text plus optional formatting information. "EA IFF 85" is Electronic Arts' standard for interchange format files. (See the IFF reference.)

An FTXT is an archival and interchange representation designed for three uses. The simplest use is for a "console device" or "glass teletype" (the minimal 2-D text layout means): a stream of "graphic" ("printable") characters plus positioning characters "space" ("SP") and line terminator ("LF"). This is not intended for cursor movements on a screen, although it does not conflict with standard cursor-moving characters. The second use is text that has explicit formatting information (or "looks") such as font family and size, typeface, etc. The third use is as the lowest layer of a structured document that also has "inherited" styles to implicitly control character looks. For that use, FORMs FTXT would be embedded within a future document FORM type. The beauty of FTXT is that these three uses are interchangeable, that is, a program written for one purpose can read and write the others' files. Thus, a word processor does not have to write a separate plain text file to communicate with other programs.

Text is stored in one or more "CHRS" chunks inside an FTXT. Each CHRS contains a stream of 8-bit text compatible with ISO and ANSI data interchange standards. FTXT uses just the central character set from the ISO/ANSI standards. (These two standards are henceforth called "ISO/ANSI" as in "see the ISO/ANSI reference.")

Because it is possible to extract just the text portions from future document FORM types, programs can exchange data without having to save both plain text and formatted text representations.

Character looks are stored as embedded control sequences within CHRS chunks. This document specifies which class of control sequences to use: the CSI group. This document does not yet specify their meanings, for example, which one means "turn on italic face." Consult ISO/ANSI.

The second part of this discussion defines the chunk types character stream "CHRS" and font specifier "FONS". These are the "standard" chunks. Specialized chunks for private or future needs can be added later. The third outlines an FTXT reader program that strips a document down to plain unformatted text. Finally, a code table for the 8-bit ISO/ANSI character set used here, an example FTXT shown as a box diagram, and a racetrack diagram of the syntax of ISO/ANSI control sequences are provided.

REFERENCE AND TRADEMARKS

Amiga[tm] is a trademark of Commodore-Amiga, Inc.

Electronic Arts[tm] is a trademark of Electronic Arts.

IFF: "EA IFF 85" Standard for Interchange Format Files describes the underlying conventions for all IFF files.

ISO/ANSI: ISO/DIS 6429.2 and ANSI X3.64-1979. International Organization for Standardization (ISO) and American National Standards Institute (ANSI) data-interchange standards. The relevant parts of these two standards documents are identical. ISO standard 2022 is also relevant.

STANDARD DATA AND PROPERTY CHUNKS

The main contents of a FORM FTXT is in its character stream "CHRS" chunks. Formatting property chunks may also appear. The only formatting property yet defined is "FONS", a font specifier. A FORM FTXT with no CHRS represents an empty text stream. A FORM FTXT may contain nested IFF FORMs, LISTs, or CATs, although a "stripping" reader (see below) will ignore them.

Character Set

FORM FTXT uses the core of the 8-bit character set defined by the ISO/ANSI standards cited at the start of this document. (See the character code table below.) This character set is divided into two "graphic" groups plus two "control" groups. Eight of the control characters begin ISO/ANSI standard control sequences. (See "Control Sequences" below.) Most control sequences and control characters are reserved for future use and for compatibility with ISO/ANSI. Current reader programs should skip them.

- C0 is the group of control characters in the range NUL (hex 0) through hex 1F. Of these, only LF (hex 0A) and ESC (hex 1B) are significant. ESC begins a control sequence. LF is the line terminator, meaning "go to the first horizontal position of the next line." All other C0 characters are not used. In particular, CR (hex 0D) is not recognized as a line terminator.

- G0 is the group of graphic characters in the range hex 20 through hex 7F. SP (hex 20) is the space character. DEL (hex 7F) is the delete character, which is not used. The rest are the standard ASCII printable characters "!" (hex 21) through "~" (hex 7E).

- C1 is the group of extended control characters in the range hex 80 through hex 9F. Some of these begin control sequences. The control sequence starting with CSI (hex 9B) is used for FTXT formatting. All other control sequences and C1 control characters are unused.

- G1 is the group of extended graphic characters in the range NBSP (hex A0) through "ÿ" (hex FF). It is one of the alternate graphic groups proposed for ISO/ANSI standardization.

Control Sequences

Eight of the control characters begin ISO/ANSI standard "control sequences" (or "escape sequences"). These sequences are described below and diagrammed below.

```
        G0          ::= (SP through DEL)
        G1          ::= (NBSP through "X")

        ESC-Seq     ::= ESC (SP through "/")* ("0" through "~")
        ShiftToG2   ::= SS2 G0
        ShiftToG3   ::= SS3 G0
        CSI-Seq     ::= CSI (SP through "?")* ("@" through "~")
        DCS-Seq     ::= (DCS | OSC | PM | APC) (SP through "~" | G1)* ST
```

"ESC-Seq" is the control sequence ESC (hex 1B), followed by zero or more characters in the range SP through "/S (hex 20 through hex 2F), followed by a character in the range "0" through "~" (hex 30 through hex 7E). These sequences are reserved for future use and should be skipped by current FTXT reader programs.

SS2 (hex 8E) and SS3 (hex 8F) shift the single following G0 character into yet-to-be-defined graphic sets G2 and G3, respectively. These sequences should not be used until the character sets G2 and G3 are standardized. A reader may simply skip the SS2 or SS3 (taking the following character as a corresponding G0 character) or replace the two-character sequence with a character like "?" to mean "absent."

FTXT uses "CSI-Seq" control sequences to store character formatting (font selection by number, type face, and text size) and perhaps layout information (position and rotation). "CSI-Seq" control sequences start with CSI (the "control sequence introducer," hex 9B). Syntactically, the sequence includes zero or more characters in the range SP through "?" (hex 20 through hex 3F) and a concluding character in the range "@" through "~" (hex 40 through hex 7E). These sequences may be skipped by a minimal FTXT reader--that is, one that ignores formatting information.

Note: A future FTXT standardization document will explain the uses of CSI-Seq sequences for setting character face (light weight vs. medium vs. bold, italic vs. roman, height, pitch, position, and rotation). For now, consult the ISO/ANSI references.

"DCS-Seq" is the control sequences starting with DCS (hex 90), OSC (hex 9D), PM (hex 9E), or APC (hex 9F); followed by zero or more characters, each of which is in the range SP through "~" (hex 20 through hex 7E) or a G1 character; and terminated by an ST (hex 9C). These sequences are reserved for future use and should be skipped by current FTXT reader programs.

Data Chunk CHRS

A CHRS chunk contains a sequence of 8-bit characters abiding by the ISO/ANSI standards cited at the start of this document. This includes the character set and control sequences as described above and summarized later.

A FORM FTXT may contain any number of CHRS chunks. Taken together, they represent a single stream of textual information. That is, the contents of CHRS chunks are effectively concatenated except that each control sequence must be completely within a single CHRS chunk, and any formatting property chunks appearing between two CHRS chunks affects the formatting of the latter chunk's text. Any formatting settings set by control sequences inside a CHRS carry over to the next CHRS in the same FORM FTXT. All formatting properties stop at the end of the FORM, because IFF specifies that adjacent FORMs are independent of each other (although not independent of any properties inherited from an enclosing LIST or FORM).

Property Chunk FONS

The optional property "FONS" holds a FontSpecifier as defined in the C declaration below. It assigns a font to a numbered "font register" so it can be referenced by number within subsequent CHRS chunks. (This function is not provided within the ISO and ANSI standards.) The font specifier gives both a name and a description for the font so the recipient program can do font substitution.

By default, CHRS text uses font 1 until it selects another font. A minimal text reader always uses font 1. If font 1 has not been specified, the reader may use the local system font as font 1.

```
        typedef struct {
            UBYTE id;
                /* 0 through 9 is a font id number referenced by an
                 * SGR control sequence selective parameter of 10
                 * through 19. Other values are reserved for future
                 * standardization.
                 */
            UBYTE pad1;   /* reserved for future use; store 0 here */
            UBYTE proportional;
                /* proportional font? 0 = unknown, 1 = no, 2 = yes */
            UBYTE serif;
                /* serif font? 0 = unknown, 1 = no, 2 = yes */
            char name[];
                /* a NULL-terminated string naming preferred font. */
            } FontSpecifier;
```

Fields are filed in the order shown. The UBYTE fields are byte-packed (2 per 16-bit word). The field pad1 is reserved for future standardization. Programs should store 0 there for now.

The field proportional indicates if the desired font is proportional width as opposed to fixed width. The field serif indicates if the desired font is serif as opposed to sans serif.

Future Properties

New optional property chunks may be defined in the future to store additional formatting information. They will be used to represent formatting not encoded in standard ISO/ANSI control sequences and for "inherited" formatting in structured documents. Text orientation might be one example.

Positioning Units

Unless otherwise specified, position and size units used in FTXT formatting properties and control sequences are in decipoints (720 decipoints/inch). This is ANSI/ISO Positioning Unit Mode (PUM) 2. While a metric standard might be nice, decipoints allow the existing U.S.A. typographic units to be encoded easily--for example, 12 points is 120 decipoints.

FTXT STRIPPER

An FTXT reader program can read the text and ignore all formatting and structural information in a document FORM that uses FORMs FTXT for the leaf nodes. This amounts to stripping a document down to a stream of plain text. It would do this by skipping over all chunks except FTXT.CHRS (CHRS chunks found inside a FORM FTXT) and skipping all control characters and control sequences within the FTXT.CHRS chunks. (This text scanner is discussed later.) It may also read FTXT.FONS chunks to find a description for font 1.

Here's a Pascal-ish program for an FTXT stripper. Given a FORM (a document of some kind), it scans for all FTXT.CHRS chunks. This would likely be applied to the first FORM in an IFF file.

```
PROCEDURE ReadFORM4CHRS();        {Read an IFF FORM for FTXT.CHRS chunks.}
   BEGIN
   IF the FORM's subtype = "FTXT"
        THEN ReadFTXT4CHRS()
        ELSE WHILE something left to read in the FORM DO BEGIN
            read the next chunk header;
            CASE the chunk's ID OF
                "LIST", "CAT ": ReadCAT4CHRS();
                "FORM": ReadFORM4CHRS();
                OTHERWISE skip the chunk's body;
                END
            END
   END;

{Read a LIST or CAT for all FTXT.CHRS chunks.}
PROCEDURE ReadCAT4CHRS();
    BEGIN
    WHILE something left to read in the LIST or CAT DO BEGIN
        read the next chunk header;
        CASE the chunk's ID OF
           "LIST", "CAT ": ReadCAT4CHRS();
           "FORM": ReadFORM4CHRS();
           "PROP": IF we're reading a LIST AND the PROP's subtype = "FTXT"
                   THEN read the PROP for "FONS" chunks;
           OTHERWISE error--malformed IFF file;
           END
        END
    END;


PROCEDURE ReadFTXT4CHRS();        {Read a FORM FTXT for CHRS chunks.}
    BEGIN
    WHILE something left to read in the FORM FTXT DO BEGIN
        read the next chunk header;
        CASE the chunk's ID OF
            "CHRS": ReadCHRS();
            "FONS": BEGIN
                read the chunk's contents into a FontSpecifier variable;
                IF the font specifier's id = 1 THEN use this font;
                END;
            OTHERWISE skip the chunk's body;
            END
        END
    END;
```

```
{Read an FTXT.CHRS. Skip all control sequences and unused control chars.}
PROCEDURE ReadCHRS();
    BEGIN
    WHILE something left to read in the CHRS chunk DO
        CASE read the next character OF
           LF:  start a new output line;
           ESC: SkipControl([' '..'/'], ['0'..'~']);
           IN [' '..'~'], IN [NBSP..'X']: output the character;
           SS2, SS3: ;  {Just handle the following G0 character
                          directly, ignoring the shift to G2 or G3.}
           CSI: SkipControl([' '..'?'], ['@'..'~']);
           DCS, OSC, PM, APC: SkipControl([' '..'~'] + [NBSP..'X'], [ST]);
           END
    END;
```

```
{Skip a control sequence of the format (rSet)* (tSet), i.e. any number
  of characters in the set rSet followed by a character in the set tSet.}
PROCEDURE SkipControl(rSet, tSet);
   VAR c: CHAR;
   BEGIN
   REPEAT c := read the next character
        UNTIL c NOT IN rSet;
   IF c NOT IN tSet
        THEN put character c back into the input stream;
   END
```

The following program is an optimized version of the above routines ReadFORM4CHRS and ReadCAT4CHRS for situations in which you are ignoring fonts as well as formatting. It takes advantage of certain facts of the IFF format to read a document FORM and its nested FORMs, LISTs, and CATs without a stack. In other words, it is a hack that ignores all fonts and faces to cheaply get to the plain text of the document.

```
{Cheap scan of an IFF FORM for FTXT.CHRS chunks.}
PROCEDURE ScanFORM4CHRS();
   BEGIN
   IF the document FORM's subtype = "FTXT"
        THEN ReadFTXT4CHRS()
        ELSE WHILE something left to read in the FORM DO BEGIN
            read the next chunk header;
            IF it's a group chunk (LIST, FORM, PROP, or CAT)
               THEN read its subtype ID;
            CASE the chunk's ID OF
              "LIST", "CAT ":;  {NOTE: See explanation below.*}
              "FORM": IF this FORM's subtype = "FTXT" THEN ReadFTXT4CHRS()
                   ELSE;        {NOTE: See explanation below.*}
              OTHERWISE skip the chunk's body;
              END
            END
    END;
```

*Note: This implementation is subtle. After reading a group header other than FORM FTXT it just continues reading. This amounts to reading all the chunks inside that group as if they were not nested in a group.

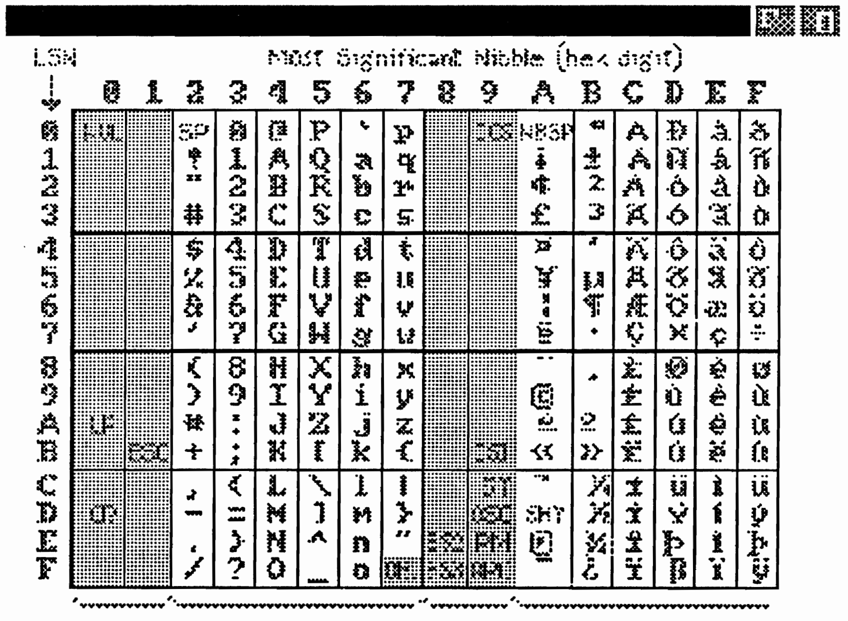
CHARACTER CODE TABLE

This table corresponds to the ISO/DIS 6429.2 and ANSI X3.64-1979 8-bit character set standards. Only the core character set of those standards is used in FTXT.

Two G1 characters are not defined in the standards and are shown as dark gray entries in this table. Light gray shading denotes control characters. (DEL is a control character although it belongs to the graphic group G0.) The following five rare G1 characters are left blank in the table below because of limitations of available fonts: hex A8, D0, DE, F0, and FE.

ISO/DIS 6429.2 and ANSI X3.64-1979 Character Code Table

+---+



LSN   Most Significant Nibble (hex digit)

LSN	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL		SP	0	@	P	`	p		DCS	NBSP	°	À	Ð	à	ð
1			!	1	A	Q	a	q			¡	±	Á	Ñ	á	ñ
2			"	2	B	R	b	r			¢	²	Â	Ò	â	ò
3			#	3	C	S	c	s			£	³	Ã	Ó	ã	ó
4			$	4	D	T	d	t			¤	´	Ä	Ô	ä	ô
5			%	5	E	U	e	u			¥	µ	Å	Õ	å	õ
6			&	6	F	V	f	v			¦	¶	Æ	Ö	æ	ö
7			'	7	G	W	g	w			§	·	Ç	×	ç	÷
8			(	8	H	X	h	x			¨	¸	È	Ø	è	ø
9			)	9	I	Y	i	y			©	¹	É	Ù	é	ù
A	LF		*	:	J	Z	j	z			ª	º	Ê	Ú	ê	ú
B		ESC	+	;	K	[	k	{		CSI	«	»	Ë	Û	ë	û
C			,	<	L	\	l	|		ST	¬	¼	Ì	Ü	ì	ü
D	CR		-	=	M	]	m	}		OSC	SHY	½	Í	Ý	í	ý
E			.	>	N	^	n	~	SS2	PM	®	¾	Î	Þ	î	þ
F			/	?	O	_	o	DEL	SS3	APC	¯	¿	Ï	ß	ï	ÿ

FTXT EXAMPLE

Here is a box diagram for a simple example: "The quick brown fox jumped Four score and seven," written in a proportional serif font named "Roman."

```
+-----------------------------------+
|'FORM'        24070                |    FORM  24070  ILBM
+-----------------------------------+
|'ILBM'                             |
+-----------------------------------+
| +-----------------------------+   |
| | 'BMHD'      20              |   |    .BMHD  20
| | 320, 200, 0, 0, 3, 0, 0, ...|   |
| | ----------------------------+   |
| | 'CMAP'      21              |   |    .CMAP  21
| | 0, 0, 0; 32, 0, 0; 64,0,0; ..|  |
| +-----------------------------+   |
| +-----------------------------+   |
| |'BODY'              24000    |   |    .BODY  24000
| |0, 0, 0, ...                 |   |
| +-----------------------------+   |
+-----------------------------------+
```

The "0" after the CMAP chunk is a pad byte.

ISO/ANSI CONTROL SEQUENCES

This is a racetrack diagram of the ISO/ANSI characters and control sequences as used in FTXT CHRS chunks.

```
        line terminator
-----+-------------------> LF --------------------------------------->
     | ESC-Seq
     +-------------------> ESC ---+>----------------+--> 0 thru ~ --->
     |                            |                 |
     |                            +-- SP thru / <---+
     | printable
     +---------------+---> SP thru ~ --+->-------------------------->
     |               |                 |
     |               +---> G1 -------->+
     | shift to G2
     +-------------------> SS2 ----> G0 ---> (produces a G2 character)
     | shift to G3
     +-------------------> SS3 ----> G0 ---> (produces a G3 character)
     | CSI-Seq
     +-------------------> CSI ---+>----------------+--> @ thru ~ --->
     |                            |                 |
     |                            +-- SP thru ? <---+
     | DCS-Seq
     +----------> DCS,OSC,PM,or APC --+>-------------+--+-> ST -+---->
     |                                |             |  |       |
     |                                +- SP thru ~ <-+  +-> G1 -+
     | discard
     +----------> any other character ------------------------------>
```

Of the various control sequences, only CSI-Seq is used for FTXT character formatting information. The others are reserved for future use and for compatibility with ISO/ANSI standards. Certain character sequences are syntactically malformed--for example, CSI followed by a C0, C1, or G1 character. Writer programs should not generate reserved or malformed sequences and reader programs should skip them.

Consult the ISO/ANSI standards for the meaning of the CSI-Seq control sequences.

The two character set shifts SS2 and SS3 may be used when the graphic character groups G2 and G3 become standardized.

STANDARDS COMMITTEE

The following people contributed to the design of this IFF standard:

Bob "Kodiak" Burns, Commodore-Amiga
R. J. Mical, Commodore-Amiga
Jerry Morrison, Electronic Arts
Greg Riker, Electronic Arts
Steve Shaw, Electronic Arts
Barry Walsh, Commodore-Amiga

"SMUS" IFF Simple Musical Score

Date: February 5, 1986
From: Jerry Morrison, Electronic Arts
Status: Adopted

INTRODUCTION

This is a reference document for the data interchange format "SMUS", which stands for Simple MUsical Score. "EA IFF 85" is Electronic Arts' standard for interchange format files. A FORM (or "data section") such as FORM SMUS can be an IFF file or a part of one. [See "EA IFF 85" Electronic Arts Interchange File Format.]

SMUS is a practical data format for uses like moving limited scores between programs and storing theme songs for game programs. The format should be geared for easy read-in and playback. So FORM SMUS uses the compact time encoding of Common Music Notation (half notes, dotted quarter rests, etc.). The SMUS format should also be structurally simple. So it has no provisions for fancy notational information needed by graphical score editors or the more general timing (overlapping notes, etc.) and continuous data (pitch bends, etc.) needed by performance-oriented MIDI recorders and sequencers.

A SMUS score can say which "instruments" are supposed to play which notes, but the score is independent of whatever output device and driver software is used to perform the notes. The score can contain device- and driver-dependent instrument data, but this is just a cache. As long as a SMUS file stays in one environment, the embedded instrument data is very convenient. When you move a SMUS file between programs or hardware configurations, the contents of this cache usually become useless.

Like all IFF formats, SMUS is a filed or "archive" format. It is completely independent of score representations in working memory, editing operations, user interface, display graphics, computation hardware, and sound hardware. Like all IFF formats, SMUS is extensible.

SMUS is not an end-all musical score format. Other formats may be more appropriate for certain uses. (We'd like to design a general-use IFF score format "GSCR". FORM GSCR would encode fancy notational data and performance data. There would be a SMUS to/from GSCR converter.)

The next two sections give important background information and detail the SMUS components by defining the required property score header "SHDR," the optional text properties name "NAME," copyright "(c)," author "AUTH," optional text annotation "ANNO," the optional instrument specifier "INS1," and the track data chunk "TRAK." The section called "Private Chunks" defines some chunks for particular programs to store private information. These are "standard" chunks; specialized chunks for future needs can be added later. Finally, there is a quick-reference summary and an example box diagram.

Update: This standard has been revised since the draft versions. The "INST" chunk type was revised to form the "INS1" chunk type. Also, several SEvent types and a few text chunk types have been added.

References

"EA IFF 85" Standard for Interchange Format Files describes the underlying conventions for all IFF files.

"8SVX" IFF 8-Bit Sampled Voice documents a data format for sampled instruments.

Electronic Arts[tm] is a trademark of Electronic Arts.

MIDI: Musical Instrument Digital Interface Specification 1.0, International MIDI Association, 1983.

MacWrite[tm] is a trademark of Apple Computer, Inc.

SSSP: See various articles on Structured Sound Synthesis Project in Foundations of Computer Music.

BACKGROUND

Here is some background information on score representation in general and design choices for SMUS.

First, we borrow some terminology from the Structured Sound Synthesis Project. (See the SSSP reference.) A "musical note" is one kind of scheduled event. Its properties include an event duration, an event delay, and a timbre object. The event duration tells the scheduler how long the note should last. The event delay tells how long after starting this note to wait before starting the next event. The timbre object selects sound driver data for the note; an "instrument" or "timbre." A "rest" is a sort of null event. Its only property is an event delay.

Classical Event Durations

SMUS is geared for "classical" scores, not free-form performances. Its event durations, therefore, are classical (whole note, dotted quarter rest, etc.). It can tie notes together to build a "note event" with an unusual event duration.

The set of useful classical durations is very small. So SMUS needs only a handful of bits to encode an event duration. This is very compact. It is also very easy to display in Common Music Notation (CMN).

Tracks

The events in a SMUS score are grouped into parallel "tracks". Each track is a linear stream of events. Tracks serve four functions:

1. Tracks make it possible to encode event delays very compactly. A "classical" score has chorded notes and sequential notes, but no overlapping notes. That is, each event begins either simultaneous with or immediately following the previous event in that track. So each event delay is either 0 or the same as the event's duration. This binary distinction requires only one bit of storage.

2. Tracks represent the "voice tracks" in Common Music Notation. CMN organizes a score in parallel staves, with one or two "voice tracks" per staff. So one or two SMUS tracks represents a CMN staff.

3. Tracks are a good match to available sound hardware. We can use "instrument settings" in a track to store the timbre assignments for that track's notes. The instrument setting may change over the track. Furthermore, tracks can help to allocate notes among available output channels or performance devices or tape recorder "tracks". Tracks can also help to adapt polyphonic data to monophonic output channels.

4. Tracks are a good match to simple sound software. Each track is a place to hold state settings like "dynamic mark pp," "time signature 3/4," "mute this track," etc., just as it is a context for instrument settings. This is a lot like a text stream with running "font" and "face" properties (attributes). Running state is usually more compact than, say, storing an instrument setting in every note event. It's also a useful way to organize "attributes" of notes. With "running track state" we can define new note attributes in an upward- and backward-compatible way.

   Running track state can be expanded (run decoded) while loading a track into memory or while playing the track. The runtime track state must be reinitialized every time the score is played.

Separated vs. Interleaved Tracks

Multi-track data could be stored either as separate event streams or interleaved into one stream. To interleave the streams, each event has to carry a "track number" attribute.

If we were designing an editable score format, we might interleave the streams so that nearby events are stored nearby. This helps when searching the data, especially if you can't fit the entire score into memory at once. But it takes extra storage for the track numbers and may take extra work to manipulate the interleaved tracks.

The musical score format FORM SMUS is intended for simple loading and playback of small scores that fit entirely in main memory. So we chose to store its tracks separately.

There can be up to 255 tracks in a FORM SMUS. Each track is stored as a TRAK chunk. The count of tracks (the number of TRAK chunks) is recorded in the SHDR chunk at the beginning of the FORM SMUS. The TRAK chunks appear in numerical order 1, 2, 3, ... This is also priority order, with the most important track first. A player program that can handle up to N parallel tracks should read the first N tracks and ignore any others.

The different tracks in a score may have different lengths. This is true both of storage length and of playback duration.

Instrument Registers

In SSSP, each note event points to a "timbre object," which supplies the "instrument" (the sound driver data) for that note. FORM SMUS stores these pointers as a "current instrument setting" for each track. It is just a run encoded version of the same information. SSSP uses a symbol table to hold all the pointers to "timbre object." SMUS uses INS1 chunks for the same purpose. They name the score's instruments.

The actual instrument data to use depends on the playback environment, but we want the score to be independent of environment. Different playback environments have different audio output hardware and different sound driver software. Also, there are channel allocation issues like how many output channels there are, which ones are polyphonic, and which I/O ports they're connected to. If you use MIDI to control the instruments, you get into issues of what kind of device is listening to each MIDI channel and what each of its preset sounds like. If you use computer-based instruments, you need driver-specific data like waveform tables and oscillator parameters.

We just want to put some orchestration in the score. If the score wants a "piano," we let the playback program to find a "piano."

Instrument reference can be by name or by MIDI channel and preset number. A reference from a SMUS score to actual instrument data is normally by name The score simply names the instrument, for instance "tubular bells." It is up to the player program to find suitable instrument data for its output devices. (More on locating instruments below.)

A SMUS score can also ask for a specific MIDI channel number and preset number. MIDI programs may honor these specific requests. But these channel allocations can become obsolete or the score may be played without MIDI hardware. In such cases, the player program should fall back to instrument reference by name.

Each reference from a SMUS track to an instrument is via an "instrument register". Each track selects an instrument register which in turn points to the specific instrument data.

Each score has an array of instrument registers. Each track has a "current instrument setting," which is simply an index number into this array. This is like setting a raster image's pixel to a specific color number (a reference to a color value through a "color register") or setting a text character to a specific font number (a reference to a font through a "font register"). This is diagrammed below.

```
Track 1 |Set Inst 2| Note | Note | Set Inst 1 | Note | Note | Note |...
              |                         |
+-------------+                         |
|  +------------------------------------+
|  |         +-------------------------------------------------------+
|  +-------->|"piano"  ----------------> (internal piano data)        |
+----------->|"guitar" ----------------> (internal guitar data)       |
             |"Spanish guitar" --------> (internal Spanish guitar data)|
   +-------->|"bass drum" -------------> (internal bass drum data)    |
   |         +-------------------------------------------------------+
   |
   +----------+
              |
Track 2 |Set Inst 4| Note | Note | Note | Note | Note | Note | Note |...
```

"INS1" chunks in a SMUS score name the instruments to use for that score. The player program uses these names to locate instrument data.

To locate instrument data, the player performs the following steps. For each instrument register, check for a suitable instrument with the right name. "Suitable" means usable with an available output device and driver. (Use case-independent name comparisons.)

1. Initialize the instrument register to point to a built-in default instrument. (Every player program must have default instruments. Simple programs stop here. For fancier programs, the default instruments are a backstop in case the search fails.)

2. Check any instrument FORMs embedded in the FORM SMUS. (This is an "instrument cache.")

3. Else check the default instruments.

4. Else search the local "instrument library". (The library might simply be a disk directory.)

5. If all else fails, display the desired instrument name and ask the user to pick an available one.

This algorithm can be implemented to varying degrees of fanciness. It is ok to stop searching after step 1, 2, 3, or 4. If exact instrument name matches fail, it's ok to try approximate matches. For example, search for any kind of "guitar" if you can't find a "Spanish guitar". In any case, a player only has to search for instruments while loading a score.

When the embedded instruments are suitable, they save the program from asking the user to insert the "right" disk in a drive and searching that disk for the "right" instrument. But it's just a cache. In practice, we rarely move scores between environments so the cache often works. When the score is moved, embedded instruments must be discarded (a cache miss) and other instrument data used.

Be careful to distinguish an instrument's name from its filename (the contents name vs. container name). A musical instrument FORM should contain a NAME chunk that says what instrument it really is. Its filename, on the other hand, is a handle used to locate the FORM. Filenames are affected by external factors like drives, directories, and filename character and length limits. Instrument names are not.

Issue: Consider instrument naming conventions for consistency. Consider a naming convention that aids approximate matches. For example, we could accept "guitar, bass1" if we didn't find "guitar, bass." Failing that, we could accept "guitar" or any name starting with "guitar."

Set Instrument Events

If the player implements the set-instrument score event, each track can change instrument numbers while playing. That is, it can switch between the loaded instruments.

Initial Instrument Settings

Each time a score is played, every tracks' running state information must be initialized. Specifically, each track's instrument number should be initialized to its track number--Track 1 to instrument 1, etc. It is as if each track began with a set-instrument event.

In this way, programs that do not implement the set-instrument event still assign an instrument to each track. The INS1 chunks imply these initial instrument settings.

MIDI Instruments

As mentioned above, A SMUS score can also ask for MIDI instruments. This is done by putting the MIDI channel and preset numbers in an INS1 chunk with the instrument name. Some programs will honor these requests, while others will just find instruments by name.

MIDI Recorder and sequencer programs may simply transcribe the MIDI channel and preset commands in a recording session. For this purpose, set-MIDI-channel and set-MIDI-preset events can be embedded in a SMUS score's tracks. Most programs should ignore these events. An editor program that wants to exchange scores with such programs should recognize these events. It should let the user change them to the more general set-instrument events.

STANDARD DATA AND PROPERTY CHUNKS

A FORM SMUS contains a required property "SHDR" followed by any number of parallel "track" data chunks "TRAK". Optional property chunks such as "NAME," copyright "(c)," and instrument reference "INS1" may also appear. Any of the properties may be shared over a LIST of FORMs SMUS by putting them in a PROP SMUS. (See the IFF reference.)

Required Property SHDR

The required property "SHDR" holds an SScoreHeader as defined in these C declarations and following documentation. An SHDR specifies global information for the score. It must appear before the TRAKs in a FORM SMUS.

```
   #define ID_SMUS MakeID('S', 'M', 'U', 'S')
   #define ID_SHDR MakeID('S', 'H', 'D', 'R')

   typedef struct {
       UWORD tempo;      /* tempo, 128ths quarter note/minute */
       UBYTE volume;     /* overall playback volume 0 through 127 */
       UBYTE ctTrack;    /* count of tracks in the score */
       } SScoreHeader;
```

Implementation details: In the C struct definitions in this document, fields are filed in the order shown. A UBYTE field is packed into an eight-bit byte. Programs should set all "pad" fields to 0. MakeID is a C macro defined in the main IFF document and in the source file IFF.h.

The field tempo gives the nominal tempo for all tracks in the score. It is expressed in 128ths of a quarter note per minute; that is, 1 represents 1 quarter note per 128 minutes while 12800 represents 100 quarter notes per minute. You may think of this as a fixed-point fraction with a nine-bit integer part and a seven-bit fractional part (to the right of the point). A course-tempoed program may simply shift tempo right by seven bits to get a whole number of quarter notes per minute. The tempo field can store tempi in the range 0 up to 512. The playback program may adjust this tempo, perhaps under user control.

Actually, this global tempo could actually be just an initial tempo if there are any "set tempo" SEvents inside the score (see TRAK, below). Or the global tempo could be scaled by "scale tempo" SEvents inside the score. These are potential extensions that can safely be ignored by current programs. (See More SEvents To Be Defined, below.)

The field volume gives an overall nominal playback volume for all tracks in the score. The range of volume values 0 through 127 is like a MIDI key velocity value. The playback program may adjust this volume, perhaps under direction of a user "volume control". Actually, this global volume level could be scaled by dynamic-mark SEvents inside the score (see TRAK, below).

The field ctTrack holds the count of tracks--that is, the number of TRAK chunks in the FORM SMUS (see below). This information helps the reader prepare for the following data.

A playback program will typically load the score and call a driver routine PlayScore(tracks, tempo, volume), supplying the tempo and volume from the SHDR chunk.

Optional Text Chunks NAME, (c), AUTH, ANNO

Several text chunks may be included in a FORM SMUS to keep ancillary information.

The optional property "NAME" names the musical score, for instance "Fugue in C".

The optional property "(c) " holds a copyright notice for the score. The chunk ID "(c) " serves the function of the copyright characters "(c) ". For example, a "(c) " chunk containing "1986 Electronic Arts" means "(c) 1986 Electronic Arts".

The optional property "AUTH" holds the name of the score's author.

The chunk types "NAME," "(c) ," and "AUTH" are property chunks. Putting more than one NAME (or other) property in a FORM is redundant. Just the last NAME counts. A property should be shorter than 256 characters. Properties can appear in a PROP SMUS to share them over a LIST of FORMs SMUS.

The optional data chunk "ANNO" holds any text annotations typed in by the author. An ANNO chunk is not a property chunk, so you can put more than one in a FORM SMUS. You can make ANNO chunks any length up to 2**31 - 1 characters, but 32767 is a practical limit. Since they are not properties, ANNO chunks don't belong in a PROP SMUS. That means they cannot be shared over a LIST of FORMs SMUS.

Syntactically, each of these chunks contains an array of eight-bit ASCII characters in the range R S (SP, hex 20) through R~S (tilde, hex 7F), just like a standard "TEXT" chunk. (See "Strings, String Chunks, and String Properties" in "EA IFF 85" Electronic Arts Interchange File Format.) The chunk's ckSize field holds the count of characters.

```
   #define ID_NAME MakeID('N', 'A', 'M', 'E')
        /* NAME chunk contains a CHAR[], the musical score's name. */

   #define ID_Copyright MakeID('(', 'c', ')', ' ')
        /* "(c) " chunk contains a CHAR[], the FORM's copyright notice. */

   #define ID_AUTH MakeID('A', 'U', 'T', 'H')
        /* AUTH chunk contains a CHAR[], the name of the score's author. */

   #define ID_ANNO MakeID('A', 'N', 'N', 'O')
        /* ANNO chunk contains a CHAR[], author's text annotations. */
```

Remember to store a 0 pad byte after any odd-length chunk.

Optional Property INS1

The "INS1" chunks in a FORM SMUS identify the instruments to use for this score. A program can ignore INS1 chunks and stick with its built-in default instrument assignments. Or it can use them to locate instrument data. (See "Instrument Registers" in the "Background" section above.)

```
#define ID_INS1 MakeID('I', 'N', 'S', '1')

/* Values for the RefInstrument field "type". */
#define INS1_Name  0 /* just use the name; ignore data1, data2 */
#define INS1_MIDI  1 /* <<data1, data2> = MIDI <<channel, preset> */

typedef struct {
    UBYTE register;  /* set this instrument register number */
    UBYTE type;      /* instrument reference type */
    UBYTE data1, data2;      /* depends on the "type" field */
    CHAR  name[];    /* instrument name      */
    } RefInstrument;
```

An INS1 chunk names the instrument for instrument register number register. The register field can range from 0 through 255. In practice, most scores will need only a few instrument registers.

The name field gives a text name for the instrument. The string length can be determined from the ckSize of the INS1 chunk. The string is simply an array of 8-bit ASCII characters in the range R S (SP, hex 20) through R~S (tilde, hex 7F).

Besides the instrument name, an INS1 chunk has two data numbers to help locate an instrument. The use of these data numbers is controlled by the type field. A value type = INS1_Name means just find an instrument by name. In this case, data1 and data2 should just be set to 0. A value type = INS1_MIDI means look for an instrument on MIDI channel # data1, preset # data2. Programs and computers without MIDI outputs will just ignore the MIDI data. They'll always look for the named instrument. Other values of the type field are reserved for future standardization.

See the "Background" section above for the algorithm for locating instrument data by name.

Obsolete Property INST

The chunk type "INST" is obsolete in SMUS. It was revised to form the "INS1" chunk.

Data Chunk TRAK

The main content of a score is stored in one or more TRAK chunks representing parallel "tracks"--one TRAK chunk per track.

The content of a TRAK chunk is an array of 16-bit "events" such as "note," "rest," and "set instrument." Events are really commands to a simple scheduler, stored in time order. The tracks can be polyphonic, that is, they can contain chorded "note" events.

Each event is stored as an "SEvent" record. ("SEvent" means "simple musical event.") Each SEvent has an eight-bit type field called an "sID" and eight bits of type-dependent data. This is like a machine language instruction with an eight-bit opcode and an eight-bit operand.

This format is extensible since new event types can be defined in the future. The "note" and "rest" events are the only ones that every program must understand. We will carefully design any new event types so that programs can safely skip over unrecognized events in a score.

Caution: SID codes must be allocated by a central clearinghouse to avoid conflicts.

Here are the C type definitions for TRAK and SEvent and the currently defined sID values. Details on each SEvent follow the definitions.

```
#define ID_TRAK MakeID('T', 'R', 'A', 'K')

/* TRAK chunk contains an SEvent[]. */

/* SEvent: Simple musical event. */
typedef struct {
    UBYTE sID;        /* SEvent type code */
    UBYTE data;       /* sID-dependent data */
    } SEvent;

/* SEvent type codes "sID". */
#define SID_FirstNote     0
#define SID_LastNote    127   /* sIDs in the range SID_FirstNote through
                               * SID_LastNote (sign bit = 0) are notes.
                               * sID is the MIDI tone number (pitch). */
#define SID_Rest        128   /* a rest (same data format as a note). */
#define SID_Instrument  129   /* set instrument number for this track. */
#define SID_TimeSig     130   /* set time signature for this track. */
#define SID_KeySig      131   /* set key signature for this track. */
#define SID_Dynamic     132   /* set volume for this track. */
#define SID_MIDI_Chnl   133   /* set MIDI channel number (sequencers) */
#define SID_MIDI_Preset 134   /* set MIDI preset number (sequencers) */

/* SID values 144 through 159: reserved for Instant Music SEvents. */

/* Remaining sID values up through 254:
 * reserved for future standardization. */

#define SID_Mark        255   /* sID reserved for an end-mark in RAM. */
```

Note and Rest SEvents

The note and rest SEvents SID_FirstNote through SID_Rest have the following structure overlaid onto the SEvent structure:

```
typedef struct {
    UBYTE    tone;        /* MIDI tone number 0 to 127; 128 = rest */
    unsigned chord  :1,   /* 1 = a chorded note */
    tieOut   :1,          /* 1 = tied to the next note or chord */
    nTuplet  :2,          /* 0 = none, 1 = triplet, 2 = quintuplet,
                           * 3 = septuplet */
    dot      :1,          /* dotted note; multiply duration by 3/2 */
    division :3;          /* basic note duration is 2-division:
                           * 0 = whole note, 1 = half note,
                           * 2 = quarter note, * I 7 = 128th note */
    } SNote;
```

Implementation details: Unsigned ":n" fields are packed into n bits in the order shown, most significant bit to least significant bit.

An SNote fits into 16 bits like any other SEvent. Warning: Some compilers do not implement bit-packed fields properly. For example, Lattice 68000 C pads a group of bit fields out to a LONG, which would make SNote take 5-bytes! In that situation, use the bit-field constants defined below.

The SNote structure describes one "note" or "rest" in a track. The field SNote.tone, which is overlaid with the SEvent.sID field, indicates the MIDI tone number (pitch) in the range 0 through 127. A value of 128 indicates a rest.

The fields nTuplet, dot, and division together give the duration of the note or rest. The division gives the basic duration: whole note, half note, etc. The dot indicates if the note or rest is dotted. A dotted note is 3/2 as long as an undotted note. The value nTuplet (0 through 3) tells if this note or rest is part of an N-tuplet of order 1 (normal), 3, 5, or 7; an N-tuplet of order (2J*JnTupletJ+J1). A triplet note is 2/3 as long as a normal note, while a quintuplet is 4/5 as long and a septuplet is 6/7 as long.

Putting these three fields together, the duration of the note or rest is 2-division * {1, 3/2} * {1, 2/3, 4/5, 6/7}. These three fields are contiguous so you can easily convert to your local duration encoding by using the combined six bits as an index into a mapping table.

The field chord indicates if the note is chorded with the following note (which is supposed to have the same duration). A group of notes may be chorded together by setting the chord bit of all but the last one. (In the terminology of SSSP and GSCR, setting the chord bit to 1 makes the "entry delay" 0.) A monophonic-track player can simply ignore any SNote event whose chord bit is set, either by discarding it when reading the track or by skipping it when playing the track.

Programs that create polyphonic tracks are expected to store the most important note of each chord last, which is the note with the 0 chord bit. This way, monophonic programs will play the most important note of the chord. The most important note might be the chord's root note or its melody note.

If the field tieOut is set, the note is tied to the following note in the track if the following note has the same pitch. A group of tied notes is played as a single note whose duration is the sum of the component durations. Actually, the tie mechanism ties a group of one or more chorded notes to another group of one or more chorded notes. Every note in a tied chord should have its tieOut bit set.

Of course, the chord and tieOut fields don't apply to SID_Rest SEvents.

Programs should be robust enough to ignore an unresolved tie, that is, a note whose tieOut bit is set but isn't followed by a note of the same pitch. If that's true, monophonic-track programs can simply ignore chorded notes even in the presense of ties. That is, tied chords pose no extra problems.

A rest event (sID = SID_Rest) has the same SEvent.data field as a note. It tells the duration of the rest. The chord and tieOut fields of rest events are ignored.

Within a TRAK chunk, note and rest events appear in time order.

Instead of the bit-packed structure SNote, it might be easier to assemble data values by or-ing constants and to disassemble them by masking and shifting. In that case, use the following definitions.

```
#define noteChord  (1<<<<7)    /* note is chorded to next note */

#define noteTieOut (1<<<<6)    /* tied to next note/chord */

#define noteNShift 4    /* shift count for nTuplet field */
#define noteN3     (1<<<<noteNShift)    /* note is a triplet */
#define noteN5     (2<<<<noteNShift)    /* note is a quintuplet */
#define noteN7     (3<<<<noteNShift)    /* note is a septuplet */
#define noteNMask  noteN7    /* bit mask for the nTuplet field */

#define noteDot    (1<<<<3)  /* note is dotted */

#define noteD1     0    /* whole note division */
#define noteD2     1    /* half note division */
#define noteD4     2    /* quarter note division */
#define noteD8     3     /* eighth note division */
#define noteD16    4     /* sixteenth note division */
#define noteD32    5     /* thirty-secondth note division */
#define noteD64    6     /* sixty-fourth note division */
#define noteD128   7     /* 1/128 note division */

#define noteDMask  noteD128    /* bit mask for the division field */
#define noteDurMask 0x3F       /* mask for combined duration fields */
```

Note: The remaining SEvent types are optional. A writer program doesn't have to generate them. A reader program can safely ignore them.

Set Instrument SEvent

One of the running state variables of every track is an instrument number. An instrument number is the array index of an "instrument register," which in turn points to an instrument. (See "Instrument Registers," in the "Background" section.) This is like a color number in a bitmap; a reference to a color through a "color register."

The initial setting for each track's instrument number is the track number. Track 1 is set to instrument 1, etc. Each time the score is played, every track's instrument number should be reset to the track number.

The SEvent SID_Instrument changes the instrument number for a track, that is, which instrument plays the following notes. Its SEvent.data field is an instrument register number in the range 0 through 255. If a program does not implement the SID_Instrument event, each track is fixed to one instrument.

Set Time Signature SEvent

The SEvent SID_TimeSig sets the time signature for the track. A "time signature" SEvent has the following structure overlaid on the SEvent structure:

```
typedef struct {
    UBYTE    type;           /* = SID_TimeSig */
    unsigned timeNSig :5,    /* time sig. "numerator" is timeNSig + 1 */
    timeDSig :3;             /* time sig. "denominator" is 2timeDSig:
                              * 0 = whole note, 1 = half note,
                              * 2 = quarter note, I 7 = 128th note */

    } STimeSig;
```

Implementation details: Unsigned ":n" fields are packed into n bits in the order shown, most significant bit to least significant bit. An STimeSig fits into 16 bits like any other SEvent. Warning: Some compilers don't implement bit-packed fields properly. For example, Lattice C pads a group of bit fields out to a LONG, which would make an STimeSig take 5-bytes! In that situation, use the bit-field constants defined below.

The field type contains the value SID_TimeSig, indicating that this SEvent is a "time signature" event. The field timeNSig indicates the time signature "numerator" is timeNSig + 1, that is, 1 through 32 beats per measure. The field timeDSig indicates the time signature "denominator" is 2timeDSig, that is each "beat" is a 2-timeDSig note (see SNote division, above). So 4/4 time is expressed as timeNSig = 3, timeDSig = 2. The default time signature is 4/4 time.

Be aware that the time signature has no effect on the score's playback. Tempo is uniformly expressed in quarter notes per minute, independent of time signature. (Quarter notes per minute equals beats per minute only if timeDSig = 2, n/4 time). Nonetheless, any program that has time signatures should put them at the beginning of each TRAK when creating a FORM SMUS because music editors need them.

Instead of the bit-packed structure STimeSig, it might be easier to assemble data values by or-ing constants and to disassemble them by masking and shifting. In that case, use the following definitions.

```
#define timeNMask  0xF8   /* bit mask for the timeNSig field */
#define timeNShift 3      /* shift count for  timeNSig field */
#define timeDMask  0x07   /* bit mask for the timeDSig field */
```

Key Signature SEvent

An SEvent SID_KeySig sets the key signature for the track. Its data field is a UBYTE number encoding a major key:

Data	Key	Music Notation	Data	Key	Music Notation
J0	C maj				
J1	G	#	J8	F	b
J2	D	##	J9	Bb	bb
J3	A	###	10	Eb	bbb
J4	E	####	11	Ab	bbbb
J5	B	#####	12	Db	bbbbb
J6	F#	######	13	Gb	bbbbbb
J7	C#	#######	14	Cb	bbbbbbb

A SID_KeySig SEvent changes the key for the following notes in that track. C major is the default key in every track before the first SID_KeySig SEvent.

Dynamic Mark SEvent

An SEvent SID_Dynamic represents a dynamic mark like ppp and fff in Common Music Notation. Its data field is a MIDI key velocity number 0 through 127. This sets a "volume control" for following notes in the track. This "track volume control" is scaled by the overall score volume in the SHDR chunk. The default dynamic level is 127 (full volume).

Set MIDI Channel SEvent

The SEvent SID_MIDI_Chnl is for recorder programs to record the set-MIDI-channel low level event. The data byte contains a MIDI channel number. Other programs should use instrument registers instead.

Set MIDI Preset SEvent

The SEvent SID_MIDI_Preset is for recorder programs to record the set-MIDI-preset low level event. The data byte contains a MIDI preset number. Other programs should use instrument registers instead.

Instant Music Private SEvents

Sixteen SEvents are used for private data for the Instant Music program. SID values 144 through 159 are reserved for this purpose. Other programs should skip over these SEvents.

End-Mark SEvent

The SEvent type SID_Mark is reserved for an end marker in working memory. This event is never stored in a file. It may be useful if you decide to use the filed TRAK format intact in working memory.

More SEvents To Be Defined

More SEvents can be defined in the future. The sID codes 133 through 143 and 160 through 254 are reserved for future needs. Caution: sID codes must be allocated by a central "clearinghouse" to avoid conflicts. When this SMUS standard passes the "draft" state, Commodore-Amiga will be in charge of this activity.

The following SEvent types are under consideration and should not yet be used.

Issue: A "change tempo" SEvent changes tempo during a score. Changing the tempo affects all tracks, not just the track containing the change tempo event. One possibility is a "scale tempo" SEvent SID_ScaleTempo that rescales the global tempo:

```
        currentTempo := globalTempo * (data + 1) / 128
```

This can scale the global tempo (in the SHDR) anywhere from x1/128 to x2 in roughly 1% increments.

An alternative is two events: SID_SetHTempo and SID_SetLTempo. SID_SetHTempo gives the high byte and SID_SetLTempo gives the low byte of a new tempo setting in 128ths quarter note/minute. SetHTempo automatically sets the low byte to 0, so the SetLTempo event isn't needed for course settings. In this scheme, the SHDR's tempo is simply a starting tempo.

An advantage of SID_ScaleTempo is that the playback program can just alter the global tempo to adjust the overall performance time and still easily implement tempo variations during the score. However, the "set tempo" SEvent may be simpler to generate.

Issue: The events SID_BeginRepeat and SID_EndRepeat define a repeat span for one track. The span of events between a BeginRepeat and an EndRepeat is played twice. The SEvent.data field in the BeginRepeat event could give an iteration count,1 through 255 times or 0 for "repeat forever."

Repeat spans can be nested. All repeat spans automatically end at the end of the track.

An event SID_Ending begins a section like "first ending" or "second ending". The SEvent.data field gives the ending number. This SID_Ending event only applies to the innermost repeat group. (Consider generalizing it.)

A more general alternative is a "subtrack" or "subscore" event. A "subtrack" event is essentially a "subroutine call" to another series of SEvents. This is a nice way to encode all the possible variations of repeats, first endings, codas, and such.

To define a subtrack, we must demark its start and end. One possibility is to define a relative brach-to-subtrack event SID_BSR and a return-from-subtrack event SID_RTS. The eight-bit data field in the SID_BSR event can reach as far as 512 SEvents. A second possibility is to call a subtrack by index number, with an IFF chunk outside the TRAK defining the start and end of all subtracks. This is very general because a portion of one subtrack can be used as another subtrack. It also models the tape recording practice of first "laying down a track" and then selecting portions of it to play and repeat. To embody the music theory idea of playing a sequence like "ABBA", just compose the "main" track entirely of subtrack events. A third possibility is to use a numbered subtrack chunk "STRK" for each subroutine.

PRIVATE CHUNKS

As in any IFF FORM, there can be private chunks in a FORM SMUS that are designed for one particular program to store its private information. All IFF reader programs skip over unrecognized chunks, so the presence of private chunks can't hurt. Instant Music stores some global score information in a chunk of ID "IRev".

QUICK REFERENCE

Type Definitions

Here is a collection of the C type definitions in this document. In the "struct" type definitions, fields are filed in the order shown. A UBYTE field is packed into an eight-bit byte. Programs should set all "pad" fields to zero.

```
#define ID_SMUS MakeID('S', 'M', 'U', 'S')
#define ID_SHDR MakeID('S', 'H', 'D', 'R')

typedef struct {
        UWORD tempo;    /* tempo, 128ths quarter note/minute */
        UBYTE volume;   /* overall playback volume 0 through 127 */
        UBYTE ctTrack;  /* count of tracks in the score */
        } SScoreHeader;

#define ID_NAME MakeID('N', 'A', 'M', 'E')
   /* NAME chunk contains a CHAR[], the musical score's name. */
#define ID_Copyright MakeID('(', 'c', ')', ' ')
   /* "(c) " chunk contains a CHAR[], the FORM's copyright notice. */
#define ID_AUTH MakeID('A', 'U', 'T', 'H')
   /* AUTH chunk contains a CHAR[], the name of the score's author. */
#define ID_ANNO MakeID('A', 'N', 'N', 'O')
   /* ANNO chunk contains a CHAR[], author's text annotations. */
#define ID_INS1 MakeID('I', 'N', 'S', '1')
   /* Values for the RefInstrument field "type". */
#define INS1_Name 0     /* just use the name; ignore data1, data2 */
#define INS1_MIDI 1     /* <<data1, data2> = MIDI <<channel, preset> */

typedef struct {
        UBYTE register; /* set this instrument register number */
        UBYTE type;     /* instrument reference type */
        UBYTE data1, data2;     /* depends on the "type" field */
        CHAR  name[];   /* instrument name      */
        } RefInstrument;

#define ID_TRAK MakeID('T', 'R', 'A', 'K')
/* TRAK chunk contains an SEvent[]. */

/* SEvent: Simple musical event. */
typedef struct {
        UBYTE sID;      /* SEvent type code */
        UBYTE data;     /* sID-dependent data */
        } SEvent;

/* SEvent type codes "sID". */
#define SID_FirstNote     0
#define SID_LastNote    127     /* sIDs in the range SID_FirstNote through
                                 * SID_LastNote (sign bit = 0) are notes.
                                 * sID is the MIDI tone number (pitch). */
#define SID_Rest        128     /* a rest (same data format as a note) */

#define SID_Instrument  129     /* set instrument number for this track. */
#define SID_TimeSig     130     /* set time signature for this track. */
```

```
#define SID_KeySig      131     /* set key signature for this track. */
#define SID_Dynamic     132     /* set volume for this track. */
#define SID_MIDI_Chnl   133     /* set MIDI channel number (sequencers) */
#define SID_MIDI_Preset 134     /* set MIDI preset number (sequencers) */

/* SID values 144 through 159: reserved for Instant Music SEvents. */

/* Remaining sID values up through 254: reserved for future
 * standardization. */

#define SID_Mark        255     /* sID reserved for an end-mark in RAM. */

/* SID_FirstNote..SID_LastNote, SID_Rest SEvents */
typedef struct {
        UBYTE    tone;         /* MIDI tone number 0 to 127; 128 = rest */
        unsigned chord  :1,    /* 1 = a chorded note */
        tieOut   :1,           /* 1 = tied to the next note or chord */
        nTuplet  :2,           /* 0 = none, 1 = triplet, 2 = quintuplet,
                                * 3 = septuplet */
        dot      :1,           /* dotted note; multiply duration by 3/2 */
        division :3            /* basic note duration is 2-division:
                                * 0 = whole * note, 1 = half note,
                                * 2 = quarter note, I 7 = 128th note */
        } SNote;

#define noteChord   (1<<<<7)      /* note is chorded to next note */
#define noteTieOut  (1<<<<6)      /* tied to next note/chord */
#define noteNShift 4    /* shift count for nTuplet field */
#define noteN3      (1<<<<noteNShift)     /* note is a triplet */
#define noteN5      (2<<<<noteNShift)     /* note is a quintuplet */
#define noteN7      (3<<<<noteNShift)     /* note is a septuplet */
#define noteNMask   noteN7        /* bit mask for the nTuplet field */
#define noteDot     (1<<<<3)      /* note is dotted */

#define noteD1      0     /* whole note division */
#define noteD2      1     /* half note division */
#define noteD4      2     /* quarter note division */
#define noteD8      3     /* eighth note division */
#define noteD16     4     /* sixteenth note division */
#define noteD32     5     /* thirty-secondth note division */
#define noteD64     6     /* sixty-fourth note division */
#define noteD128    7     /* 1/128 note division */

#define noteDMask   noteD128      /* bit mask for the division field */
#define noteDurMask 0x3F          /* mask for combined duration fields */

/* SID_Instrument SEvent "data" value is an
 * instrument register number 0 through 255. */

/* SID_TimeSig SEvent   */
typedef struct {
        UBYTE    type;          /* = SID_TimeSig */
        unsigned timeNSig :5,   /* time sig. "numerator" is timeNSig + 1 */
        timeDSig :3;            /* time sig. "denominator" is 2timeDSig:
                                 * 0 = whole note, 1 = half note,
                                 * 2 = quarter note, I 7 = 128th note */
        } STimeSig;

#define timeNMask  0xF8 /* bit mask for the timeNSig field */
#define timeNShift 3    /* shift count for  timeNSig field */
#define timeDMask  0x07 /* bit mask for the timeDSig field */

/* SID_KeySig SEvent */
/* "data" value 0 = Cmaj; 1 through 7 = G,D,A,E,B,F#,C#;
 * 8 through 14 = F,Bb,Eb,Ab,Db,Gb,Cb. */

/* SID_Dynamic SEvent */
/* "data" value is a MIDI key velocity 0..127. */
```

```
SMUS Regular Expression
-----------------------
```

Here's a regular expression summary of the FORM SMUS syntax. This could be an IFF file or part of one.

```
SMUS       ::= "FORM" #{ "SMUS" SHDR [NAME] [Copyright] [AUTH] [IRev]
                         ANNO* INS1*  TRAK*  InstrForm* }

SHDR       ::= "SHDR" #{ SScoreHeader  }
NAME       ::= "NAME" #{ CHAR*         } [0]
Copyright  ::= "(c) " #{ CHAR*         } [0]
AUTH       ::= "AUTH" #{ CHAR*         } [0]
IRev       ::= "IRev" #{ ...           }

ANNO       ::= "ANNO" #{ CHAR*         } [0]
INS1       ::= "INS1" #{ RefInstrument } [0]

TRA        ::= "TRAK" #{ SEvent*       }

InstrForm  ::= "FORM" #{ ...           }
```

The token "#" represents a ckSize LONG count of the following {braced} data bytes. Literal items are shown in "quotes", [square bracket items] are optional, and "*" means 0 or more replications. A sometimes-needed pad byte is shown as "[0]".

Actually, the order of chunks in a FORM SMUS is not as strict as this regular expression indicates. The SHDR, NAME, Copyright, AUTH, IRev, ANNO, and INS1 chunks may appear in any order, as long as they precede the TRAK chunks.

The chunk RInstrFormS represents any kind of instrument data FORM embedded in the FORM SMUS. For example, see the document "8SVX" IFF 8-Bit Sampled Voice. Of course, a recipient program will ignore an instrument FORM if it doesn't recognize that FORM type.

SMUS EXAMPLE

Here's a box diagram for a simple example, a SMUS with two instruments
and two tracks. Each track contains 1 note event and 1 rest event.

```
+-----------------------------------------+          ------
|'FORM'            94                     |            ^
|   +-------------------------------------+ |          |
|   |'SMUS'                               | |          |
|   +-------------------------------------+ |          |
|   | +---------------------------------+ | |          |
|   | | 'SHDR'       4                  | | |          |
|   | | 12800, 127, 2                   | | |          |
|   | +---------------------------------+ | |          |
|   | +---------------------------------+ | |          |
|   | | 'NAME'       10                 | | |          |
|   | | 'Fugue in C'                    | | |          |
|   | +---------------------------------+ | |          |
|   | +---------------------------------+ | |          |
|   | | 'INS1'       9                  | | |      (94 bytes)
|   | | 1,0,0,0,'piano'                 | | |          |
|   | +---------------------------------+ | |          |
|   | 0                                   | |          |
|   | +---------------------------------+ | |          |
|   | | 'INS1'       10                 | | |          |
|   | | 2,0,0,0,'guitar'                | | |          |
|   | +---------------------------------+ | |          |
|   | +---------------------------------+ | |          |
|   | | 'TRAK'       4                  | | |          |
|   | | 60, 16, 128, 16                 | | |          |
|   | +---------------------------------+ | |          |
|   | +---------------------------------+ | |          |
|   | | 'TRAK'       4                  | | |          |
|   | | 128, 16, 60, 16                 | | |          |
|   | +---------------------------------+ | |          |
|   +-------------------------------------+ |          V
+-----------------------------------------+          -----
```

STANDARDS COMMITTEE

The following people contributed to the design of this IFF standard:

Bob "Kodiak" Burns, Commodore-Amiga
R. J. Mical, Commodore-Amiga
Jerry Morrison, Electronic Arts
Greg Riker, Electronic Arts
Steve Shaw, Electronic Arts
Barry Walsh, Commodore-Amiga
Ralph Bellafatto, Cherry Lane Technologies
Geoff Brown, Uhuru Sound Software
Steve Hayes, Electronic Arts
Jerry Morrison, Electronic Arts

"8SVX" IFF 8-Bit Sampled Voice

Date: February 7, 1985
From: Steve Hayes and Jerry Morrison, Electronic Arts
Status: Adopted

INTRODUCTION

This section covers FORM "8SVX". An 8SVX is an IFF "data section" or "FORM" (which can be an IFF file or a part of one) containing a digitally sampled audio voice consisting of eight-bit samples.
A voice can be a one-shot sound or (with repetition and pitch scaling) a musical instrument.

The 8SVX format is designed for playback hardware that uses eight-bit samples attenuated by a volume control for good overall signal-to-noise ratio. A FORM 8SVX stores eight-bit samples and a volume level.

A similar data format (or two) will be needed for higher-resolution samples (typically 12 or 16 bits). Properly converting a high-resolution sample down to 8 bits requires one pass over the data to find the minimum and maximum values and a second pass to scale each sample into the range -128 through 127. It is reasonable to store higher-resolution data in a different FORM type and convert between them.

For instruments, FORM 8SVX can record a repeating waveform optionally preceded by a start-up transient waveform. These two recorded signals can be presynthesized or sampled from an acoustic instrument. For many instruments, this representation is compact. FORM 8SVX is less practical for an instrument whose waveform changes from cycle to cycle, such as a plucked string, where a long sample is needed for accurate results.

FORM 8SVX can store an "envelope" or "amplitude contour" to enrich musical notes. A future voice FORM could also store amplitude, frequency, and filter modulations.

FORM 8SVX is geared for relatively simple musical voices, in which one waveform per octave is sufficient, the waveforms for the different octaves follow a factor-of-two size rule, and one envelope is adequate for all octaves. You could store a more general voice as a LIST containing one or more FORMs 8SVX per octave. A future voice FORM could go beyond one "one-shot" waveform and one "repeat" waveform per octave.

The next section defines the required property sound header "VHDR," optional properties name "NAME," copyright "(c) ," author "AUTH," the optional annotation data chunk "ANNO," the required data chunk "BODY," and optional envelope chunks "ATAK" and "RLSE." These are the "standard" chunks. Specialized chunks for private or future needs can be added later, for example, to hold a frequency contour or Fourier series coefficients. The 8SVX syntax is summarized in the "Quick Reference" section as a regular expression and in the "8SVX Example" section as an example box diagram. The "Fibonacci Delta Compression" section explains the optional Fibonacci-delta compression algorithm.

Caution: The VHDR structure Voice8Header has changed since the draft proposal #4 version of this document. The new structure is incompatible with that version.

Reference

"EA IFF 85" Standard for Interchange Format Files describes the underlying conventions for all IFF files.

Amiga[tm] is a trademark of Commodore-Amiga, Inc.

Electronic Arts[tm] is a trademark of Electronic Arts.

MacWrite[tm] is a trademark of Apple Computer, Inc.

STANDARD DATA AND PROPERTY CHUNKS

FORM 8SVX stores all the waveform data in one body chunk "BODY." It stores playback parameters in the required header chunk "VHDR." "VHDR" and any optional property chunks "NAME," "(c) ," and "AUTH" must all appear before the BODY chunk. Any of these properties may be shared over a LIST of FORMs 8SVX by putting them in a PROP 8SVX. (See "EA IFF 85" Standard for Interchange Format Files.)

Background

There are two ways to use FORM 8SVX: as a one-shot sampled sound or as a sampled musical instrument that plays "notes". Storing both kinds of sounds in the same kind of FORM makes it easy to play a one-shot sound as a (staccato) instrument or an instrument as a (one-note) sound.

A one-shot sound is a series of audio data samples with a nominal playback rate and amplitude. The recipient program can optionally adjust or modulate the amplitude and playback data rate.

For musical instruments, the idea is to store a sampled (or pre-synthesized) waveform that will be parameterized by pitch, duration, and amplitude to play each "note." The creator of the FORM 8SVX can supply a waveform per octave over a range of octaves for this purpose. The intent is to perform a pitch by selecting the closest octave's waveform and scaling the playback data rate. An optional "one-shot" waveform supplies an arbitrary startup transient; then a "repeat" waveform is iterated as long as necessary to sustain the note.

A FORM 8SVX can also store an envelope to modulate the waveform. Envelopes are mostly useful for variable-duration notes but could also be used for one-shot sounds.

The FORM 8SVX standard has some restrictions. For example, each octave of data must be twice as long as the next higher octave. Most sound driver software and hardware imposes additional restrictions. For example, the Amiga sound hardware requires an even number of samples in each one-shot and repeat waveform.

Required Property VHDR

The required property "VHDR" holds a Voice8Header structure as defined in these C declarations and in the following documentation. This structure holds the playback parameters for the sampled waveforms in the BODY chunk. (See "Data Chunk BODY," below, for the storage layout of these waveforms.)

```
    #define ID_8SVX MakeID('8', 'S', 'V', 'X')
    #define ID_VHDR MakeID('V', 'H', 'D', 'R')

    typedef LONG Fixed;
        /* A fixed-point value, 16 bits to the left of the point and 16
         * to the right. A Fixed is a number of 216ths, i.e. 65536ths. */

    #define Unity 0x10000L     /* Unity = Fixed 1.0 = maximum volume */

        /* sCompression: Choice of compression
         * algorithm applied to the samples. */

    #define sCmpNone       0      /* not compressed */
    #define sCmpFibDelta   1      /* Fibonacci-delta encoding (Appendix C) */

    /* Can be more kinds in the future. */

    typedef struct {
        ULONG oneShotHiSamples, /* # samples in high octave 1-shot part */
        repeatHiSamples,     /* # samples in high octave repeat part */
        samplesPerHiCycle;   /* # samples/cycle in high octave, else 0 */
        UWORD samplesPerSec;    /* data sampling rate */
        UBYTE ctOctave,      /* # octaves of waveforms */
        sCompression;        /* data compression technique used */
        Fixed volume;        /* playback volume from 0 to Unity (full
                              * volume). Map this value into the output
                              * hardware's dynamic range. */
        } Voice8Header;
```

Implementation details: Fields are filed in the order shown. The UBYTE fields are byte-packed (two per 16-bit word). MakeID is a C macro defined in the main IFF document and in the source file IFF.h.

A FORM 8SVX holds waveform data for one or more octaves, each containing a one-shot part and a repeat part. The fields oneShotHiSamples and repeatHiSamples tell the number of audio samples in the two parts of the highest frequency octave. Each successive (lower frequency) octave contains twice as many data samples in both its one-shot and repeat parts. One of these two parts can be empty across all octaves.

Limitations of Audio Output Hardware and Software

Most audio output hardware and software has limitations. The Amiga computer's sound hardware requires that all one-shot and repeat parts have even numbers of samples. Amiga sound driver software would have to adjust an odd-sized waveform, ignore an odd-sized lowest octave, or ignore odd FORMs 8SVX altogether. Some other output devices require all sample sizes to be powers of two.

The field samplesPerHiCycle tells the number of samples/cycle in the highest frequency octave of data, or else 0 for "unknown". Each successive (lower frequency) octave contains twice as many samples/cycle. The samplesPerHiCycle value is needed to compute the data rate for a desired playback pitch.

Actually, samplesPerHiCycle is an average number of samples/cycle. If the one-shot part contains pitch bends, store the samples/cycle of the repeat part in samplesPerHiCycle. The division repeatHiSamples/samplesPe rHiCycle should yield an integer number of cycles. (When the repeat waveform is repeated, a partial cycle would come out as a higher-frequency cycle with a "click".)

Some Amiga music drivers require samplesPerHiCycle to be a power of two in order to play the FORM 8SVX as a musical instrument in tune. They may even assume samplesPerHiCycle is a particular power of two without checking. (If samplesPerHiCycle is different by a factor of two, the instrument will just be played an octave too low or high.)

The field samplesPerSec gives the sound sampling rate. A program may adjust this to achieve frequency shifts or vary it dynamically to achieve pitch bends and vibrato. A program that plays a FORM 8SVX as a musical instrument would ignore samplesPerSec and select a playback rate for each musical pitch.

The field ctOctave tells how many octaves of data are stored in the BODY chunk. See "Data Chunk BODY", below, for the layout of the octaves.

The field sCompression indicates the compression scheme, if any, that was applied to the entire set of data samples stored in the BODY chunk. This field should contain one of the values defined above. Of course, the matching decompression algorithm must be applied to the BODY data before the sound can be played. (The Fibonacci-delta encoding scheme sCmpFibDelta is described at the end of this document.) Note that the whole series of data samples is compressed as a unit.

The field volume gives an overall playback volume for the waveforms (all octaves). It lets the eight-bit data samples use the full range -128 through 127 for good signal-to-noise ratio and be attenuated on playback to the desired level. The playback program should multiply this value by a "volume control" and perhaps by a playback envelope (see ATAK and RLSE, below).

Recording a One-shot Sound

To store a one-shot sound in a FORM 8SVX, set oneShotHiSamples = number of samples, repeatHiSamples = 0, samplesPerHiCycle = 0, samplesPerSec = sampling rate, and ctOctave = 1. Scale the signal amplitude to the full sampling range -128 through 127. Set the volume so the sound will playback at the desired volume level. If you set the samplesPerHiCycle field properly, the data can also be used as a musical instrument.

Experiment with data compression. If the decompressed signal sounds ok, store the compressed data in the BODY chunk and set sCompression to the compression code number.

Recording a Musical Instrument

To store a musical instrument in a FORM 8SVX, first record or synthesize as many octaves of data as you want to make available for playback. Set ctOctaves to the count of octaves. From the recorded data, excerpt an integral number of steady state cycles for the repeat part and set repeatHiSamples and samplesPerHiCycle. Either excerpt a startup transient waveform and set oneShotHiSamples or else set oneShotHiSamples to 0. Remember, the one-shot and repeat parts of each octave must be twice as long as those of the next higher octave. Scale the signal amplitude to the full sampling range and set volume to adjust the instrument playback volume. If you set the samplesPerSec field properly, the data can also be used as a one-shot sound.

A distortion-introducing compressor like sCmpFibDelta is not recommended for musical instruments, but you might try it anyway.

Typically, creators of FORM 8SVX record an acoustic instrument at just one frequency. Decimate (down-sample with filtering) to compute higher octaves. Interpolate to compute lower octaves.

If you sample an acoustic instrument at different octaves, you may find it hard to make the one-shot and repeat waveforms follow the factor-of-two rule for octaves. To compensate, lengthen an octave's one-shot part by appending replications of the repeating cycle or prepending zeros. (This will have minimal impact on the sound's start time.) You may be able to equalize the ratio one-shot-samples : repeat-samples across all octaves.

Note that a "one-shot sound" may be played as a "musical instrument" and vice versa. However, an instrument player depends on samplesPerHiCycle, and a one-shot player depends on samplesPerSec.

Playing a One-shot Sound

To play any FORM 8SVX data as a one-shot sound, first select an octave if ctOctave > 1. (The lowest-frequency octave has the greatest resolution.) Play the one-shot samples then the repeat samples, scaled by volume, at a data rate of samplesPerSec. Of course, you may adjust the playback rate and volume. You can play out an envelope, too. (See ATAK and RLSE, below.)

Playing a Musical Note

To play a musical note using any FORM 8SVX, first select the nearest octave of data from those available. Play the one-shot waveform then cycle on the repeat waveform as long as needed to sustain the note. Scale the signal by volume, perhaps also by an envelope, and by a desired note volume. Select a playback data rate s samples/second to achieve the desired frequency (in Hz):

frequency = sJ/JsamplesPerHiCycle

for the highest frequency octave.

The idea is to select an octave and one of 12 sampling rates (assuming a 12-tone scale). If the FORM 8SVX doesn't have the right octave, you can decimate or interpolate from the available data.

For musical instruments, FORM 8SVX is geared for a simple sound driver. Such a driver uses a single table of 12 data rates to reach all notes in all octaves. That is why 8SVX requires each octave of data to have twice as many samples as the next higher octave. If you restrict samplesPerHiCycle to a power of two, you can use a predetermined table of data rates.

Optional Text Chunks NAME, (c), AUTH, ANNO

Several text chunks may be included in a FORM 8SVX to keep ancillary information.

The optional property "NAME" names the voice, for instance "tubular bells".

The optional property "(c) " holds a copyright notice for the voice. The chunk ID "(c) " serves as the copyright characters ")J." For example, a "(c) " chunk containing "1986 Electronic Arts" means "(c) 1986 Electronic Arts".

The optional property "AUTH" holds the name of the instrument's "author" or "creator".

The chunk types "NAME," "(c)," and "AUTH" are property chunks. Putting more than one NAME (or other) property in a FORM is redundant. Just the last NAME counts. A property should be shorter than 256 characters. Properties can appear in a PROP 8SVX to share them over a LIST of FORMs 8SVX.

The optional data chunk "ANNO" holds any text annotations typed in by the author. An ANNO chunk is not a property chunk, so you can put more than one in a FORM 8SVX. You can make ANNO chunks any length up to 2**31 - 1 characters, but 32767 is a practical limit. Since they are not properties, ANNO chunks do not belong in a PROP 8SVX. That means they cannot be shared over a LIST of FORMs 8SVX.

Syntactically, each of these chunks contains an array of eight-bit ASCII characters in the range " " (SP, hex 20) through "~" (tilde, hex 7F), just like a standard "TEXT" chunk. (See "Strings, String Chunks, and String Properties" in "EA IFF 85" Electronic Arts Interchange File Format.) The chunk's ckSize field holds the count of characters.

```
        #define ID_NAME MakeID('N', 'A', 'M', 'E')
        /* NAME chunk contains a CHAR[], the voice's name. */

        #define ID_Copyright MakeID('(', 'c', ')', ' ')
        /* "(c) " chunk contains a CHAR[], the FORM's copyright notice. */

        #define ID_AUTH MakeID('A', 'U', 'T', 'H')
        /* AUTH chunk contains a CHAR[], the author's name. */

        #define ID_ANNO MakeID('A', 'N', 'N', 'O')
        /* ANNO chunk contains a CHAR[], author's text annotations. */
```

Remember to store a 0 pad byte after any odd-length chunk.

Optional Data Chunks ATAK and RLSE

The optional data chunks ATAK and RLSE together give a piecewise-linear "envelope" or "amplitude contour." This contour may be used to modulate the sound during playback. It is especially useful for playing musical notes of variable durations. Playback programs may ignore the supplied envelope or substitute another.

```
        #define ID_ATAK MakeID('A', 'T', 'A', 'K')
        #define ID_RLSE MakeID('R', 'L', 'S', 'E')

        typedef struct {
                UWORD duration; /* segment duration in milliseconds, > 0 */
                Fixed dest;     /* destination volume factor */
                } EGPoint;

        /* ATAK and RLSE chunks contain an EGPoint[],
         * piecewise-linear envelope.  The envelope defines a
         * function of time returning Fixed values. It is used
         * to scale the nominal volume specified in the Voice8Header. */
```

To explain the meaning of the ATAK and RLSE chunks, here is an overview of the envelope-generation algorithm. Start at 0 volume, step through the ATAK contour, hold at the sustain level (the last ATAK EGPoint's dest), and then step through the RLSE contour. Begin the release at the desired note stop time minus the total duration of the release contour (the sum of the RLSE EGPoints' durations). The attack contour should be cut short if the note is shorter than the release contour.

The envelope is a piecewise-linear function. The envelope generator interpolates between the EGPoints.

Remember to multiply the envelope function by the nominal voice header volume and by any desired note volume.

Note: The number of EGPoints in an ATAK or RLSE chunk is its ckSize / sizeof(EGPoint). In RAM, the playback program may terminate the array with a 0-duration EGPoint.

Issue: Synthesizers also provide frequency contour (pitch bend), filtering contour (wah-wah), amplitude oscillation (tremolo), frequency oscillation (vibrato), and filtering oscillation (leslie). In the future, we may define optional chunks to encode these modulations. The contours can be encoded in linear segments. The oscillations can be stored as segments with rate and depth parameters.

Data Chunk BODY

The BODY chunk contains the audio data samples.

```
    #define ID_BODY MakeID('B', 'O', 'D', 'Y')

    typedef character BYTE;     /* 8 bit signed number, -128 through 127. */

    /* BODY chunk contains a BYTE[], array of audio data samples. */
```

The BODY contains data samples grouped by octave. Within each octave are one-shot and repeat portions.

In general, the BODY has ctOctave octaves of data. The highest frequency octave comes first, comprising the fewest samples:

oneShotHiSamples + repeatHiSamples.

Each successive octave contains twice as many samples as the next higher octave but the same number of cycles. The lowest frequency octave comes last with the most samples:

$2^{ctOctave-1}$ * (oneShotHiSamples + repeatHiSamples).

The number of samples in the BODY chunk is

$(2^0 + \dots + 2^{(ctOctave-1)})$ * (oneShotHiSamples + repeatHiSamples)

To avoid playback "clicks", the one-shot part should begin with a small sample value, the one-shot part should flow smoothly into the repeat part, and the end of the repeat part should flow smoothly into the beginning of the repeat part.

If the VHDR field sCompression - sCmpNone, the BODY chunk is just an array of data bytes to feed through the specified decompresser function. All this stuff about sample sizes, octaves, and repeat parts applies to the decompressed data.

Be sure to follow an odd-length BODY chunk with a 0 pad byte.

Other Chunks

Issue: In the future, we may define an optional chunk containing Fourier series coefficients for a repeating waveform. An editor for this kind of synthesized voice could modify the coefficients and regenerate the waveform.

QUICK REFERENCE

Type Definitions

```
   #define ID_8SVX MakeID('8', 'S', 'V', 'X')
   #define ID_VHDR MakeID('V', 'H', 'D', 'R')

   typedef LONG Fixed;   /* A fixed-point value, 16 bits to the left
                          * of the point and 16 to the right. A Fixed
                          * is a number of 216ths, i.e. 65536ths. */

   #define Unity 0x10000L       /* Unity = Fixed 1.0 = maximum volume */

   /* sCompression: Choice of compression algorithm applied to the samples */

   #define sCmpNone        0    /* not compressed */
   #define sCmpFibDelta    1    /* Fibonacci-delta encoding (Appendix C) */
      /* Can be more kinds in the future. */

   typedef struct {
      ULONG oneShotHiSamples, /* # samples in the high octave 1-shot part */
      repeatHiSamples,        /* # samples in the high octave repeat part */
      samplesPerHiCycle;      /* # samples/cycle in high octave, else 0 */
      UWORD samplesPerSec;    /* data sampling rate */
      UBYTE ctOctave,         /* # octaves of waveforms */
      sCompression;           /* data compression technique used */
      Fixed volume;           /* playback volume from 0 to Unity (full
                               * volume). Map this value into the output
                               * hardware's dynamic range. */
      } Voice8Header;

   #define ID_NAME MakeID('N', 'A', 'M', 'E')
       /* NAME chunk contains a CHAR[], the voice's name. */
   #define ID_Copyright MakeID('(', 'c', ')', ' ')
       /* "(c) " chunk contains a CHAR[], the FORM's copyright notice. */
   #define ID_AUTH MakeID('A', 'U', 'T', 'H')
       /* AUTH chunk contains a CHAR[], the author's name. */
   #define ID_ANNO MakeID('A', 'N', 'N', 'O')
       /* ANNO chunk contains a CHAR[], author's text annotations. */
   #define ID_ATAK MakeID('A', 'T', 'A', 'K')
   #define ID_RLSE MakeID('R', 'L', 'S', 'E')

   typedef struct {
       UWORD duration;  /* segment duration in milliseconds, > 0 */
       Fixed dest;      /* destination volume factor */
       } EGPoint;

   /* ATAK and RLSE chunks contain an EGPoint[] piecewise-linear envelope. */
   /* The envelope defines a function of time returning Fixed values. It's
    * used to scale the nominal volume specified in the Voice8Header  */

   #define ID_BODY MakeID('B', 'O', 'D', 'Y')

   typedef character BYTE;    /* 8 bit signed number, -128 through 127. */

   /* BODY chunk contains a BYTE[], array of audio data samples. */
```

8SVX Regular Expression

Here is a regular expression summary of the FORM 8SVX syntax. This could be an IFF file or part of one.

```
8SVX       ::= "FORM" #{ "8SVX" VHDR [NAME] [Copyright] [AUTH] ANNO*
                              [ATAK] [RLSE] BODY }

VHDR       ::= "VHDR" #{ Voice8Header        }
NAME       ::= "NAME" #{ CHAR*               } [0]
Copyright  ::= "(c) " #{ CHAR*               } [0]
AUT        ::= "AUTH" #{ CHAR*               } [0]
ANNO       ::= "ANNO" #{ CHAR*               } [0]

ATAK       ::= "ATAK" #{ EGPoint*            }
RLSE       ::= "RLSE" #{ EGPoint*            }
BODY       ::= "FORM" #{ BYTE*               } [0]
```

The token "#" represents a ckSize LONG count of the following {braced} data bytes. For example, a VHDR's "#" should equal sizeof(Voice8Header). Literal items are shown in "quotes", [square bracket items] are optional, and "*" means 0 or more replications. A sometimes-needed pad byte is shown as "[0]".

Actually, the order of chunks in a FORM 8SVX is not as strict as this regular expression indicates. The property chunks VHDR, NAME, Copyright, and AUTH may actually appear in any order as long as they all precede the BODY chunk. The optional data chunks ANNO, ATAK, and RLSE don't have to precede the BODY chunk. And of course, new kinds of chunks may appear inside a FORM 8SVX in the future.

8SVX EXAMPLE

Here's a box diagram for a simple example containing a three-octave BODY.

```
+------------------------------------+
|'FORM'         362                  |
+------------------------------------+
|'8SVX'                              |
+------------------------------------+
|  +-----------------------------+   |
|  |'VHDR'               20      |   |
|  |24,16,8,10000,3,0,1.0        |   |
|  +-----------------------------+   |
|  +-----------------------------+   |
|  |'NAME'               11      |   |
|  |'bass guitar'                |   |
|  +-----------------------------+   |
|  0                                 |
|  +-----------------------------+   |
|  |'(c)'                20      |   |
|  |1985 Electronic Arts         |   |
|  +-----------------------------+   |
|  +-----------------------------+   |
|  |'BODY'               280     |   |
|  |1, 2, 3, 4, ...              |   |
|  +-----------------------------+   |
+------------------------------------+
```

The "0" after the NAME chunk is a pad byte.

STANDARDS COMMITTEE

The following people contributed to the design of this IFF standard:

Bob "Kodiak" Burns, Commodore-Amiga
R. J. Mical, Commodore-Amiga
Jerry Morrison, Electronic Arts
Greg Riker, Electronic Arts
Steve Shaw, Electronic Arts
Barry Walsh, Commodore-Amiga

FIBONACCI DELTA COMPRESSION

This is Steve Hayes' Fibonacci Delta sound compression technique. It is like the traditional delta encoding but encodes each delta in a mere four bits. The compressed data is half the size of the original data plus a two-byte overhead for the initial value. This much compression introduces some distortion; use it with discretion.

To achieve a reasonable slew rate, this algorithm looks up each stored four-bit value in a table of Fibonacci numbers. Very small deltas are encoded precisely while larger deltas are approximated. When it has to make approximations, the compressor should adjust all the values (forwards and backwards in time) for minimum overall distortion.

Here is the decompressor written in the C programming language.

```
/* Fibonacci delta encoding for sound data. */

BYTE codeToDelta[16] = {-34,-21,-13,-8,-5,-3,-2,-1,0,1,2,3,5,8,13,21};

/* Unpack Fibonacci-delta encoded data from n byte source
 * buffer into 2*n byte dest buffer, given initial data
 * value x. It returns the last data value x so you can
 * call it several times to incrementally decompress the data. */

short D1Unpack(source, n, dest, x)
    BYTE source[], dest[];
    LONG n;
    BYTE x;
    {
    BYTE d;
    LONG i, lim;

    lim = n <<<< 1;
    for (i = 0; i << lim; ++i)
           {
           /* Decode a data nibble; high nibble then low nibble. */
           d = source[i >> 1];    /* get a pair of nibbles */
           if (i & 1)             /* select low or high nibble? */
               d &= 0xf;          /* mask to get the low nibble */
           else
               d >>= 4;           /* shift to get the high nibble */
           x += codeToDelta[d];   /* add in the decoded delta */
           dest[i] = x;           /* store a 1-byte sample */
           }
    return(x);
    }

/* Unpack Fibonacci-delta encoded data from n byte
 * source buffer into 2*(n-2) byte dest buffer. Source
 * buffer has a pad byte, an 8-bit initial value, followed
 * by n-2 bytes comprising 2*(n-2) 4-bit encoded samples. */

void DUnpack(source, n, dest)
    BYTE source[], dest[];
    LONG n;
    {
           D1Unpack(source + 2, n - 2, dest, source[1]);
    }
```

```
#ifndef COMPILER_H
#define COMPILER_H
/*** compiler.h ***********************************************************/
/*  Steve Shaw                                                    1/29/86 */
/*  Portability file to handle compiler idiosyncrasies.                   */
/*  Version: Lattice 3.03 cross-compiler for the Amiga from the IBM PC.  */
/*                                                                        */
/* This software is in the public domain.                                 */
/*                                                                        */
/**************************************************************************/

#ifndef EXEC_TYPES_H
#include "exec/types.h"
#endif


/* NOTE  --  NOTE  --  NOTE  --  NOTE  --  NOTE
 * Some C compilers can handle Function Declarations with Argument Types
 * (FDwAT) like this:
 *     extern LONG Seek(BPTR, LONG, LONG)
 * while others choke unless you just say
 *     extern LONG Seek()
 *
 * Comment out the #define FDwAT if you have a compiler that chokes. */


/* #define FDwAT        COMMENTED OUT BECAUSE GREENHILLS CANT TAKE IT */


#endif COMPILER_H
```

```
#ifndef GIO_H
#define GIO_H
/*----------------------------------------------------------------------*/
/* GIO.H  defs for Generic I/O Speed Up Package.                 1/23/86 */
/* See GIOCall.C for an example of usage.                                */
/* Read not speeded-up yet.  Only one Write file buffered at a time.     */
/*                                                                       */
/* Note: The speed-up provided is ONLY significant for code such as IFF */
/* which does numerous small Writes and Seeks.                           */
/*                                                                       */
/* WARNING: If gio reports an error to you and you care what specific   */
/* Dos error was, you must call IoErr() BEFORE calling any other gio    */
/* functions.                                                            */
/*                                                                       */
/* By Jerry Morrison and Steve Shaw, Electronic Arts.                    */
/* This software is in the public domain.                                */
/*                                                                       */
/* This version for the Commodore-Amiga computer.                        */
/*                                                                       */
/*----------------------------------------------------------------------*/

/* Use this file interface in place of ALL Open,Close,Read,Write,Seek DOS
 * calls for an optional i/o speed-up via buffering.  You must use ONLY
 * these G routines for a file that is being buffered; e.g., call GClose
 * to Close the file, etc.
 * It is harmless though not necessary to use G routines for a file that
 * is not being buffered; e.g., GClose and Close are equivalent in that
 * case.
 * This Version only buffers one file at a time, and only for writing.
 * If you call GWriteDeclare for a second file before the first file
 * is GClosed, the first file becomes unbuffered.  This is harmless, no
 * data is lost, the first file is simply no longer speeded-up.
 */

/* Before compiling any modules that make G calls, or compiling gio.c,
 * you must set the GIO_ACTIVE flag below.
 *
 * To omit the speed-up code,
 *    #define GIO_ACTIVE 0
 *
 * To make the speed-up happen:
 * 1. #define GIO_ACTIVE 1
 * 2. link gio.o into your progrm
 * 3. GWriteDeclare(file, buffer, size)
 *    after GOpening the file and before doing
 *    any writing.
 * 4. ONLY use GRead, GWrite, GSeek, GClose -- do not use the DOS i/o
 *    routines directly.
 * 5. When done, do GClose.  Or to stop buffering without closing the
 *    file, do GWriteUndeclare(file).
 */
#define GIO_ACTIVE 0

#ifndef COMPILER_H
#include "iff/compiler.h"
#endif
```

```
#ifndef LIBRARIES_DOS_H
#include "libraries/dos.h"
#endif

#ifndef OFFSET_BEGINNING
#define OFFSET_BEGINNING OFFSET_BEGINING
#endif

#if GIO_ACTIVE

#ifdef FDwAT  /* Compiler handles Function Declaration with Argument Types */

/* Present for completeness in the interface.
 * "openmode" is either MODE_OLDFILE to read/write an existing file, or
 * MODE_NEWFILE to write a new file.
 * RETURNs a "file" pointer to a system-supplied structure that describes
 * the open file.  This pointer is passed in to the other routines below.*/
extern BPTR GOpen(char * /*filename*/, LONG /*openmode*/);

/* NOTE: Flushes & Frees the write buffer.
 * Returns -1 on error from Write.*/
extern LONG GClose(BPTR /*file*/);

/* Read not speeded-up yet.
 * GOpen the file, then do GReads to get successive chunks of data in
 * the file.  Assumes the system can handle any number of bytes in each
 * call, regardless of any block-structure of the device being read from.
 * When done, GClose to free any system resources associated with an
 * open file.*/
extern LONG GRead(BPTR /*file*/, BYTE * /*buffer*/, LONG /*nBytes*/);

/* Writes out any data in write buffer for file.
 * NOTE WHEN have Seeked into middle of buffer:
 * GWriteFlush causes current position to be the end of the data written.
 * -1 on error from Write.*/
extern LONG GWriteFlush(BPTR /*file*/);

/* Sets up variables to describe a write buffer for the file.*/
/* If the buffer already has data in it from an outstanding GWriteDeclare,
 * then that buffer must first be flushed.
 * RETURN -1 on error from Write for that previous buffer flush.
 * See also "GWriteUndeclare".*/
extern LONG GWriteDeclare(BPTR /*file*/, BYTE * /*buffer*/, LONG /*nBytes*/);

/* ANY PROGRAM WHICH USES "GWrite" MUST USE "GSeek" rather than "Seek"
 * TO SEEK ON A FILE BEING WRITTEN WITH "GWrite".
 * "Write" with Generic speed-up.
 * -1 on error from Write.  else returns # bytes written to disk.
 * Call GOpen, then do successive GWrites with GSeeks if required,
 * then GClose when done.  (IFF does require GSeek.)*/
extern LONG GWrite(BPTR /*file*/, BYTE * /*buffer*/, LONG /*nBytes*/);

/* "Seek" with Generic speed-up, for a file being written with GWrite.*/
/* Returns what Seek returns, which appears to be the position BEFORE
 * seeking, though the documentation says it returns the NEW position.
```

```
 * In fact, the code now explicitly returns the OLD position when
 * seeking within the buffer.
 * Eventually, will support two independent files, one being read, the
 * other being written.  Or could support even more.  Designed so is safe
 * to call even for files which aren't being buffered.*/
extern LONG GSeek(BPTR /*file*/, LONG /*position*/, LONG /*mode*/);

#else /*not FDwAT*/

extern BPTR GOpen();
extern LONG GClose();
extern LONG GRead();
extern LONG GWriteFlush();
extern LONG GWriteDeclare();
extern LONG GWrite();
extern LONG GSeek();

#endif FDwAT

#else /* not GIO_ACTIVE */

#define GOpen(filename, openmode)            Open(filename, openmode)
#define GClose(file)                         Close(file)
#define GRead(file, buffer, nBytes)          Read(file, buffer, nBytes)
#define GWriteFlush(file)                    (0)
#define GWriteDeclare(file, buffer, nBytes)  (0)
#define GWrite(file, buffer, nBytes)         Write(file, buffer, nBytes)
#define GSeek(file, position, mode)          Seek(file, position, mode)

#endif GIO_ACTIVE


/* Release the buffer for that file, flushing it to disk if it has any
 * contents.  GWriteUndeclare(NULL) to release ALL buffers.
 * Currently, only one file can be buffered at a time anyway.*/
#define GWriteUndeclare(file)  GWriteDeclare(file, NULL, 0)


#endif
```

```
#ifndef IFF_H
#define IFF_H
/*----------------------------------------------------------------------*/
/* IFF.H  Defs for IFF-85 Interchange Format Files.             1/22/86 */
/*                                                                      */
/* By Jerry Morrison and Steve Shaw, Electronic Arts.                   */
/* This software is in the public domain.                               */
/*----------------------------------------------------------------------*/

#ifndef COMPILER_H
#include "iff/compiler.h"
#endif

#ifndef LIBRARIES_DOS_H
#include "libraries/dos.h"
#endif

#ifndef OFFSET_BEGINNING
#define OFFSET_BEGINNING OFFSET_BEGINING
#endif

typedef LONG IFFP;      /* status code result from an IFF procedure */
        /* LONG, because must be type compitable with ID for GetChunkHdr.*/
        /* Note that the error codes below are not legal IDs.*/
#define IFF_OKAY   0L   /* Keep going...*/
#define END_MARK  -1L   /* as if there was a chunk at end of group.*/
#define IFF_DONE  -2L   /* clientProc returns this when it has READ enough.
                         * It means return thru all levels. File is Okay.*/
#define DOS_ERROR -3L
#define NOT_IFF   -4L   /* Not an IFF file.*/
#define NO_FILE   -5L   /* Tried to open file, DOS didn't find it.*/
#define CLIENT_ERROR -6L /* Client made invalid request, for instance, write
                         * a negative size chunk.*/
#define BAD_FORM  -7L   /* A client read proc complains about FORM semantics;
                         * e.g. valid IFF, but missing a required chunk.*/
#define SHORT_CHUNK -8L /* Client asked to IFFReadBytes more bytes than left
                         * in the chunk. Could be client bug or bad form.*/
#define BAD_IFF   -9L   /* mal-formed IFF file. [TBD] Expand this into a
                         * range of error codes.*/
#define LAST_ERROR BAD_IFF

/* This MACRO is used to RETURN immediately when a termination condition is
 * found. This is a pretty weird macro. It requires the caller to declare a
 * local "IFFP iffp" and assign it. This wouldn't work as a subroutine since
 * it returns for its caller. */
#define CheckIFFP()   { if (iffp != IFF_OKAY) return(iffp); }


/* ---------- ID -------------------------------------------------------*/

typedef LONG ID;        /* An ID is four printable ASCII chars but
                         * stored as a LONG for efficient copy & compare.*/

/* Four-character IDentifier builder.*/
#define MakeID(a,b,c,d)  ( (LONG)(a)<<24L | (LONG)(b)<<16L | (c)<<8 | (d) )
```

```
/* Standard group IDs.  A chunk with one of these IDs contains a
   SubTypeID followed by zero or more chunks.*/
#define FORM MakeID('F','O','R','M')
#define PROP MakeID('P','R','O','P')
#define LIST MakeID('L','I','S','T')
#define CAT  MakeID('C','A','T',' ')
#define FILLER MakeID(' ',' ',' ',' ')
/* The IDs "FOR1".."FOR9", "LIS1".."LIS9", & "CAT1".."CAT9" are reserved
 * for future standardization.*/

/* Pseudo-ID used internally by chunk reader and writer.*/
#define NULL_CHUNK 0L          /* No current chunk.*/


/* ---------- Chunk ----------------------------------------------------*/

/* All chunks start with a type ID and a count of the data bytes that
   follow--the chunk's "logicl size" or "data size". If that number is odd,
   a 0 pad byte is written, too. */
typedef struct {
    ID    ckID;
    LONG  ckSize;
    } ChunkHeader;

typedef struct {
    ID    ckID;
    LONG  ckSize;
    UBYTE ckData[ 1 /*REALLY: ckSize*/ ];
    } Chunk;

/* Pass ckSize = szNotYetKnown to the writer to mean "compute the size".*/
#define szNotYetKnown 0x80000001L

/* Need to know whether a value is odd so can word-align.*/
#define IS_ODD(a)    ((a) & 1)

/* This macro rounds up to an even number. */
#define WordAlign(size)    ((size+1)&~1)

/* ALL CHUNKS MUST BE PADDED TO EVEN NUMBER OF BYTES.
 * ChunkPSize computes the total "physical size" of a padded chunk from
 * its "data size" or "logical size". */
#define ChunkPSize(dataSize)   (WordAlign(dataSize) + sizeof(ChunkHeader))

/* The Grouping chunks (LIST, FORM, PROP, & CAT) contain concatenations of
 * chunks after a subtype ID that identifies the content chunks.
 * "FORM type XXXX", "LIST of FORM type XXXX", "PROPerties associated
 * with FORM type XXXX", or "conCATenation of XXXX".*/
typedef struct {
    ID    ckID;
    LONG  ckSize;       /* this ckSize includes "grpSubID".*/
    ID    grpSubID;
    } GroupHeader;

typedef struct {
    ID    ckID;
    LONG  ckSize;
    ID    grpSubID;
    UBYTE grpData[ 1 /*REALLY: ckSize-sizeof(grpSubID)*/ ];
    } GroupChunk;


/* ---------- IFF Reader ---------------------------------------------*/

/******** Routines to support a stream-oriented IFF file reader *******
 *
 * These routines handle lots of details such as error checking and skipping
 * over padding. They are also careful not to read past any containing context
 *
 * These routines ASSUME that they are the only ones reading from the file.
 * Client should check IFFP error codes. Don't press on after an error!
 * These routines try to have no side effects in the error case, except
 * partial I/O is sometimes unavoidable.
 *
 * All of these routines may return DOS_ERROR. In that case, ask DOS for the
 * specific error code.
 *
 * The overall scheme for the low level chunk reader is to open a "group read
 * context" with OpenRIFF or OpenRGroup, read the chunks with GetChunkHdr
 * (and its kin) and IFFReadBytes, and close the context with CloseRGroup.
 *
 * The overall scheme for reading an IFF file is to use ReadIFF, ReadIList,
 * and ReadICat to scan the file. See those procedures, ClientProc (below),
 * and the skeleton IFF reader. */

/* Client passes ptrs to procedures of this type to ReadIFF, which calls them
 * back to handle LISTs, FORMs, CATs, and PROPs.
 *
 * Use the GroupContext ptr when calling reader routines like GetChunkHdr.
 * Look inside the GroupContext ptr for your ClientFrame ptr. You'll
 * want to type cast it into a ptr to your containing struct to get your
 * private contextual data (stacked property settings). See below. */
#ifdef FDwAT
typedef IFFP ClientProc(struct _GroupContext *);
#else
typedef IFFP ClientProc();
#endif

/* Client's context for reading an IFF file or a group.
 * Client should actually make this the first component of a larger struct
 * (its personal stack "frame") that has a field to store each "interesting"
 * property encountered.
 * Either initialize each such field to a global default or keep a boolean
 * indicating whether you've read a property chunk into that field.
 * Your getList and getForm procs should allocate a new "frame" and copy the
 * parent frame's contents. The getProp procedure should store into the frame
 * allocated by getList for the containing LIST. */
typedef struct _ClientFrame {
   ClientProc *getList, *getProp, *getForm, *getCat;
    /* client's own data follows; place to stack property settings */
    } ClientFrame;
```

```
/* Our context for reading a group chunk. */
typedef struct _GroupContext {
    struct _GroupContext *parent; /* Containing group; NULL => whole file. */
    ClientFrame *clientFrame;     /* Reader data & client's context state. */
    BPTR file;          /* Byte-stream file handle. */
    LONG position;      /* The context's logical file position. */
    LONG bound;         /* File-absolute context bound
                         * or szNotYetKnown (writer only). */
    ChunkHeader ckHdr;  /* Current chunk header. ckHdr.ckSize = szNotYetKnown
                         * means we need to go back and set the size (writer only)
                         * See also Pseudo-IDs, above. */
    ID subtype;         /* Group's subtype ID when reading. */
    LONG bytesSoFar;    /* # bytes read/written of current chunk's data. */
    } GroupContext;

/* Computes the number of bytes not yet read from the current chunk, given
 * a group read context gc. */
#define ChunkMoreBytes(gc)  ((gc)->ckHdr.ckSize - (gc)->bytesSoFar)


/***** Low Level IFF Chunk Reader *****/

#ifdef FDwAT

/* Given an open file, open a read context spanning the whole file.
 * This is normally only called by ReadIFF.
 * This sets new->clientFrame = clientFrame.
 * ASSUME context allocated by caller but not initialized.
 * ASSUME caller doesn't deallocate the context before calling CloseRGroup.
 * NOT_IFF ERROR if the file is too short for even a chunk header.*/
extern IFFP OpenRIFF(BPTR, GroupContext *, ClientFrame *);
                 /*  file, new,            clientFrame  */

/* Open the remainder of the current chunk as a group read context.
 * This will be called just after the group's subtype ID has been read
 * (automatically by GetChunkHdr for LIST, FORM, PROP, and CAT) so the
 * remainder is a sequence of chunks.
 * This sets new->clientFrame = parent->clientFrame. The caller should repoint
 * it at a new clientFrame if opening a LIST context so it'll have a "stack
 * frame" to store PROPs for the LIST. (It's usually convenient to also
 * allocate a new Frame when you encounter FORM of the right type.)
 *
 * ASSUME new context allocated by caller but not initialized.
 * ASSUME caller doesn't deallocate the context or access the parent context
 * before calling CloseRGroup.
 * BAD_IFF ERROR if context end is odd or extends past parent. */
extern IFFP OpenRGroup(GroupContext *, GroupContext *);
                   /*  parent,         new  */

/* Close a group read context, updating its parent context.
 * After calling this, the old context may be deallocated and the parent
 * context can be accessed again. It's okay to call this particular procedure
 * after an error has occurred reading the group.
 * This always returns IFF_OKAY. */
extern IFFP CloseRGroup(GroupContext *);
```

```
                    /*  old  */

/* Skip any remaining bytes of the previous chunk and any padding, then
 * read the next chunk header into context.ckHdr.
 * If the ckID is LIST, FORM, CAT, or PROP, this automatically reads the
 * subtype ID into context->subtype.
 * Caller should dispatch on ckID (and subtype) to an appropriate handler.
 *
 * RETURNS context.ckHdr.ckID (the ID of the new chunk header); END_MARK
 * if there are no more chunks in this context; or NOT_IFF if the top level
 * file chunk isn't a FORM, LIST, or CAT; or BAD_IFF if malformed chunk, e.g.
 * ckSize is negative or too big for containing context, ckID isn't positive,
 * or we hit end-of-file.
 *
 * See also GetFChunkHdr, GetF1ChunkHdr, and GetPChunkHdr, below.*/
extern ID       GetChunkHdr(GroupContext *);
  /*  context.ckHdr.ckID    context  */

/* Read nBytes number of data bytes of current chunk. (Use OpenGroup, etc.
 * instead to read the contents of a group chunk.) You can call this several
 * times to read the data piecemeal.
 * CLIENT_ERROR if nBytes < 0. SHORT_CHUNK if nBytes > ChunkMoreBytes(context)
 * which could be due to a client bug or a chunk that's shorter than it
 * ought to be (bad form). (on either CLIENT_ERROR or SHORT_CHUNK,
 * IFFReadBytes won't read any bytes.) */
extern IFFP IFFReadBytes(GroupContext *, BYTE *, LONG);
                    /*  context,        buffer, nBytes  */


/***** IFF File Reader *****/

/* This is a noop ClientProc that you can use for a getList, getForm, getProp,
 * or getCat procedure that just skips the group. A simple reader might just
 * implement getForm, store ReadICat in the getCat field of clientFrame, and
 * use SkipGroup for the getList and getProp procs.*/
extern IFFP SkipGroup(GroupContext *);

/* IFF file reader.
 * Given an open file, allocate a group context and use it to read the FORM,
 * LIST, or CAT and its contents. The idea is to parse the file's contents,
 * and for each FORM, LIST, CAT, or PROP encountered, call the getForm,
 * getList, getCat, or getProp procedure in clientFrame, passing the
 * GroupContext ptr.
 * This is achieved with the aid of ReadIList (which your getList should
 * call) and ReadICat (which your getCat should call, if you don't just use
 * ReadICat for your getCat). If you want to handle FORMs, LISTs, and CATs
 * nested within FORMs, the getForm procedure must dispatch to getForm,
 * getList, and getCat (it can use GetF1ChunkHdr to make this easy).
 *
 * Normal return is IFF_OKAY (if whole file scanned) or IFF_DONE (if a client
 * proc said "done" first).
 * See the skeletal getList, getForm, getCat, and getProp procedures. */
extern IFFP ReadIFF(BPTR, ClientFrame *);
                /*  file, clientFrame  */

/* IFF LIST reader.
```

```
 * Your "getList" procedure should allocate a ClientFrame, copy the parent's
 * ClientFrame, and then call this procedure to do all the work.
 *
 * Normal return is IFF_OKAY (if whole LIST scanned) or IFF_DONE (if a client
 * proc said "done" first).
 * BAD_IFF ERROR if a PROP appears after a non-PROP. */
extern IFFP ReadIList(GroupContext *, ClientFrame *);
                   /*  parent,          clientFrame  */

/* IFF CAT reader.
 * Most clients can simply use this to read their CATs. If you must do extra
 * set-up work, put a ptr to your getCat procedure in the clientFrame, and
 * have that procedure call ReadICat to do the detail work.
 *
 * Normal return is IFF_OKAY (if whole CAT scanned) or IFF_DONE (if a client
 * proc said "done" first).
 * BAD_IFF ERROR if a PROP appears in the CAT. */
extern IFFP ReadICat(GroupContext *);
                  /*  parent  */

/* Call GetFChunkHdr instead of GetChunkHdr to read each chunk inside a FORM.
 * It just calls GetChunkHdr and returns BAD_IFF if it gets a PROP chunk. */
extern ID       GetFChunkHdr(GroupContext *);
  /*  context.ckHdr.ckID    context  */

/* GetF1ChunkHdr is like GetFChunkHdr, but it automatically dispatches to the
 * getForm, getList, and getCat procedure (and returns the result) if it
 * encounters a FORM, LIST, or CAT. */
extern ID       GetF1ChunkHdr(GroupContext *);
  /*  context.ckHdr.ckID    context  */

/* Call GetPChunkHdr instead of GetChunkHdr to read each chunk inside a PROP.
 * It just calls GetChunkHdr and returns BAD_IFF if it gets a group chunk. */
extern ID       GetPChunkHdr(GroupContext *);
  /*  context.ckHdr.ckID    context  */

#else /* not FDwAT */

extern IFFP OpenRIFF();
extern IFFP OpenRGroup();
extern IFFP CloseRGroup();
extern ID   GetChunkHdr();
extern IFFP IFFReadBytes();
extern IFFP SkipGroup();
extern IFFP ReadIFF();
extern IFFP ReadIList();
extern IFFP ReadICat();
extern ID   GetFChunkHdr();
extern ID   GetF1ChunkHdr();
extern ID   GetPChunkHdr();

#endif /* not FDwAT */

/* ---------- IFF Writer ---------------------------------------------------*/

/******* Routines to support a stream-oriented IFF file writer *******
```

```
 *
 * These routines will random access back to set a chunk size value when the
 * caller doesn't know it ahead of time. They'll also do things automatically
 * like padding and error checking.
 *
 * These routines ASSUME they're the only ones writing to the file.
 * Client should check IFFP error codes. Don't press on after an error!
 * These routines try to have no side effects in the error case, except that
 * partial I/O is sometimes unavoidable.
 *
 * All of these routines may return DOS_ERROR. In that case, ask DOS for the
 * specific error code.
 *
 * The overall scheme is to open an output GroupContext via OpenWIFF or
 * OpenWGroup, call either PutCk or {PutCkHdr {IFFWriteBytes}* PutCkEnd} for
 * each chunk, then use CloseWGroup to close the GroupContext.
 *
 * To write a group (LIST, FORM, PROP, or CAT), call StartWGroup, write out
 * its chunks, then call EndWGroup. StartWGroup automatically writes the
 * group header and opens a nested context for writing the contents.
 * EndWGroup closes the nested context and completes the group chunk. */


#ifdef FDwAT

/* Given a file open for output, open a write context.
 * The "limit" arg imposes a fence or upper limit on the logical file
 * position for writing data in this context. Pass in szNotYetKnown to be
 * bounded only by disk capacity.
 * ASSUME new context structure allocated by caller but not initialized.
 * ASSUME caller doesn't deallocate the context before calling CloseWGroup.
 * The caller is allowed to write out only one FORM, LIST, or CAT in this top
 * level context (see StartWGroup and PutCkHdr).
 * CLIENT_ERROR if limit is odd.*/
extern IFFP OpenWIFF(BPTR, GroupContext *, LONG);
                  /*  file, new,           limit {file position}  */

/* Start writing a group (presumably LIST, FORM, PROP, or CAT), opening a
 * nested context. The groupSize includes all nested chunks + the subtype ID.
 *
 * The subtype of a LIST or CAT is a hint at the contents' FORM type(s). Pass
 * in FILLER if it's a mixture of different kinds.
 *
 * This writes the chunk header via PutCkHdr, writes the subtype ID via
 * IFFWriteBytes, and calls OpenWGroup. The caller may then write the nested
 * chunks and finish by calling EndWGroup.
 * The OpenWGroup call sets new->clientFrame = parent->clientFrame.
 *
 * ASSUME new context structure allocated by caller but not initialized.
 * ASSUME caller doesn't deallocate the context or access the parent context
 * before calling CloseWGroup.
 * ERROR conditions: See PutCkHdr, IFFWriteBytes, OpenWGroup. */
extern IFFP StartWGroup(GroupContext *, ID, LONG, ID, GroupContext *);
                     /*  parent, groupType, groupSize, subtype, new  */

/* End a group started by StartWGroup.
```

```
 * This just calls CloseWGroup and PutCkEnd.
 * ERROR conditions: See CloseWGroup and PutCkEnd. */
extern IFFP EndWGroup(GroupContext *);
                   /*  old  */

/* Open the remainder of the current chunk as a group write context.
 * This is normally only called by StartWGroup.
 *
 * Any fixed limit to this group chunk or a containing context will impose
 * a limit on the new context.
 * This will be called just after the group's subtype ID has been written
 * so the remaining contents will be a sequence of chunks.
 * This sets new->clientFrame = parent->clientFrame.
 * ASSUME new context structure allocated by caller but not initialized.
 * ASSUME caller doesn't deallocate the context or access the parent context
 * before calling CloseWGroup.
 * CLIENT_ERROR if context end is odd or PutCkHdr wasn't called first. */
extern IFFP OpenWGroup(GroupContext *, GroupContext *);
                    /*  parent,        new  */

/* Close a write context and update its parent context.
 * This is normally only called by EndWGroup.
 *
 * If this is a top level context (created by OpenWIFF) we'll set the file's
 * EOF (end of file) but won't close the file.
 * After calling this, the old context may be deallocated and the parent
 * context can be accessed again.
 *
 * Amiga DOS Note: There's no call to set the EOF. We just position to the
 * desired end and return. Caller must Close file at that position.
 * CLIENT_ERROR if PutCkEnd wasn't called first. */
extern IFFP CloseWGroup(GroupContext *);
                     /*  old  */

/* Write a whole chunk to a GroupContext. This writes a chunk header, ckSize
 * data bytes, and (if needed) a pad byte. It also updates the GroupContext.
 * CLIENT_ERROR if ckSize == szNotYetKnown. See also PutCkHdr errors. */
extern IFFP PutCk(GroupContext *, ID,   LONG,   BYTE *);
               /*  context,       ckID, ckSize, *data  */

/* Write just a chunk header. Follow this will any number of calls to
 * IFFWriteBytes and finish with PutCkEnd.
 * If you don't yet know how big the chunk is, pass in ckSize = szNotYetKnown,
 * then PutCkEnd will set the ckSize for you later.
 * Otherwise, IFFWriteBytes and PutCkEnd will ensure that the specified
 * number of bytes get written.
 * CLIENT_ERROR if the chunk would overflow the GroupContext's bound, if
 * PutCkHdr was previously called without a matching PutCkEnd, if ckSize < 0
 * (except szNotYetKnown), if you're trying to write something other
 * than one FORM, LIST, or CAT in a top level (file level) context, or
 * if ckID <= 0 (these illegal ID values are used for error codes). */
extern IFFP PutCkHdr(GroupContext *, ID,   LONG);
                  /*  context,       ckID, ckSize  */

/* Write nBytes number of data bytes for the current chunk and update
 * GroupContext.
 * CLIENT_ERROR if this would overflow the GroupContext's limit or the
 * current chunk's ckSize, or if PutCkHdr wasn't called first, or if
 * nBytes < 0. */
extern IFFP IFFWriteBytes(GroupContext *, BYTE *, LONG);
                     /*  context,        *data,  nBytes  */

/* Complete the current chunk, write a pad byte if needed, and update
 * GroupContext.
 * If current chunk's ckSize = szNotYetKnown, this goes back and sets the
 * ckSize in the file.
 * CLIENT_ERROR if PutCkHdr wasn't called first, or if client hasn't
 * written 'ckSize' number of bytes with IFFWriteBytes. */
extern IFFP PutCkEnd(GroupContext *);
                /*  context  */

#else /* not FDwAT */

extern IFFP OpenWIFF();
extern IFFP StartWGroup();
extern IFFP EndWGroup();
extern IFFP OpenWGroup();
extern IFFP CloseWGroup();
extern IFFP PutCk();
extern IFFP PutCkHdr();
extern IFFP IFFWriteBytes();
extern IFFP PutCkEnd();

#endif /* not FDwAT */

#endif IFF_H
```

```
#ifndef ILBM_H
#define ILBM_H
/*----------------------------------------------------------------------*
 * ILBM.H  Definitions for InterLeaved BitMap raster image.     1/23/86
 *
 * By Jerry Morrison and Steve Shaw, Electronic Arts
 * This software is in the public domain.
 *
 * This version for the Commodore-Amiga computer.
 *----------------------------------------------------------------------*/
#ifndef COMPILER_H
#include "iff/compiler.h"
#endif

#ifndef GRAPHICS_GFX_H
#include "graphics/gfx.h"
#endif

#include "iff/iff.h"

#define ID_ILBM MakeID('I','L','B','M')
#define ID_BMHD MakeID('B','M','H','D')
#define ID_CMAP MakeID('C','M','A','P')
#define ID_GRAB MakeID('G','R','A','B')
#define ID_DEST MakeID('D','E','S','T')
#define ID_SPRT MakeID('S','P','R','T')
#define ID_CAMG MakeID('C','A','M','G')
#define ID_BODY MakeID('B','O','D','Y')

/* ---------- BitMapHeader ---------------------------------------------*/

typedef UBYTE Masking;          /* Choice of masking technique.*/
#define mskNone                 0
#define mskHasMask              1
#define mskHasTransparentColor  2
#define mskLasso                3

typedef UBYTE Compression;      /* Choice of compression algorithm applied tc
     * each row of the source and mask planes. "cmpByteRun1" is the byte run
     * encoding generated by Mac's PackBits. See Packer.h . */
#define cmpNone      0
#define cmpByteRun1  1

/* Aspect ratios: The proper fraction xAspect/yAspect represents the pixel
 * aspect ratio pixel_width/pixel_height.
 *
 * For the 4 Amiga display modes:
 *   320 x 200: 10/11  (these pixels are taller than they are wide)
 *   320 x 400: 20/11
 *   640 x 200:  5/11
 *   640 x 400: 10/11            */
#define x320x200Aspect 10
#define y320x200Aspect 11
#define x320x400Aspect 20
#define y320x400Aspect 11
#define x640x200Aspect  5
```

```
#define y640x200Aspect 11
#define x640x400Aspect 10
#define y640x400Aspect 11

/* A BitMapHeader is stored in a BMHD chunk. */
typedef struct {
    UWORD w, h;                 /* raster width & height in pixels */
    WORD  x, y;                 /* position for this image */
    UBYTE nPlanes;              /* # source bitplanes */
    Masking masking;            /* masking technique */
    Compression compression;    /* compression algoithm */
    UBYTE pad1;                 /* UNUSED.  For consistency, put 0 here.*/
    UWORD transparentColor;     /* transparent "color number" */
    UBYTE xAspect, yAspect;     /* aspect ratio, a rational number x/y */
    WORD  pageWidth, pageHeight;  /* source "page" size in pixels */
    } BitMapHeader;

/* RowBytes computes the number of bytes in a row, from the width in pixels.*/
#define RowBytes(w)    (((w) + 15) >> 4 << 1)


/* ---------- ColorRegister -------------------------------------------*/
/* A CMAP chunk is a packed array of ColorRegisters (3 bytes each). */
typedef struct {
    UBYTE red, green, blue;   /* MUST be UBYTEs so ">> 4" won't sign extend.*/
    } ColorRegister;

/* Use this constant instead of sizeof(ColorRegister). */
#define sizeofColorRegister  3

typedef WORD Color4;    /* Amiga RAM version of a color-register,
                         * with 4 bits each RGB in low 12 bits.*/

/* Maximum number of bitplanes in RAM. Current Amiga max w/dual playfield. */
#define MaxAmDepth 6

/* ---------- Point2D -------------------------------------------------*/
/* A Point2D is stored in a GRAB chunk. */
typedef struct {
    WORD x, y;          /* coordinates (pixels) */
    } Point2D;

/* ---------- DestMerge -----------------------------------------------*/
/* A DestMerge is stored in a DEST chunk. */
typedef struct {
    UBYTE depth;        /* # bitplanes in the original source */
    UBYTE pad1;         /* UNUSED; for consistency store 0 here */
    UWORD planePick;    /* how to scatter source bitplanes into destination */
    UWORD planeOnOff;   /* default bitplane data for planePick */
    UWORD planeMask;    /* selects which bitplanes to store into */
    } DestMerge;

/* ---------- SpritePrecedence ----------------------------------------*/
/* A SpritePrecedence is stored in a SPRT chunk. */
typedef UWORD SpritePrecedence;
```

```
/* ---------- Viewport Mode -------------------------------------------*/
/* A Commodore Amiga ViewPort->Modes is stored in a CAMG chunk. */
/* The chunk's content is declared as a LONG. */

/* ---------- CRange --------------------------------------------------*/
/* A CRange is store in a CRNG chunk. */
typedef struct {
    WORD  pad1;         /* reserved for future use; store 0 here */
    WORD  rate;         /* color cycling rate, 16384 = 60 steps/second */
    WORD  active;       /* nonzero means color cycling is turned on */
    UBYTE low, high;    /* lower and upper color registers selected */
    } CRange;

/* ---------- ILBM Writer Support Routines ----------------------------*/

/* Note: Just call PutCk to write a BMHD, GRAB, DEST, SPRT, or CAMG
 * chunk. As below. */
#define PutBMHD(context, bmHdr)  \
    PutCk(context, ID_BMHD, sizeof(BitMapHeader), (BYTE *)bmHdr)
#define PutGRAB(context, point2D)  \
    PutCk(context, ID_GRAB, sizeof(Point2D), (BYTE *)point2D)
#define PutDEST(context, destMerge)  \
    PutCk(context, ID_DEST, sizeof(DestMerge), (BYTE *)destMerge)
#define PutSPRT(context, spritePrec)  \
    PutCk(context, ID_SPRT, sizeof(SpritePrecedence), (BYTE *)spritePrec)

#ifdef FDwAT

/* Initialize a BitMapHeader record for a full-BitMap ILBM picture.
 * This gets w, h, and nPlanes from the BitMap fields BytesPerRow, Rows, and
 * Depth. It assumes you want  w = bitmap->BytesPerRow * 8 .
 * CLIENT_ERROR if bitmap->BytesPerRow isn't even, as required by ILBM format
 *
 * If (pageWidth, pageHeight) is (320, 200), (320, 400), (640, 200), or
 * (640, 400) this sets (xAspect, yAspect) based on those 4 Amiga display
 * modes. Otherwise, it sets them to (1, 1).
 *
 * After calling this, store directly into the BitMapHeader if you want to
 * override any settings, e.g. to make nPlanes smaller, to reduce w a little,
 * or to set a position (x, y) other than (0, 0).*/
extern IFFP InitBMHdr(BitMapHeader *, struct BitMap *,
                  /*  bmHdr,          bitmap  */
      int,     int,        int,         WORD,      WORD);
 /*  masking, compression, transparentColor, pageWidth, pageHeight */
 /*  Masking, Compression, UWORD -- are the desired types, but get
  *  compiler warnings if use them.                                 */

/* Output a CMAP chunk to an open FORM ILBM write context. */
extern IFFP PutCMAP(GroupContext *, WORD *,    UBYTE);
                 /*  context,       colorMap, depth  */

/* This procedure outputs a BitMap as an ILBM's BODY chunk with
 * bitplane and mask data. Compressed if bmHdr->compression == cmpByteRun1.
 * If the "mask" argument isn't NULL, it merges in the mask plane, too.
 * (A fancier routine could write a rectangular portion of an image.)
 * This gets Planes (bitplane ptrs) from "bitmap".
 *
 * CLIENT_ERROR if bitmap->Rows != bmHdr->h, or if
 * bitmap->BytesPerRow != RowBytes(bmHdr->w), or if
 * bitmap->Depth < bmHdr->nPlanes, or if bmHdr->nPlanes > MaxAmDepth, or if
 * bufsize < MaxPackedSize(bitmap->BytesPerRow), or if
 * bmHdr->compression > cmpByteRun1. */
extern IFFP PutBODY(
    GroupContext *, struct BitMap *, BYTE *, BitMapHeader *, BYTE *, LONG);
    /*  context,          bitmap,    mask,   bmHdr,         buffer, bufsize */

#else /*not FDwAT*/

extern IFFP InitBMHdr();
extern IFFP PutCMAP();
extern IFFP PutBODY();

#endif FDwAT

/* ---------- ILBM Reader Support Routines -----------------------------*/

/* Note: Just call IFFReadBytes to read a BMHD, GRAB, DEST, SPRT, or CAMG
 * chunk. As below. */
#define GetBMHD(context, bmHdr)  \
    IFFReadBytes(context, (BYTE *)bmHdr, sizeof(BitMapHeader))
#define GetGRAB(context, point2D)  \
    IFFReadBytes(context, (BYTE *)point2D, sizeof(Point2D))
#define GetDEST(context, destMerge)  \
    IFFReadBytes(context, (BYTE *)destMerge, sizeof(DestMerge))
#define GetSPRT(context, spritePrec)  \
    IFFReadBytes(context, (BYTE *)spritePrec, sizeof(SpritePrecedence))

/* GetBODY can handle a file with up to 16 planes plus a mask.*/
#define MaxSrcPlanes 16+1

#ifdef FDwAT

/* Input a CMAP chunk from an open FORM ILBM read context.
 * This converts to an Amiga color map: 4 bits each of red, green, blue packed
 * into a 16 bit color register.
 * pNColorRegs is passed in as a pointer to a UBYTE variable that holds
 * the number of ColorRegisters the caller has space to hold. GetCMAP sets
 * that variable to the number of color registers actually read.*/
extern IFFP GetCMAP(GroupContext *, WORD *,   UBYTE *);
                /*  context,       colorMap, pNColorRegs  */

/* GetBODY reads an ILBM's BODY into a client's bitmap, de-interleaving and
 * decompressing.
 *
 * Caller should first compare bmHdr dimensions (rowWords, h, nPlanes) with
 * bitmap dimensions, and consider reallocating the bitmap.
 * If file has more bitplanes than bitmap, this reads first few planes (low
 * order ones). If bitmap has more bitplanes, the last few are untouched.
 * This reads the MIN(bmHdr->h, bitmap->Rows) rows, discarding the bottom
 * part of the source or leaving the bottom part of the bitmap untouched.
 *
 * GetBODY returns CLIENT_ERROR if asked to perform a conversion it doesn't
```

```
 * handle. It only understands compression algorithms cmpNone and cmpByteRun1.
 * The filed row width (# words) must agree with bitmap->BytesPerRow.
 *
 * Caller should use bmHdr.w; GetBODY uses it only to compute the row width
 * in words. Pixels to the right of bmHdr.w are not defined.
 *
 * [TBD] In the future, GetBODY could clip the stored image horizontally or
 * fill (with transparentColor) untouched parts of the destination bitmap.
 *
 * GetBODY stores the mask plane, if any, in the buffer pointed to by mask.
 * If mask == NULL, GetBODY will skip any mask plane. If
 * (bmHdr.masking != mskHasMask) GetBODY just leaves the caller's mask alone.
 *
 * GetBODY needs a buffer large enough for two compressed rows.
 * It returns CLIENT_ERROR if bufsize < 2 * MaxPackedSize(bmHdr.rowWords * 2).
 *
 * GetBODY can handle a file with up to MaxSrcPlanes planes. It returns
 * CLIENT_ERROR if the file has more. (Could be because of a bum file, though.)
 * If GetBODY fails, itt might've modified the client's bitmap. Sorry.*/
extern IFFP GetBODY(
    GroupContext *, struct BitMap *, BYTE *, BitMapHeader *, BYTE *, LONG);
    /*  context,          bitmap,    mask,   bmHdr,         buffer, bufsize */

/* [TBD] Add routine(s) to create masks when reading ILBMs whose
 * masking != mskHasMask. For mskNone, create a rectangular mask. For
 * mskHasTransparentColor, create a mask from transparentColor. For mskLasso,
 * create an "auto mask" by filling transparent color from the edges. */

#else /*not FDwAT*/

extern IFFP GetCMAP();
extern IFFP GetBODY();

#endif FDwAT

#endif ILBM_H
```

```
/*** intuall.h ********************************************************/
/*  intuall.h, Include lots of Amiga-provided header files.     1/22/86 */
/*  Plus the portability file "iff/compiler.h" which should be tailored */
/*  for your compiler.                                                 */
/*                                                                     */
/* By Jerry Morrison and Steve Shaw, Electronic Arts.                  */
/* This software is in the public domain.                              */
/*                                                                     */
/* This version for the Commodore-Amiga computer.                      */
/*                                                                     */
/***********************************************************************/

#include "iff/compiler.h"       /* COMPILER-DEPENDENCIES */

/* Dummy definitions because some includes below are commented out.
 * This avoids 'undefined structure' warnings when compile.
 * This is safe as long as only use POINTERS to these structures.
 */

struct Region { int dummy; };
struct VSprite { int dummy; };
struct collTable { int dummy; };
struct CopList { int dummy; };
struct UCopList { int dummy; };
struct cprlist { int dummy; };
struct copinit { int dummy; };
struct TimeVal { int dummy; };

#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/libraries.h"
#include "exec/ports.h"

#include "exec/tasks.h"
#include "exec/devices.h"


#include "exec/interrupts.h"

#include "exec/io.h"
#include "exec/memory.h"
#include "exec/alerts.h"

/* ALWAYS INCLUDE GFX.H before any other amiga includes */

#include "graphics/gfx.h"
/*#include "hardware/blit.h"*/

/*****
#include "graphics/collide.h"
#include "graphics/copper.h"
#include "graphics/display.h"
#include "hardware/dmabits.h"
#include "graphics/gels.h"
***/
```

```
#include "graphics/clip.h"

#include "graphics/rastport.h"
#include "graphics/view.h"
#include "graphics/gfxbase.h"
/*#include "hardware/intbits.h"*/
#include "graphics/gfxmacros.h"

#include "graphics/layers.h"

#include "graphics/text.h"
#include "graphics/sprite.h"
/*#include "hardware/custom.h"*/

/*#include "libraries/dos.h"*/
/*#include "libraries/dosextens.h"*/

#include "devices/timer.h"
#include "devices/inputevent.h"
#include "devices/keymap.h"

#include "intuition/intuition.h"

/*#include "intuitionbase.h"*/
/*#include "intuinternal.h"*/
```

```
#ifndef PACKER_H
#define PACKER_H
/*----------------------------------------------------------------------*
 * PACKER.H  typedefs for Data-Compresser.                     1/22/86
 *
 * This module implements the run compression algorithm "cmpByteRun1"; the
 * same encoding generated by Mac's PackBits.
 *
 * By Jerry Morrison and Steve Shaw, Electronic Arts.
 * This software is in the public domain.
 *
 * This version for the Commodore-Amiga computer.
 *----------------------------------------------------------------------*/

#ifndef COMPILER_H
#include "iff/compiler.h"
#endif

/* This macro computes the worst case packed size of a "row" of bytes. */
#define MaxPackedSize(rowSize)   ( (rowSize) + ( ((rowSize)+127) >> 7 ) )


#ifdef FDwAT  /* Compiler handles Function Declaration with Argument Types */

/* Given POINTERS to POINTER variables, packs one row, updating the source
 * and destination pointers. Returns the size in bytes of the packed row.
 * ASSUMES destination buffer is large enough for the packed row.
 * See MaxPackedSize. */
extern LONG PackRow(BYTE **, BYTE **, LONG);
                 /*  pSource, pDest,   rowSize */

/* Given POINTERS to POINTER variables, unpacks one row, updating the source
 * and destination pointers until it produces dstBytes bytes (i.e., the
 * rowSize that went into PackRow).
 * If it would exceed the source's limit srcBytes or if a run would overrun
 * the destination buffer size dstBytes, it stops and returns TRUE.
 * Otherwise, it returns FALSE (no error). */
extern BOOL UnPackRow(BYTE **, BYTE **, WORD,     WORD);
                   /*  pSource, pDest,   srcBytes, dstBytes  */

#else /* not FDwAT */

extern LONG PackRow();
extern BOOL UnPackRow();

#endif /* FDwAT */

#endif
```

```
#ifndef PUTPICT_H
#define PUTPICT_H
/** putpict.h ************************************************************ */
/* PutPict().  Given a BitMap and a color map in RAM on the Amiga,         */
/* outputs as an ILBM.  See /iff/ilbm.h & /iff/ilbmw.c.        23-Jan-86 */
/*                                                                          */
/* By Jerry Morrison and Steve Shaw, Electronic Arts.                       */
/* This software is in the public domain.                                   */
/*                                                                          */
/* This version for the Commodore-Amiga computer.                           */
/*                                                                          */
/****************************************************************************/
#ifndef COMPILER_H
#include "iff/compiler.h"
#endif

#ifndef ILBM_H
#include "iff/ilbm.h"
#endif

#ifdef FDwAT

/****** IffErr *************************************************************/
/* Returns the iff error code and resets it to zero                          */
/****************************************************************************/
extern IFFP IffErr(void);

/****** PutPict ************************************************************/
/* Put a picture into an IFF file                                            */
/* Pass in mask == NULL for no mask.                                         */
/*                                                                           */
/* Buffer should be big enough for one packed scan line                      */
/* Buffer used as temporary storage to speed-up writing.                     */
/* A large buffer, say 8KB, is useful for minimizing Write and Seek calls. */
/* (See /iff/gio.h & /iff/gio.c).                                            */
/****************************************************************************/
extern BOOL PutPict(LONG, struct BitMap *, WORD,WORD, WORD *, BYTE *, LONG);
                  /* file, bm,          pageW,pageH,colorMap, buffer,bufsize */

#else /*not FDwAT*/

extern IFFP IffErr();
extern BOOL PutPict();

#endif FDwAT

#endif PUTPICT_H
```

```
#ifndef READPICT_H
#define READPICT_H
/** ReadPict.h ***************************************************************/
/*                                                                           */
/* Read an ILBM raster image file into RAM.   1/23/86.                       */
/*                                                                           */
/* By Jerry Morrison, Steve Shaw, and Steve Hayes, Electronic Arts.          */
/* This software is in the public domain.                                    */
/*                                                                           */
/* USE THIS AS AN EXAMPLE PROGRAM FOR AN IFF READER.                         */
/*                                                                           */
/* The IFF reader portion is essentially a recursive-descent parser.         */
/*****************************************************************************/

/* ILBMFrame is our "client frame" for reading FORMs ILBM in an IFF file.
 * We allocate one of these on the stack for every LIST or FORM encountered
 * in the file and use it to hold BMHD & CMAP properties. We also allocate
 * an initial one for the whole file. */
typedef struct {
   ClientFrame clientFrame;
   UBYTE foundBMHD;
   UBYTE nColorRegs;
   BitMapHeader bmHdr;
   Color4 colorMap[32 /*1<<MaxAmDepth*/ ];
   /* If you want to read any other property chunks, e.g. GRAB or CAMG, add
    * fields to this record to store them. */
   } ILBMFrame;

/** ReadPicture() ************************************************************
 *
 * Read a picture from an IFF file, given a file handle open for reading.
 * Allocates BitMap RAM by calling (*Allocator)(size).
 *
 *****************************************************************************/

typedef UBYTE *UBytePtr;

#ifdef FDwAT

typedef UBytePtr Allocator(LONG);
   /* Allocator: a memory allocation procedure which only requires a size
    * argument. (No Amiga memory flags argument.) */

extern IFFP ReadPicture(LONG, struct BitMap *, ILBMFrame *, Allocator *);
                     /*  file, bm,              iFrame,      allocator  */
   /* iFrame is the top level "client frame". */
   /* allocator is a ptr to your allocation procedure. It must always
    *   allocate in Chip memory (for bitmap data). */

   /* PS: Notice how we used two "typedef"s above to make allocator's type
    * meaningful to humans.
    * Consider the usual C style: UBYTE *(*)(), or is it (UBYTE *)(*()) ? */

#else /* not FDwAT */

typedef UBytePtr Allocator();
```

```
extern IFFP ReadPicture();

#endif

#endif READPICT_H
```

```
/** RemAlloc.h *************************************************************/
/*  ChipAlloc(), ExtAlloc(), RemAlloc(), RemFree().                        */
/*  ALLOCators which REMember the size allocated, for simpler freeing.     */
/*                                                                         */
/* Date      Who Changes                                                   */
/* --------- --- --------------------------------------------------------*/
/* 16-Jan-86 sss Created from DPaint/DAlloc.c                              */
/* 22-Jan-86 jhm Include Compiler.h                                        */
/* 25-Jan-86 sss Added ChipNoClearAlloc,ExtNoClearAlloc                    */
/*                                                                         */
/* By Jerry Morrison and Steve Shaw, Electronic Arts.                      */
/* This software is in the public domain.                                  */
/*                                                                         */
/* This version for the Commodore-Amiga computer.                          */
/*                                                                         */
/***************************************************************************/
#ifndef REM_ALLOC_H
#define REM_ALLOC_H

#ifndef COMPILER_H
#include "iff/compiler.h"
#endif


/* How these allocators work:
 * The allocator procedures get the memory from the system allocator,
 * actually allocating 4 extra bytes. We store the length of the node in
 * the first 4 bytes then return a ptr to the rest of the storage. The
 * deallocator can then find the node size and free it. */


#ifdef FDwAT

/* RemAlloc allocates a node with "size" bytes of user data.
 * Example:
 *   struct BitMap *bm;
 *   bm = (struct BitMap *)RemAlloc( sizeof(struct BitMap), ...flags... );
 */
extern UBYTE *RemAlloc(LONG, LONG);
                     /* size, flags */

/* ALLOCator that remembers size, allocates in CHIP-accessable memory.
 * Use for all data to be displayed on screen, all sound data, all data to be
 * blitted, disk buffers, or access by any other DMA channel.
 * Does clear memory being allocated.*/
extern UBYTE *ChipAlloc(LONG);
                      /* size */

/* ChipAlloc, without clearing memory.  Purpose: speed when allocate
 * large area that will be overwritten anyway.*/
extern UBYTE *ChipNoClearAlloc(LONG);

/* ALLOCator that remembers size, allocates in extended memory.
 * Does clear memory being allocated.
 * NOTICE: does NOT declare "MEMF_FAST".  This allows machines
 * lacking extended memory to allocate within chip memory,
```

```
 * assuming there is enough memory left.*/
extern UBYTE *ExtAlloc(LONG);
                     /* size */

/* ExtAlloc, without clearing memory.  Purpose: speed when allocate
 * large area that will be overwritten anyway.*/
extern UBYTE *ExtNoClearAlloc(LONG);


/* FREEs either chip or extended memory, if allocated with an allocator
 * which REMembers size allocated.
 * Safe: won't attempt to de-allocate a NULL pointer.
 * Returns NULL so caller can do
 *   p = RemFree(p);
 */
extern UBYTE *RemFree(UBYTE *);
                    /*  p  */

#else /* not FDwAT */

extern UBYTE *RemAlloc();
extern UBYTE *ChipAlloc();
extern UBYTE *ExtAlloc();
extern UBYTE *RemFree();

#endif /* FDwAT */

#endif REM_ALLOC_H
```

```
/*----------------------------------------------------------------*/
/*                                                                */
/*                        bmprintc.c                              */
/*                                                                */
/* print out a C-language representation of data for bitmap      */
/*                                                                */
/* By Jerry Morrison and Steve Shaw, Electronic Arts.            */
/* This software is in the public domain.                        */
/*                                                                */
/* This version for the Commodore-Amiga computer.                */
/*                                                                */
/*----------------------------------------------------------------*/

#include <iff/intuall.h>
#undef NULL
#include <lattice/stdio.h>

#define NO 0
#define YES 1

static BOOL doCRLF;

PrCRLF(fp) FILE *fp; {
    if (doCRLF) fprintf(fp,"%c%c",0xD,0xA); else fprintf(fp,"\n");
    }

PrintBob(bm, fp, name)
    struct BitMap *bm;
    FILE *fp;
    UBYTE *name;
    {
    UWORD *wp;

    int p,j,nb;
    int nwords = (bm->BytesPerRow/2)*bm->Rows;

    fprintf(fp,"/*----- bitmap : w = %ld, h = %ld ------ */",
       bm->BytesPerRow*8, bm->Rows);

    PrCRLF(fp);

    for (p=0; p<bm->Depth; ++p) {
        wp = (UWORD *)bm->Planes[p];
        fprintf(fp, "/*------ plane # %ld: --------*/",p);
        PrCRLF(fp);
        fprintf(fp, "UWORD %s%c[%ld] = { ", name, (p?('0'+p):' '), nwords);
        PrCRLF(fp);
        for (j = 0 ;  ; j++) {
            for (nb = 0; ; ) {
                fprintf(fp,"  0x%lx", *wp++);
                nb += 2;
                if (nb == bm->BytesPerRow) {
                    if (j == bm->Rows-1)  goto endplane;
                    else { fprintf(fp,", "); PrCRLF(fp);  break; }
                    }
                else fprintf(fp,", ");
```

```
                }
            }
        endplane: fprintf(fp," };");
        PrCRLF(fp); PrCRLF(fp);
        }
    }

PSprite(bm,fp,name,p,dohead)
    struct BitMap *bm;
    FILE *fp;
    UBYTE *name;
    int p;
    BOOL dohead;
    {
    UWORD *wp0,*wp1;
    int j,nwords;
    int wpl = bm->BytesPerRow/2;
    nwords =  2*bm->Rows + (dohead?4:0);
    wp0 = (UWORD *)bm->Planes[p];
    wp1 = (UWORD *)bm->Planes[p+1];
    fprintf(fp,"UWORD %s[%ld] = {", name, nwords);
    PrCRLF(fp);
    if (dohead){
        fprintf(fp,"  0x0000, 0x0000, /* VStart, VStop */");
        PrCRLF(fp);
        }
    for (j=0 ; j<bm->Rows; j++) {
        fprintf(fp,"  0x%lx, 0x%lx", *wp0, *wp1);
        if (dohead || (j!=bm->Rows-1)) {
            fprintf(fp, ",");
            PrCRLF(fp);
            }
        wp0 += wpl;
        wp1 += wpl;
        }
    if (dohead) fprintf(fp,"  0x0000, 0x0000 }; /* End of Sprite */");
    else fprintf(fp," };");
    PrCRLF(fp); PrCRLF(fp);
    }

static UBYTE one[] = "1";

PrintSprite(bm, fp, name, attach, dohdr)
    struct BitMap *bm; FILE *fp;
    UBYTE *name;
    BOOL attach, dohdr;
    {
    fprintf(fp,"/*----- Sprite format: h = %ld ------ */", bm->Rows);
    PrCRLF(fp);
    if (bm->Depth>1) {
        fprintf(fp, "/*--Sprite containing lower order two planes:   */");
        PrCRLF(fp);
        PSprite(bm,fp,name,0,dohdr);
        }
    if (attach && (bm->Depth > 3) ) {
        strcat(name,one);
        fprintf(fp, "/*--Sprite containing higher order two planes:   */"),
        PrCRLF(fp);
        PSprite(bm,fp,name,2,dohdr);
        }
    }

#define BOB 0
#define SPRITE 1

BMPrintCRep(bm, fp, name, fmt)
    struct BitMap *bm;  /* Contains the image data */
    FILE *fp;           /* file we will write to */
    UBYTE *name;        /* name associated with the bitmap */
    UBYTE *fmt;         /* string of characters describing output fmt*,
    {
    BOOL attach, doHdr;
    char c;
    SHORT type;
    doCRLF = NO;
    doHdr = YES;
    type = BOB;
    attach = NO;
    while ( (c=*fmt++) != 0 )
        switch (c) {
            case 'b': type = BOB; break;
            case 's': type = SPRITE; attach = NO; break;
            case 'a': type = SPRITE; attach = YES; break;
            case 'n': doHdr = NO; break;
            case 'c': doCRLF = YES; break;
            }
    switch(type) {
        case BOB: PrintBob(bm,fp,name); break;
        case SPRITE:  PrintSprite(bm,fp,name,attach,doHdr) · break;
        }
    }
```

```
/*----------------------------------------------------------------------*/
/* GIO.C  Generic I/O Speed Up Package                         1/23/86  */
/* See GIOCall.C for an example of usage.                               */
/* Read not speeded-up yet.  Only one Write file buffered at a time.    */
/* Note: The speed-up provided is ONLY significant for code such as IFF */
/* which does numerous small Writes and Seeks.                          */
/*                                                                      */
/* By Jerry Morrison and Steve Shaw, Electronic Arts                    */
/* This software is in the public domain                                */
/*                                                                      */
/* This version for the Commodore-Amiga computer.                       */
/*                                                                      */
/*----------------------------------------------------------------------*/
#include "iff/gio.h"    /* See comments here for explanation.*/

#if GIO_ACTIVE

#define local static


local BPTR wFile      = NULL;
local BYTE *wBuffer   = NULL;
local LONG wNBytes    = 0; /* buffer size in bytes.*/
local LONG wIndex     = 0; /* index of next available byte.*/
local LONG wWaterline = 0; /* Count of # bytes to be written.
                            * Different than wIndex because of GSeek.*/

/*---------- GOpen ------------------------------------------------------*/
LONG GOpen(filename, openmode)   char *filename;  LONG openmode; {
    return( Open(filename, openmode) );
    }

/*---------- GClose -----------------------------------------------------*/
LONG GClose(file)  BPTR file; {
    LONG signal = 0, signal2;
    if (file == wFile)
        signal = GWriteUndeclare(file);
    signal2 = Close(file);      /* Call Close even if trouble with write.*/
    if (signal2 < 0)
        signal = signal2;
    return( signal );
    }

/*---------- GRead ------------------------------------------------------*/
LONG GRead(file, buffer, nBytes)   BPTR file;  BYTE *buffer;  LONG nBytes; {
    LONG signal = 0;
    /* We don't yet read directly from the buffer, so flush it to disk and
     * let the DOS fetch it back. */
    if (file == wFile)
        signal = GWriteFlush(file);
    if (signal >= 0)
        signal = Read(file, buffer, nBytes);
    return( signal );
    }

/* --------- GWriteFlush ------------------------------------------------*/
```

```
LONG GWriteFlush(file)  BPTR file; {
    LONG gWrite = 0;
    if (wFile != NULL  &&  wBuffer != NULL  &&  wIndex > 0)
        gWrite = Write(wFile, wBuffer, wWaterline);
    wWaterline = wIndex = 0;    /* No matter what, make sure this happens.*/
    return( gWrite );
    }

/* ---------- GWriteDeclare --------------------------------------------*/
LONG GWriteDeclare(file, buffer, nBytes)
    BPTR file;  BYTE *buffer;  LONG nBytes; {
    LONG gWrite = GWriteFlush(wFile);  /* Finish any existing usage.*/
    if ( file==NULL  ||  (file==wFile  &&  buffer==NULL)  ||  nBytes<=3) {
        wFile = NULL;   wBuffer = NULL;     wNBytes = 0; }
    else {
        wFile = file;   wBuffer = buffer;   wNBytes = nBytes; }
    return( gWrite );
    }

/* ---------- GWrite ---------------------------------------------------*/
LONG GWrite(file, buffer, nBytes)   BPTR file;  BYTE *buffer;  LONG nBytes; {
    LONG gWrite = 0;

    if (file == wFile  &&  wBuffer != NULL) {
        if (wNBytes >= wIndex + nBytes) {
            /* Append to wBuffer.*/
            movmem(buffer, wBuffer+wIndex, nBytes);
            wIndex += nBytes;
            if (wIndex > wWaterline)
                wWaterline = wIndex;
            nBytes = 0;         /* Indicate data has been swallowed.*/
            }
        else {
            wWaterline = wIndex;     /* We are about to overwrite any
                * data above wIndex, up to at least the buffer end.*/
            gWrite = GWriteFlush(file);  /* Write data out in proper order.*/
            }
        }
    if (nBytes > 0  &&  gWrite >= 0)
        gWrite += Write(file, buffer, nBytes);
    return( gWrite );
    }

/* ---------- GSeek ----------------------------------------------------*/
LONG GSeek(file, position, mode)
    BPTR file;   LONG position;   LONG mode; {
    LONG gSeek = -2;
    LONG newWIndex = wIndex + position;

    if (file == wFile  &&  wBuffer != NULL) {
        if (mode == OFFSET_CURRENT  &&
            newWIndex >= 0  &&  newWIndex <= wWaterline) {
            gSeek = wIndex;     /* Okay; return *OLD* position */
            wIndex = newWIndex;
            }
        else {
```

```
            /* We don't even try to optimize the other cases.*/
            gSeek = GWriteFlush(file);
            if (gSeek >= 0)   gSeek = -2;  /* OK so far */
            }
        }
    if (gSeek == -2)
        gSeek = Seek(file, position, mode);
    return( gSeek );
    }

#else /* not GIO_ACTIVE */

void GIODummy() { }     /* to keep the compiler happy */

#endif GIO_ACTIVE
```

```
/*------------------------------------------------------------------------ */
/* IFFCheck.C  Print out the structure of an IFF-85 file,      1/23/86  */
/* checking for structural errors.                                        */
/*                                                                        */
/* DO NOT USE THIS AS A SKELETAL PROGRAM FOR AN IFF READER!              */
/* See ShowILBM.C for a skeletal example.                                */
/*                                                                        */
/* By Jerry Morrison and Steve Shaw, Electronic Arts.                    */
/* This software is in the public domain.                                */
/*                                                                        */
/* This version for the Commodore-Amiga computer.                        */
/*                                                                        */
/* -----------------------------------------------------------------------*/
#include "iff/iff.h"


/* ---------- IFFCheck -------------------------------------------------*/
/* [TBD] More extensive checking could be done on the IDs encountered in the
 * file. Check that the reserved IDs "FOR1".."FOR9", "LIS1".."LIS9", and
 * "CAT1".."CAT9" aren't used. Check that reserved IDs aren't used as Form
 * types. Check that all IDs are made of 4 printable characters (trailing
 * spaces ok). */

typedef struct {
    ClientFrame clientFrame;
    int levels;         /* # groups currently nested within.*/
    } Frame;

char MsgOkay[] = { "----- (IFF_OKAY) A good IFF file." };
char MsgEndMark[] = {"----- (END_MARK) How did you get this message??" };
char MsgDone[] = { "----- (IFF_DONE) How did you get this message??" };
char MsgDos[] = { "----- (DOS_ERROR) The DOS gave back an error." };
char MsgNot[] = { "----- (NOT_IFF) not an IFF file." };
char MsgNoFile[] = { "----- (NO_FILE) no such file found." };
char MsgClientError[] = {"----- (CLIENT_ERROR) IFF Checker bug."};
char MsgForm[] = { "----- (BAD_FORM) How did you get this message??" };
char MsgShort[] = { "----- (SHORT_CHUNK) How did you get this message??" };
char MsgBad[] = { "----- (BAD_IFF) a mangled IFF file." };

/* MUST GET THESE IN RIGHT ORDER!!*/
char *IFFPMessages[-(int)LAST_ERROR+1] = {
    /*IFF_OKAY*/  MsgOkay,
    /*END_MARK*/  MsgEndMark,
    /*IFF_DONE*/  MsgDone,
    /*DOS_ERROR*/ MsgDos,
    /*NOT_IFF*/   MsgNot,
    /*NO_FILE*/   MsgNoFile,
    /*CLIENT_ERROR*/ MsgClientError,
    /*BAD_FORM*/  MsgForm,
    /*SHORT_CHUNK*/  MsgShort,
    /*BAD_IFF*/   MsgBad
    };

/* FORWARD REFERENCES */
extern IFFP GetList(GroupContext *);
extern IFFP GetForm(GroupContext *);
```

```
extern IFFP GetProp(GroupContext *);
extern IFFP GetCat  (GroupContext *);

void IFFCheck(name)  char *name; {
    IFFP iffp;
    BPTR file = Open(name, MODE_OLDFILE);
    Frame frame;

    frame.levels = 0;
    frame.clientFrame.getList = GetList;
    frame.clientFrame.getForm = GetForm;
    frame.clientFrame.getProp = GetProp;
    frame.clientFrame.getCat  = GetCat ;

    printf("----- Checking file '%s' -----\n", name);
    if (file == 0)
        iffp = NO_FILE;
    else
        iffp = ReadIFF(file, (ClientFrame *)&frame);

    Close(file);
    printf("%s\n", IFFPMessages[-iffp]);
    }

main(argc, argv)   int argc;  char **argv; {
    if (argc != 1+1) {
        printf("Usage: 'iffcheck filename'\n");
        exit(0);
        }
    IFFCheck(argv[1]);
    }

/* ---------- Put... ---------------------------------------------------*/

PutLevels(count)   int count; {
    for ( ;  count > 0;  --count) {
        printf(".");
        }
    }

PutID(id)   ID id; {
    printf("%c%c%c%c",
           (char)((id>>24L) & 0x7f),
           (char)((id>>16L) & 0x7f),
           (char)((id>>8)   & 0x7f),
           (char)(id        & 0x7f) );
    }

PutN(n)   int n; {
    printf(" %d ", n);
    }

/* Put something like "...BMHD 14" or "...LIST 14 PLBM". */
PutHdr(context)  GroupContext *context; {
    PutLevels( ((Frame *)context->clientFrame)->levels );
    PutID(context->ckHdr.ckID);
    PutN(context->ckHdr.ckSize);

    if (context->subtype != NULL_CHUNK)
        PutID(context->subtype);

    printf("\n");
    }

/* ---------- AtLeaf ---------------------------------------------------*/

/* At Leaf chunk.  That is, a chunk which does NOT contain other chunks.
 * Print "ID size".*/
IFFP AtLeaf(context)  GroupContext *context; {

    PutHdr(context);
    /* A typical reader would read the chunk's contents, using the "Frame"
     * for local data, esp. shared property settings (PROP).*/
    /* IFFReadBytes(context, ...buffer, context->ckHdr->ckSize); */
    return(IFF_OKAY);
    }

/* ---------- GetList --------------------------------------------------*/
/* Handle a LIST chunk.  Print "LIST size subTypeID".
 * Then dive into it.*/
IFFP GetList(parent)  GroupContext *parent; {
    Frame newFrame;

    newFrame = *(Frame *)parent->clientFrame;  /* copy parent's frame*/
    newFrame.levels++;

    PutHdr(parent);

    return( ReadIList(parent, (ClientFrame *)&newFrame) );
    }

/* ---------- GetForm --------------------------------------------------*/
/* Handle a FORM chunk.  Print "FORM size subTypeID".
 * Then dive into it.*/
IFFP GetForm(parent)   GroupContext *parent; {
    /*CompilerBug register*/ IFFP iffp;
    GroupContext new;
    Frame newFrame;

    newFrame = *(Frame *)parent->clientFrame;  /* copy parent's frame*/
    newFrame.levels++;

    PutHdr(parent);

    iffp = OpenRGroup(parent, &new);
    CheckIFFP();
    new.clientFrame = (ClientFrame *)&newFrame;

    /* FORM reader for Checker. */
    /* LIST, FORM, PROP, CAT already handled by GetF1ChunkHdr. */
    do {if ( (iffp = GetF1ChunkHdr(&new)) > 0 )
            iffp = AtLeaf(&new);
```

```
        } while (iffp >= IFF_OKAY);

    CloseRGroup(&new);
    return(iffp == END_MARK ? IFF_OKAY : iffp);
    }

/* ---------- GetProp --------------------------------------------------*/
/* Handle a PROP chunk.  Print "PROP size subTypeID".
 * Then dive into it.*/
IFFP GetProp(listContext)  GroupContext *listContext; {
    /*CompilerBug register*/ IFFP iffp;
    GroupContext new;

    PutHdr(listContext);

    iffp = OpenRGroup(listContext, &new);
    CheckIFFP();

    /* PROP reader for Checker. */
    ((Frame *)listContext->clientFrame)->levels++;

    do {if ( (iffp = GetPChunkHdr(&new)) > 0 )
            iffp = AtLeaf(&new);
        } while (iffp >= IFF_OKAY);

    ((Frame *)listContext->clientFrame)->levels--;

    CloseRGroup(&new);
    return(iffp == END_MARK ? IFF_OKAY : iffp);
    }

/* ---------- GetCat ---------------------------------------------------*/
/* Handle a CAT chunk.  Print "CAT size subTypeID".
 * Then dive into it.*/
IFFP GetCat(parent)  GroupContext *parent;  {
    IFFP iffp;

    ((Frame *)parent->clientFrame)->levels++;

    PutHdr(parent);

    iffp = ReadICat(parent);

    ((Frame *)parent->clientFrame)->levels--;
    return(iffp);
    }
```

```
/*----------------------------------------------------------------------*
 * IFFR.C  Support routines for reading IFF-85 files.          1/23/86
 * (IFF is Interchange Format File.)
 *
 * By Jerry Morrison and Steve Shaw, Electronic Arts.
 * This software is in the public domain.
 *
 * This version for the Commodore-Amiga computer.
 *
 * Uses "gio".  Either link with gio.c, or set the GIO_ACTIVE flag to 0
 * in gio.h.
 *----------------------------------------------------------------------*/
#include "iff/gio.h"
#include "iff/iff.h"

/* ----- Private subroutine FileLength() --------------------------------*/
/* Returns the length of the file or else a negative IFFP error code
 * (NO_FILE or DOS_ERROR). AmigaDOS-specific implementation.
 * SIDE EFFECT: Thanks to AmigaDOS, we have to change the file's position
 * to find its length.
 * Now if Amiga DOS maintained fh_End, we'd just do this:
 *    fileLength = (FileHandle *)BADDR(file)->fh_End; */
LONG FileLength(file)  BPTR file;  {
    LONG fileLength = NO_FILE;

    if (file > 0)  {
        GSeek(file, 0, OFFSET_END);      /* Seek to end of file.*/
        fileLength = GSeek(file, 0, OFFSET_CURRENT);
            /* Returns position BEFORE the seek, which is #bytes in file. */
        if (fileLength < 0)
            fileLength = DOS_ERROR;      /* DOS being absurd.*/
        }

    return(fileLength);
    }

/* ---------- Read -------------------------------------------------------*/

/* ---------- OpenRIFF ---------------------------------------------------*/
IFFP OpenRIFF(file0, new0, clientFrame)
        BPTR file0;   GroupContext *new0;   ClientFrame *clientFrame; {
    register BPTR file = file0;
    register GroupContext *new = new0;
    IFFP iffp = IFF_OKAY;

    new->parent       = NULL;            /* "whole file" has no parent.*/
    new->clientFrame  = clientFrame;
    new->file         = file;
    new->position     = 0;
    new->ckHdr.ckID   = new->subtype    = NULL_CHUNK;
    new->ckHdr.ckSize = new->bytesSoFar = 0;

    /* Set new->bound and go to the file's beginning. */
    new->bound = FileLength(file);
    if (new->bound < 0)
        iffp = new->bound;                  /* File system error! */
```

```
    else if ( new->bound < sizeof(ChunkHeader) )
        iffp = NOT_IFF;                      /* Too small for an IFF file. */
    else
        GSeek(file, 0, OFFSET_BEGINNING);  /* Go to file start. */

    return(iffp);
    }

/* ---------- OpenRGroup -------------------------------------------------*/
IFFP OpenRGroup(parent0, new0)   GroupContext *parent0, *new0; {
    register GroupContext *parent = parent0;
    register GroupContext *new    = new0;
    IFFP iffp = IFF_OKAY;

    new->parent       = parent;
    new->clientFrame  = parent->clientFrame;
    new->file         = parent->file;
    new->position     = parent->position;
    new->bound        = parent->position + ChunkMoreBytes(parent);
    new->ckHdr.ckID   = new->subtype    = NULL_CHUNK;
    new->ckHdr.ckSize = new->bytesSoFar = 0;

    if ( new->bound > parent->bound  ||  IS_ODD(new->bound) )
        iffp = BAD_IFF;
    return(iffp);
    }

/* ---------- CloseRGroup ------------------------------------------------*/
IFFP CloseRGroup(context)   GroupContext *context; {
    register LONG position;

    if (context->parent == NULL) {
        }  /* Context for whole file.*/
    else {
        position = context->position;
        context->parent->bytesSoFar += position - context->parent->position;
        context->parent->position = position;
        }
    return(IFF_OKAY);
    }

/* ---------- SkipFwd ----------------------------------------------------*/
/* Skip over bytes in a context. Won't go backwards.*/
/* Updates context->position but not context->bytesSoFar.*/
/* This implementation is AmigaDOS specific.*/
IFFP SkipFwd(context, bytes)   GroupContext *context;  LONG bytes; {
    IFFP iffp = IFF_OKAY;

    if (bytes > 0) {
        if (-1 == GSeek(context->file, bytes, OFFSET_CURRENT))
            iffp = BAD_IFF;      /* Ran out of bytes before chunk complete.*/
        else
            context->position += bytes;
        }
    return(iffp);
    }
```

```
/* ---------- GetChunkHdr ----------------------------------------------*/
ID GetChunkHdr(context0)   GroupContext *context0;  {
    register GroupContext *context = context0;
    register IFFP iffp;
    LONG remaining;

    /* Skip remainder of previous chunk & padding. */
    iffp = SkipFwd(context,
        ChunkMoreBytes(context) + IS_ODD(context->ckHdr.ckSize));
    CheckIFFP();

    /* Set up to read the new header. */
    context->ckHdr.ckID = BAD_IFF;      /* Until we know it's okay, mark it BAD.*/
    context->subtype    = NULL_CHUNK;
    context->bytesSoFar = 0;

    /* Generate a psuedo-chunk if at end-of-context. */
    remaining = context->bound - context->position;
    if (remaining == 0) {
        context->ckHdr.ckSize = 0;
        context->ckHdr.ckID   = END_MARK;
        }

    /* BAD_IFF if not enough bytes in the context for a ChunkHeader.*/
    else if (sizeof(ChunkHeader) > remaining) {
        context->ckHdr.ckSize = remaining;
        }

    /* Read the chunk header (finally). */
    else {
        switch (
            GRead(context->file, (BYTE *)&context->ckHdr, sizeof(ChunkHeader))
            ) {
            case -1: return(context->ckHdr.ckID = DOS_ERROR);
            case 0:  return(context->ckHdr.ckID = BAD_IFF);
            }

        /* Check: Top level chunk must be LIST or FORM or CAT. */
        if (context->parent == NULL)
            switch(context->ckHdr.ckID) {
                case FORM:  case LIST:  case CAT:  break;
                default:    return(context->ckHdr.ckID = NOT_IFF);
                }

        /* Update the context. */
        context->position += sizeof(ChunkHeader);
        remaining         -= sizeof(ChunkHeader);

        /* Non-positive ID values are illegal and used for error codes.*/
        /* We could check for other illegal IDs...*/
        if (context->ckHdr.ckID <= 0)
             context->ckHdr.ckID = BAD_IFF;

        /* Check: ckSize negative or larger than # bytes left in context? */
        else if (context->ckHdr.ckSize < 0  ||
```

```
                 context->ckHdr.ckSize > remaining) {
            context->ckHdr.ckSize = remaining;
            context->ckHdr.ckID   = BAD_IFF;
            }

        /* Automatically read the LIST, FORM, PROP, or CAT subtype ID */
        else switch (context->ckHdr.ckID) {
            case LIST:  case FORM:  case PROP:  case CAT:  {
                iffp = IFFReadBytes(context,
                                    (BYTE *)&context->subtype,
                                    sizeof(ID));
                if (iffp != IFF_OKAY)
                    context->ckHdr.ckID = iffp;
                break; }
            }

        }
    return(context->ckHdr.ckID);
    }

/* ---------- IFFReadBytes ---------------------------------------------*/
IFFP IFFReadBytes(context, buffer, nBytes)
    GroupContext *context;   BYTE *buffer;   LONG nBytes; {
    register IFFP iffp = IFF_OKAY;

    if (nBytes < 0)
        iffp = CLIENT_ERROR;
    else if (nBytes > ChunkMoreBytes(context))
        iffp = SHORT_CHUNK;
    else if (nBytes > 0)
        switch ( GRead(context->file, buffer, nBytes) ) {
            case -1: {iffp = DOS_ERROR; break; }
            case 0:  {iffp = BAD_IFF;   break; }
            default: {
                context->position   += nBytes;
                context->bytesSoFar += nBytes;
                }
            }

    return(iffp);
    }

/* ---------- SkipGroup ------------------------------------------------*/
IFFP SkipGroup(context)  GroupContext *context;  {
    }  /* Nothing to do, thanks to GetChunkHdr */

/* ---------- ReadIFF --------------------------------------------------*/
IFFP ReadIFF(file, clientFrame)  BPTR file;  ClientFrame *clientFrame;  {
    /*CompilerBug register*/ IFFP iffp;
    GroupContext context;

    iffp = OpenRIFF(file, &context);
    context.clientFrame = clientFrame;

    if (iffp == IFF_OKAY)
        switch (iffp = GetChunkHdr(&context)) {
            case FORM: { iffp = (*clientFrame->getForm) (&context); break; }
            case LIST: { iffp = (*clientFrame->getList) (&context); break; }
            case CAT : { iffp = (*clientFrame->getCat ) (&context); break; }
            /* default: Includes IFF_DONE, BAD_IFF, NOT_IFF... */
            }

    CloseRGroup(&context);

    if (iffp > 0)               /* Make sure we don't return an ID.*/
        iffp = NOT_IFF;         /* GetChunkHdr should've caught this.*/
    return(iffp);
    }

/* ---------- ReadIList ----------------------------------------------*/
IFFP ReadIList(parent, clientFrame)
    GroupContext *parent;  ClientFrame *clientFrame; {
    GroupContext listContext;
    IFFP iffp;
    BOOL propOk = TRUE;

    iffp = OpenRGroup(parent, &listContext);
    CheckIFFP();

    /* One special case test lets us handle CATs as well as LISTs.*/
    if (parent->ckHdr.ckID == CAT)
        propOk = FALSE;
    else
        listContext.clientFrame = clientFrame;

    do {
        switch (iffp = GetChunkHdr(&listContext)) {
            case PROP: {
                if (propOk)
                    iffp = (*clientFrame->getProp) (&listContext);
                else
                    iffp = BAD_IFF;
                break;
                }
            case FORM: { iffp = (*clientFrame->getForm) (&listContext); break; }
            case LIST: { iffp = (*clientFrame->getList) (&listContext); break; }
            case CAT : { iffp = (*clientFrame->getCat ) (&listContext); break; }
            /* default: Includes END_MARK, IFF_DONE, BAD_IFF, NOT_IFF... */
            }
        if (listContext.ckHdr.ckID != PROP)
            propOk = FALSE;     /* No PROPs allowed after this point.*/
        } while (iffp == IFF_OKAY);

    CloseRGroup(&listContext);

    if (iffp > 0)       /* Only chunk types above are allowed in a LIST/CAT.*/
        iffp = BAD_IFF;
    return(iffp == END_MARK ? IFF_OKAY : iffp);
    }

/* ---------- ReadICat -----------------------------------------------*/
/* By special arrangement with the ReadIList implement'n, this is trivial.*/
```

```
IFFP ReadICat(parent)  GroupContext *parent;  {
    return( ReadIList(parent, NULL) );
    }

/* ---------- GetFChunkHdr ------------------------------------------*/
ID GetFChunkHdr(context)   GroupContext *context; {
    register ID id;

    id = GetChunkHdr(context);
    if (id == PROP)
        context->ckHdr.ckID = id = BAD_IFF;
    return(id);
    }

/* ---------- GetF1ChunkHdr -----------------------------------------*/
ID GetF1ChunkHdr(context)   GroupContext *context; {
    register ID id;
    register ClientFrame *clientFrame = context->clientFrame;

    switch (id = GetChunkHdr(context))  {
        case PROP: { id = BAD_IFF; break; }
        case FORM: { id = (*clientFrame->getForm) (context); break; }
        case LIST: { id = (*clientFrame->getList) (context); break; }
        case CAT : { id = (*clientFrame->getCat ) (context); break; }
        /* Default: let the caller handle other chunks */
        }
    return(context->ckHdr.ckID = id);
    }

/* ---------- GetPChunkHdr ------------------------------------------*/
ID GetPChunkHdr(context)   GroupContext *context; {
    register ID id;

    id = GetChunkHdr(context);
    switch (id) {
        case LIST:  case FORM:  case PROP:  case CAT:  {
            id = context->ckHdr.ckID = BAD_IFF;
            break; }
        }
    return(id);
    }
```

```
/*----------------------------------------------------------------------*
 * IFFW.C  Support routines for writing IFF-85 files.          1/23/86
 * (IFF is Interchange Format File.)
 *
 * By Jerry Morrison and Steve Shaw, Electronic Arts.
 * This software is in the public domain.
 *
 * This version for the Commodore-Amiga computer.
 *----------------------------------------------------------------------*/
#include "iff/iff.h"
#include "iff/gio.h"

/* ---------- IFF Writer -----------------------------------------------*/

/* A macro to test if a chunk size is definite, i.e. not szNotYetKnown.*/
#define Known(size)   ( (size) != szNotYetKnown )

/* Yet another weird macro to make the source code simpler...*/
#define IfIffp(expr)  {if (iffp == IFF_OKAY)  iffp = (expr);}

/* ---------- OpenWIFF -------------------------------------------------*/

IFFP OpenWIFF(file, new0, limit)  BPTR file; GroupContext *new0; LONG limit; {
    register GroupContext *new = new0;
    register IFFP iffp = IFF_OKAY;

    new->parent       = NULL;
    new->clientFrame  = NULL;
    new->file         = file;
    new->position     = 0;
    new->bound        = limit;
    new->ckHdr.ckID   = NULL_CHUNK;  /* indicates no current chunk */
    new->ckHdr.ckSize = new->bytesSoFar = 0;

    if (0 > Seek(file, 0, OFFSET_BEGINNING))    /* Go to start of the file.*/
        iffp = DOS_ERROR;
    else if ( Known(limit) && IS_ODD(limit) )
        iffp = CLIENT_ERROR;
    return(iffp);
    }

/* ---------- StartWGroup ----------------------------------------------*/
IFFP StartWGroup(parent, groupType, groupSize, subtype, new)
      GroupContext *parent, *new; ID groupType, subtype; LONG groupSize;  {
    register IFFP iffp;

    iffp = PutCkHdr(parent, groupType, groupSize);
    IfIffp( IFFWriteBytes(parent, (BYTE *)&subtype, sizeof(ID)) );
    IfIffp( OpenWGroup(parent, new) );
    return(iffp);
    }

/* ---------- OpenWGroup -----------------------------------------------*/
IFFP OpenWGroup(parent0, new0)  GroupContext *parent0, *new0; {
    register GroupContext *parent = parent0;
    register GroupContext *new    = new0;
```

```
    register LONG ckEnd;
    register IFFP iffp = IFF_OKAY;

    new->parent       = parent;
    new->clientFrame  = parent->clientFrame;
    new->file         = parent->file;
    new->position     = parent->position;
    new->bound        = parent->bound;
    new->ckHdr.ckID   = NULL_CHUNK;
    new->ckHdr.ckSize = new->bytesSoFar = 0;

    if ( Known(parent->ckHdr.ckSize) ) {
        ckEnd = new->position + ChunkMoreBytes(parent);
        if ( new->bound == szNotYetKnown || new->bound > ckEnd )
            new->bound = ckEnd;
        };

    if ( parent->ckHdr.ckID == NULL_CHUNK || /* not currently writing a chunk*/
         IS_ODD(new->position) ||
         (Known(new->bound) && IS_ODD(new->bound)) )
        iffp = CLIENT_ERROR;
    return(iffp);
    }

/* ---------- CloseWGroup ----------------------------------------------*/
IFFP CloseWGroup(old0)  GroupContext *old0; {
    register GroupContext *old = old0;
    IFFP iffp = IFF_OKAY;

    if ( old->ckHdr.ckID != NULL_CHUNK )  /* didn't close the last chunk */
        iffp = CLIENT_ERROR;
    else if ( old->parent == NULL ) {     /* top level file context */
        if (GWriteFlush(old->file) < 0)  iffp = DOS_ERROR;
        }
    else {                                /* update parent context */
        old->parent->bytesSoFar += old->position - old->parent->position;
        old->parent->position = old->position;
        };
    return(iffp);
    }

/* ---------- EndWGroup ------------------------------------------------*/
IFFP EndWGroup(old)  GroupContext *old;  {
    register GroupContext *parent = old->parent;
    register IFFP iffp;

    iffp = CloseWGroup(old);
    IfIffp( PutCkEnd(parent) );
    return(iffp);
    }

/* ---------- PutCk ----------------------------------------------------*/
IFFP PutCk(context, ckID, ckSize, data)
      GroupContext *context; ID ckID; LONG ckSize; BYTE *data; {
    register IFFP iffp = IFF_OKAY;
    if ( ckSize == szNotYetKnown )
        iffp = CLIENT_ERROR;
    IfIffp( PutCkHdr(context, ckID, ckSize) );
    IfIffp( IFFWriteBytes(context, data, ckSize) );
    IfIffp( PutCkEnd(context) );
    return(iffp);
    }

/* ---------- PutCkHdr ----------------------------------------------------*/
IFFP PutCkHdr(context0, ckID, ckSize)
      GroupContext *context0;  ID ckID;  LONG ckSize; {
    register GroupContext *context = context0;
    LONG minPSize = sizeof(ChunkHeader); /* physical chunk >= minPSize bytes*/

    /* CLIENT_ERROR if we're already inside a chunk or asked to write
     * other than one FORM, LIST, or CAT at the top level of a file */
    /* Also, non-positive ID values are illegal and used for error codes.*/
    /* (We could check for other illegal IDs...)*/
    if ( context->ckHdr.ckID != NULL_CHUNK  ||  ckID <= 0 )
        return(CLIENT_ERROR);
    else if (context->parent == NULL)  {
        switch (ckID)  {
            case FORM:  case LIST:  case CAT:  break;
            default: return(CLIENT_ERROR);
            }
        if (context->position != 0)
            return(CLIENT_ERROR);
        }

    if ( Known(ckSize) ) {
        if ( ckSize < 0 )
            return(CLIENT_ERROR);
        minPSize += ckSize;
        };
    if ( Known(context->bound)  &&
         context->position + minPSize > context->bound )
        return(CLIENT_ERROR);

    context->ckHdr.ckID   = ckID;
    context->ckHdr.ckSize = ckSize;
    context->bytesSoFar   = 0;
    if (0 >
        GWrite(context->file, (BYTE *)&context->ckHdr, sizeof(ChunkHeader))
        )
        return(DOS_ERROR);
    context->position += sizeof(ChunkHeader);
    return(IFF_OKAY);
    }

/* ---------- IFFWriteBytes ---------------------------------------------*/
IFFP IFFWriteBytes(context0, data, nBytes)
      GroupContext *context0;  BYTE *data;  LONG nBytes; {
    register GroupContext *context = context0;

    if ( context->ckHdr.ckID == NULL_CHUNK  ||  /* not in a chunk */
         nBytes < 0  ||                         /* negative nBytes */
```

```
        (Known(context->bound)  &&              /* overflow context */
          context->position + nBytes > context->bound)  ||
        (Known(context->ckHdr.ckSize)  &&       /* overflow chunk */
          context->bytesSoFar + nBytes > context->ckHdr.ckSize) )
       return(CLIENT_ERROR);

    if (0 > GWrite(context->file, data, nBytes))
        return(DOS_ERROR);

    context->bytesSoFar += nBytes;
    context->position   += nBytes;
    return(IFF_OKAY);
    }

/* ---------- PutCkEnd ---------------------------------------------------*,
IFFP PutCkEnd(context0)  GroupContext *context0; {
    register GroupContext *context = context0;
    WORD zero = 0;      /* padding source */

    if ( context->ckHdr.ckID == NULL_CHUNK )  /* not in a chunk */
        return(CLIENT_ERROR);

    if ( context->ckHdr.ckSize == szNotYetKnown ) {
        /* go back and set the chunk size to bytesSoFar */
        if ( 0 >
GSeek(context->file, -(context->bytesSoFar + sizeof(LONG)), OFFSET_CURRENT) ||
             0 >
GWrite(context->file, (BYTE *)&context->bytesSoFar, sizeof(LONG))  ||
             0 >
GSeek(context->file, context->bytesSoFar, OFFSET_CURRENT)  )
            return(DOS_ERROR);
        }
    else {  /* make sure the client wrote as many bytes as planned */
        if ( context->ckHdr.ckSize != context->bytesSoFar )
            return(CLIENT_ERROR);
        };

    /* Write a pad byte if needed to bring us up to an even boundary.
     * Since the context end must be even, and since we haven't
     * overwritten the context, if we're on an odd position there must
     * be room for a pad byte. */
    if ( IS_ODD(context->bytesSoFar) ) {
        if ( 0 > GWrite(context->file, (BYTE *)&zero, 1) )
            return(DOS_ERROR);
        context->position += 1;
        };

    context->ckHdr.ckID   = NULL_CHUNK;
    context->ckHdr.ckSize = context->bytesSoFar = 0;
    return(IFF_OKAY);
    }
```

```
/*----------------------------------------------------------------*/
/* ilbm2raw.c                                                      */
/*                    2/4/86                                       */
/* Reads in ILBM, outputs raw format, which is                     */
/* just the planes of bitmap data followed by the color map        */
/*                                                                 */
/* By Jerry Morrison and Steve Shaw, Electronic Arts.              */
/* This software is in the public domain.                          */
/*                                                                 */
/* This version for the Commodore-Amiga computer.                  */
/*                                                                 */
/*      Callable from CLI only                                     */
/*----------------------------------------------------------------*/

#include "iff/intuall.h"
#include "libraries/dos.h"
#include "libraries/dosextens.h"
#include "iff/ilbm.h"
#include "iff/readpict.h"
#include "iff/remalloc.h"

#undef NULL
#include "lattice/stdio.h"
/*------------------------------------------------------------------------*/
/*      Iff error messages                                                */
/*------------------------------------------------------------------------*/

char MsgOkay[] = { "----- (IFF_OKAY) A good IFF file." };
char MsgEndMark[] = {"----- (END_MARK) How did you get this message??" };
char MsgDone[] = { "----- (IFF_DONE) How did you get this message??" };
char MsgDos[] = { "----- (DOS_ERROR) The DOS gave back an error." };
char MsgNot[] = { "----- (NOT_IFF) not an IFF file." };
char MsgNoFile[] = { "----- (NO_FILE) no such file found." };
char MsgClientError[] = {"----- (CLIENT_ERROR) IFF Checker bug."};
char MsgForm[] = { "----- (BAD_FORM) How did you get this message??" };
char MsgShort[] = { "----- (SHORT_CHUNK) How did you get this message??" };
char MsgBad[] = { "----- (BAD_IFF) a mangled IFF file." };

/* MUST GET THESE IN RIGHT ORDER!!*/
char *IFFPMessages[-LAST_ERROR+1] = {
    /*IFF_OKAY*/  MsgOkay,
    /*END_MARK*/  MsgEndMark,
    /*IFF_DONE*/  MsgDone,
    /*DOS_ERROR*/ MsgDos,
    /*NOT_IFF*/   MsgNot,
    /*NO_FILE*/   MsgNoFile,
    /*CLIENT_ERROR*/ MsgClientError,
    /*BAD_FORM*/  MsgForm,
    /*SHORT_CHUNK*/  MsgShort,
    /*BAD_IFF*/   MsgBad
    };

LONG GfxBase;

/*----------------------------------------------------------------*/
```

```
SaveBitMap(name,bm,cols)
    UBYTE *name;
    struct BitMap *bm;
    SHORT *cols;
    {
    SHORT i;
    LONG nb,plsize;
    LONG file = Open( name, MODE_NEWFILE);
    if( file == 0 )  {
        printf(" couldn't open %s \n",name);
        return (-1);    /* couldnt open a load-file */
        }
    plsize = bm->BytesPerRow*bm->Rows;
    for (i=0; i<bm->Depth; i++) {
        nb =  Write(file, bm->Planes[i], plsize);
        if (nb<plsize) break;
        }
    Write(file, cols, (1<<bm->Depth)*2);        /* save color map */
    Close(file);
    return(0);
    }

struct BitMap bitmap = {0};

char depthString[] = "0";       /* Replaced with desired digit below.*/

ILBMFrame ilbmFrame;    /* Top level "client frame".*/

/** main() ******************************************************************/

UBYTE defSwitch[] = "b";

void main(argc, argv)  int argc;  char **argv;  {
    LONG iffp, file;
    UBYTE fname[40];
    GfxBase = (LONG)OpenLibrary("graphics.library",0);
    if (GfxBase==NULL) exit(0);

    if (argc) {
        /* Invoked via CLI.  Make a lock for current directory. */
        if (argc < 2) {
            printf("Usage from CLI: 'ilbm2raw filename '\n");
            }
        else {

            file = Open(argv[1], MODE_OLDFILE);

            if (file) {
                iffp = ReadPicture(file, &bitmap, &ilbmFrame, ChipAlloc),
                Close(file);
                if (iffp != IFF_DONE) {
                    printf(" Couldn't read file %s \n", argv[1]);
                    printf("%s\n",IFFPMessages[-iffp]);
                    }
                else {
```

```
                strcpy(fname,argv[1]);

                if (ilbmFrame.bmHdr.pageWidth > 320) {
                    if (ilbmFrame.bmHdr.pageHeight > 200)
                         strcat(fname, ".hi");
                    else strcat(fname, ".me");
                    }
                else     strcat(fname, ".lo");

                depthString[0] = '0' + bitmap.Depth;
                strcat(fname, depthString);

                printf(" Creating file %s \n", fname);
                SaveBitMap(fname, &bitmap, ilbmFrame.colorMap);
                }
            }
        else printf(" Couldn't open file: %s. \n", argv[1]);

        if (bitmap.Planes[0])  RemFree(bitmap.Planes[0]);

        printf("\n");
        }
    }
  CloseLibrary(GfxBase);
  exit(0);
  }
```

```
/*----------------------------------------------------------------------*/
/*                                                                      */
/* ILBMDump.c: reads in ILBM, prints out ascii representation,          */
/*  for including in C files.                                           */
/*                                                                      */
/* By Jerry Morrison and Steve Shaw, Electronic Arts.                   */
/* This software is in the public domain.                               */
/*                                                                      */
/* This version for the Commodore-Amiga computer.                       */
/*                                                                      */
/*  Callable from CLI ONLY                                              */
/*  Jan 31, 1986                                                        */
/*----------------------------------------------------------------------*/


#include "iff/intuall.h"
#include "libraries/dos.h"
#include "libraries/dosextens.h"
#include "iff/ilbm.h"
#include "iff/readpict.h"
#include "iff/remalloc.h"

#undef NULL
#include "lattice/stdio.h"
/*------------------------------------------------------------------------------*/
/*      Iff error messages                                                      */
/*------------------------------------------------------------------------------*/

char MsgOkay[] = { "----- (IFF_OKAY) A good IFF file." };
char MsgEndMark[] = {"----- (END_MARK) How did you get this message??" };
char MsgDone[] = { "----- (IFF_DONE) How did you get this message??" };
char MsgDos[] = { "----- (DOS_ERROR) The DOS gave back an error." };
char MsgNot[] = { "----- (NOT_IFF) not an IFF file." };
char MsgNoFile[] = { "----- (NO_FILE) no such file found." };
char MsgClientError[] = {"----- (CLIENT_ERROR) IFF Checker bug."};
char MsgForm[] = { "----- (BAD_FORM) How did you get this message??" };
char MsgShort[] = { "----- (SHORT_CHUNK) How did you get this message??" };
char MsgBad[] = { "----- (BAD_IFF) a mangled IFF file." };

/* MUST GET THESE IN RIGHT ORDER!!*/
char *IFFPMessages[-LAST_ERROR+1] = {
    /*IFF_OKAY*/  MsgOkay,
    /*END_MARK*/  MsgEndMark,
    /*IFF_DONE*/  MsgDone,
    /*DOS_ERROR*/ MsgDos,
    /*NOT_IFF*/   MsgNot,
    /*NO_FILE*/   MsgNoFile,
    /*CLIENT_ERROR*/ MsgClientError,
    /*BAD_FORM*/  MsgForm,
    /*SHORT_CHUNK*/  MsgShort,
    /*BAD_IFF*/   MsgBad
    };

/* this returns a string containing characters after the
        last '/' or ':' */
```

```
GetSuffix(to, fr) UBYTE *to, *fr; {
    int i;
    UBYTE c,*s = fr;
    for (i=0; ;i++) {
        c = *s++;
        if (c == 0) break;
        if (c == '/') fr = s;
        else if (c == ':') fr = s;
        }
    strcpy(to,fr);
    }

LONG GfxBase;
struct BitMap bitmap = {0};

ILBMFrame ilbmFrame;    /* Top level "client frame".*/

/** main() ******************************************************************/

UBYTE defSwitch[] = "b";

void main(argc, argv)  int argc;  char **argv;  {
    UBYTE *sw;
    FILE *fp;
    LONG iffp,file;
    UBYTE name[40], fname[40];
    GfxBase = (LONG)OpenLibrary("graphics.library",0);
    if (GfxBase==NULL) exit(0);

    if (argc) {
        /* Invoked via CLI.  Make a lock for current directory. */
        if (argc < 2) {
            printf("Usage from CLI: 'ILBMDump filename switch-string'\n");
            printf(" where switch-string = \n");
            printf("  <nothing> : Bob format (default)\n");
            printf("  s         : Sprite format (with header and trailer words)\n");
            printf("  sn        : Sprite format (No header and trailer words)\n");
            printf("  a         : Attached sprite (with header and trailer)\n");
            printf("  an        : Attached sprite (No header and trailer)\n");
            printf(" Add 'c' to switch list to output CR's with LF's   \n");
            }
        else {
            sw = (argc>2)? argv[2]: defSwitch;

            file = Open(argv[1], MODE_OLDFILE);

            if (file) {
                iffp = ReadPicture(file, &bitmap, &ilbmFrame, ChipAlloc);
                Close(file);
                if (iffp != IFF_DONE) {
                    printf(" Couldn't read file %s \n", argv[1]);
                    printf("%s\n",IFFPMessages[-iffp]);
                    }
                else {
                    printf(" Creating file %s.c \n",argv[1]);
                    GetSuffix(name,argv[1]);
```

```
                    strcpy(fname,argv[1]);
                    strcat(fname,".c");
                    fp = fopen(fname,"w");
                    BMPrintCRep(&bitmap,fp,name,sw);
                    fclose(fp);
                    }
                }
            else printf(" Couldn't open file: %s. \n", argv[1]);

            if (bitmap.Planes[0])  RemFree(bitmap.Planes[0]);

            printf("\n");
            }
        }
    CloseLibrary(GfxBase);
    exit(0);
    }
```

```
/*----------------------------------------------------------------------*
 * ILBMR.C  Support routines for reading ILBM files.          11/27/85
 * (IFF is Interchange Format File.)
 *
 * By Jerry Morrison and Steve Shaw, Electronic Arts.
 * This software is in the public domain.
 *
 * This version for the Commodore-Amiga computer.
 *----------------------------------------------------------------------*/
#include "iff/packer.h"
#include "iff/ilbm.h"


/* ---------- GetCMAP ------------------------------------------------*/
/* pNColorRegs is passed in as a pointer to the number of ColorRegisters
 * caller has space to hold.  GetCMAP sets to the number actually read.*/
IFFP GetCMAP(ilbmContext, colorMap, pNColorRegs)
      GroupContext *ilbmContext;  WORD *colorMap;  UBYTE *pNColorRegs;
   {
   register int nColorRegs;
   register IFFP iffp;
   ColorRegister colorReg;

   nColorRegs = ilbmContext->ckHdr.ckSize / sizeofColorRegister;
   if (*pNColorRegs < nColorRegs)   nColorRegs = *pNColorRegs;
   *pNColorRegs = nColorRegs;   /* Set to the number actually there.*/

   for ( ;  nColorRegs > 0;  --nColorRegs)  {
      iffp = IFFReadBytes(ilbmContext, (BYTE *)&colorReg,sizeofColorRegister);
      CheckIFFP();
      *colorMap++ = ( ( colorReg.red   >> 4 ) << 8 ) |
                    ( ( colorReg.green >> 4 ) << 4 ) |
                    ( ( colorReg.blue  >> 4 )      );
      }
   return(IFF_OKAY);
   }

/*---------- GetBODY ---------------------------------------------------*/
/* NOTE: This implementation could be a LOT faster if it used more of the
 * supplied buffer. It would make far fewer calls to IFFReadBytes (and
 * therefore to DOS Read) and to movemem. */
IFFP GetBODY(context, bitmap, mask, bmHdr, buffer, bufsize)
      GroupContext *context;  struct BitMap *bitmap;  BYTE *mask;
      BitMapHeader *bmHdr;  BYTE *buffer;  LONG bufsize;
   {
   register IFFP iffp;
   UBYTE srcPlaneCnt = bmHdr->nPlanes;   /* Haven't counted for mask plane yet*/
   WORD srcRowBytes = RowBytes(bmHdr->w);
   LONG bufRowBytes = MaxPackedSize(srcRowBytes);
   int nRows = bmHdr->h;
   Compression compression = bmHdr->compression;
   register int iPlane, iRow, nEmpty;
   register WORD nFilled;
   BYTE *buf, *nullDest, *nullBuf, **pDest;
   BYTE *planes[MaxSrcPlanes]; /* array of ptrs to planes & mask */
```

```
   if (compression > cmpByteRun1)
      return(CLIENT_ERROR);

   /* Complain if client asked for a conversion GetBODY doesn't handle.*/
   if ( srcRowBytes  !=  bitmap->BytesPerRow  ||
         bufsize < bufRowBytes * 2  ||
         srcPlaneCnt > MaxSrcPlanes )
      return(CLIENT_ERROR);

   if (nRows > bitmap->Rows)
      nRows = bitmap->Rows;

   /* Initialize array "planes" with bitmap ptrs; NULL in empty slots.*/
   for (iPlane = 0; iPlane < bitmap->Depth; iPlane++)
      planes[iPlane] = (BYTE *)bitmap->Planes[iPlane];
   for ( ;  iPlane < MaxSrcPlanes;  iPlane++)
      planes[iPlane] = NULL;

   /* Copy any mask plane ptr into corresponding "planes" slot.*/
   if (bmHdr->masking == mskHasMask) {
      if (mask != NULL)
         planes[srcPlaneCnt] = mask;  /* If there are more srcPlanes than
                * dstPlanes, there will be NULL plane-pointers before this.*/
      else
         planes[srcPlaneCnt] = NULL;  /* In case more dstPlanes than src.*/
      srcPlaneCnt += 1;  /* Include mask plane in count.*/
      }

   /* Set up a sink for dummy destination of rows from unwanted planes.*/
   nullDest = buffer;
   buffer  += srcRowBytes;
   bufsize -= srcRowBytes;

   /* Read the BODY contents into client's bitmap.
    * De-interleave planes and decompress rows.
    * MODIFIES: Last iteration modifies bufsize.*/
   buf = buffer + bufsize;  /* Buffer is currently empty.*/
   for (iRow = nRows; iRow > 0; iRow--)  {
      for (iPlane = 0; iPlane < srcPlaneCnt; iPlane++)  {

         pDest = &planes[iPlane];

         /* Establish a sink for any unwanted plane.*/
         if (*pDest == NULL) {
            nullBuf = nullDest;
            pDest   = &nullBuf;
            }

         /* Read in at least enough bytes to uncompress next row.*/
         nEmpty  = buf - buffer;          /* size of empty part of buffer.*/
         nFilled = bufsize - nEmpty;      /* this part has data.*/
         if (nFilled < bufRowBytes)  {
            /* Need to read more.*/

            /* Move the existing data to the front of the buffer.*/
            /* Now covers range buffer[0]..buffer[nFilled-1].*/
            movmem(buf, buffer, nFilled);  /* Could be moving 0 bytes.*/

            if (nEmpty > ChunkMoreBytes(context)) {
               /* There aren't enough bytes left to fill the buffer.*/
               nEmpty = ChunkMoreBytes(context);
               bufsize = nFilled + nEmpty;  /* heh-heh */
               }

            /* Append new data to the existing data.*/
            iffp = IFFReadBytes(context, &buffer[nFilled], nEmpty);
            CheckIFFP();

            buf     = buffer;
            nFilled = bufsize;
            nEmpty  = 0;
            }

         /* Copy uncompressed row to destination plane.*/
         if (compression == cmpNone) {
            if (nFilled < srcRowBytes)  return(BAD_FORM);
            movmem(buf, *pDest, srcRowBytes);
            buf    += srcRowBytes;
            *pDest += srcRowBytes;
            }
         else
         /* Decompress row to destination plane.*/
            if ( UnPackRow(&buf, pDest, nFilled,  srcRowBytes) )
                    /*  pSource, pDest, srcBytes, dstBytes  */
               return(BAD_FORM);
         }
      }

   return(IFF_OKAY);
   }
```

```
/*----------------------------------------------------------------------*
 * ILBMW.C  Support routines for writing ILBM files.           1/23/86
 * (IFF is Interchange Format File.)
 *
 * By Jerry Morrison and Steve Shaw, Electronic Arts.
 * This software is in the public domain.
 *
 * This version for the Commodore-Amiga computer.
 *----------------------------------------------------------------------*/
#include "iff/packer.h"
#include "iff/ilbm.h"

/*---------- InitBMHdr -------------------------------------------------*/
IFFP InitBMHdr(bmHdr0, bitmap, masking, compression, transparentColor,
            pageWidth, pageHeight)
        BitMapHeader *bmHdr0;  struct BitMap *bitmap;
        WORD masking;           /* Masking */
        WORD compression;       /* Compression */
        WORD transparentColor;  /* UWORD */
        WORD pageWidth, pageHeight;
    {
    register BitMapHeader *bmHdr = bmHdr0;
    register WORD rowBytes = bitmap->BytesPerRow;

    bmHdr->w = rowBytes << 3;
    bmHdr->h = bitmap->Rows;
    bmHdr->x = bmHdr->y = 0;    /* Default position is (0,0).*/
    bmHdr->nPlanes = bitmap->Depth;
    bmHdr->masking = masking;
    bmHdr->compression = compression;
    bmHdr->pad1 = 0;
    bmHdr->transparentColor = transparentColor;
    bmHdr->xAspect = bmHdr->yAspect = 1;
    bmHdr->pageWidth = pageWidth;
    bmHdr->pageHeight = pageHeight;

    if (pageWidth = 320)
        switch (pageHeight) {
            case 200: {bmHdr->xAspect = x320x200Aspect;
                       bmHdr->yAspect = y320x200Aspect; break;}
            case 400: {bmHdr->xAspect = x320x400Aspect;
                       bmHdr->yAspect = y320x400Aspect; break;}
            }
    else if (pageWidth = 640)
        switch (pageHeight) {
            case 200: {bmHdr->xAspect = x640x200Aspect;
                       bmHdr->yAspect = y640x200Aspect; break;}
            case 400: {bmHdr->xAspect = x640x400Aspect;
                       bmHdr->yAspect = y640x400Aspect; break;}
            }

    return( IS_ODD(rowBytes) ? CLIENT_ERROR : IFF_OKAY );
    }

/*---------- PutCMAP ---------------------------------------------------*/
IFFP PutCMAP(context, colorMap, depth)
```

```
       GroupContext *context;  WORD *colorMap;  UBYTE depth;
   {
   register LONG nColorRegs;
   IFFP iffp;
   ColorRegister colorReg;

   if (depth > MaxAmDepth)    depth = MaxAmDepth;
   nColorRegs = 1 << depth;

   iffp = PutCkHdr(context, ID_CMAP, nColorRegs * sizeofColorRegister);
   CheckIFFP();

   for ( ;  nColorRegs;  --nColorRegs)  {
      colorReg.red   = ( *colorMap >> 4 ) & 0xf0;
      colorReg.green = ( *colorMap      ) & 0xf0;
      colorReg.blue  = ( *colorMap << 4 ) & 0xf0;
      iffp = IFFWriteBytes(context, (BYTE *)&colorReg, sizeofColorRegister);
      CheckIFFP();
      ++colorMap;
      }

   iffp = PutCkEnd(context);
   return(iffp);
   }

/*---------- PutBODY ----------------------------------------------------*/
/* NOTE: This implementation could be a LOT faster if it used more of the
 * supplied buffer. It would make far fewer calls to IFFWriteBytes (and
 * therefore to DOS Write). */
IFFP PutBODY(context, bitmap, mask, bmHdr, buffer, bufsize)
      GroupContext *context;  struct BitMap *bitmap;  BYTE *mask;
      BitMapHeader *bmHdr;  BYTE *buffer;  LONG bufsize;
   {
   IFFP iffp;
   LONG rowBytes = bitmap->BytesPerRow;
   int dstDepth = bmHdr->nPlanes;
   Compression compression = bmHdr->compression;
   int planeCnt;              /* number of bit planes including mask */
   register int iPlane, iRow;
   register LONG packedRowBytes;
   BYTE *buf;
   BYTE *planes[MaxAmDepth + 1]; /* array of ptrs to planes & mask */

   if ( bufsize < MaxPackedSize(rowBytes)  ||    /* Must buffer a comprsd row*/
        compression > cmpByteRun1  ||            /* bad arg */
        bitmap->Rows != bmHdr->h   ||            /* inconsistent */
        rowBytes != RowBytes(bmHdr->w)  ||       /* inconsistent*/
        bitmap->Depth < dstDepth   ||            /* inconsistent */
        dstDepth > MaxAmDepth )                  /* too many for this routine*/
      return(CLIENT_ERROR);

   planeCnt = dstDepth + (mask == NULL ? 0 : 1);

   /* Copy the ptrs to bit & mask planes into local array "planes" */
   for (iPlane = 0; iPlane < dstDepth; iPlane++)
      planes[iPlane] = (BYTE *)bitmap->Planes[iPlane];
```

```
   if (mask != NULL)
      planes[dstDepth] = mask;

   /* Write out a BODY chunk header */
   iffp = PutCkHdr(context, ID_BODY, szNotYetKnown);
   CheckIFFP();

   /* Write out the BODY contents */
   for (iRow = bmHdr->h; iRow > 0; iRow--)  {
      for (iPlane = 0; iPlane < planeCnt; iPlane++)  {

         /* Write next row.*/
         if (compression == cmpNone) {
            iffp = IFFWriteBytes(context, planes[iPlane], rowBytes);
            planes[iPlane] += rowBytes;
            }

         /* Compress and write next row.*/
         else {
            buf = buffer;
            packedRowBytes = PackRow(&planes[iPlane], &buf, rowBytes);
            iffp = IFFWriteBytes(context, buffer, packedRowBytes);
            }

         CheckIFFP();
         }
      }

   /* Finish the chunk */
   iffp = PutCkEnd(context);
   return(iffp);
   }
```

```
/*----------------------------------------------------------------------*
 * packer.c Convert data to "cmpByteRun1" run compression.     11/15/85
 *
 * By Jerry Morrison and Steve Shaw, Electronic Arts.
 * This software is in the public domain.
 *
 *      control bytes:
 *       [0..127]   : followed by n+1 bytes of data.
 *       [-1..-127] : followed by byte to be repeated (-n)+1 times.
 *       -128       : NOOP.
 *
 * This version for the Commodore-Amiga computer.
 *----------------------------------------------------------------------*/
#include "iff/packer.h"

#define DUMP    0
#define RUN     1

#define MinRun 3
#define MaxRun 128
#define MaxDat 128

LONG putSize;
#define GetByte()       (*source++)
#define PutByte(c)      { *dest++ = (c);    ++putSize; }

char buf[256];  /* [TBD] should be 128?  on stack?*/

BYTE *PutDump(dest, nn)  BYTE *dest;  int nn; {
        int i;

        PutByte(nn-1);
        for(i = 0;  i < nn;  i++)   PutByte(buf[i]);
        return(dest);
        }

BYTE *PutRun(dest, nn, cc)    BYTE *dest;  int nn, cc; {
        PutByte(-(nn-1));
        PutByte(cc);
        return(dest);
        }

#define OutDump(nn)    dest = PutDump(dest, nn)
#define OutRun(nn,cc) dest = PutRun(dest, nn, cc)

/*----------- PackRow --------------------------------------------------*/
/* Given POINTERS TO POINTERS, packs one row, updating the source and
   destination pointers.  RETURNs count of packed bytes.*/
LONG PackRow(pSource, pDest, rowSize)
    BYTE **pSource, **pDest;   LONG rowSize; {
    BYTE *source, *dest;
    char c,lastc = '\0';
    BOOL mode = DUMP;
    short nbuf = 0;            /* number of chars in buffer */
    short rstart = 0;          /* buffer index current run starts */
```

```
    source = *pSource;
    dest = *pDest;
    putSize = 0;
    buf[0] = lastc = c = GetByte();  /* so have valid lastc */
    nbuf = 1;   rowSize--;      /* since one byte eaten.*/


    for (;  rowSize;  --rowSize) {
        buf[nbuf++] = c = GetByte();
        switch (mode) {
                case DUMP:
                        /* If the buffer is full, write the length byte,
                           then the data */
                        if (nbuf>MaxDat) {
                                OutDump(nbuf-1);
                                buf[0] = c;
                                nbuf = 1;   rstart = 0;
                                break;
                                }

                        if (c == lastc) {
                            if (nbuf-rstart >= MinRun) {
                                if (rstart > 0) OutDump(rstart);
                                mode = RUN;
                                }
                            else if (rstart == 0)
                                mode = RUN;     /* no dump in progress,
                                so can't lose by making these 2 a run.*/
                            }
                        else  rstart = nbuf-1;          /* first of run */
                        break;

                case RUN: if ( (c != lastc)|| ( nbuf-rstart > MaxRun)) {
                        /* output run */
                        OutRun(nbuf-1-rstart,lastc);
                        buf[0] = c;
                        nbuf = 1; rstart = 0;
                        mode = DUMP;
                        }
                        break;
                }

        lastc = c;
        }

    switch (mode) {
        case DUMP: OutDump(nbuf); break;
        case RUN: OutRun(nbuf-rstart,lastc); break;
        }
    *pSource = source;
    *pDest = dest;
    return(putSize);
    }
```

```
/** putpict.c ******************************************************/
/* PutPict().  Given a BitMap and a color map in RAM on the      */
/* Amiga, outputs as an ILBM.  See /iff/ilbm.h & /iff/ilbmw.c.  */
/*                  23-Jan-86                                    */
/*                                                               */
/* By Jerry Morrison and Steve Shaw, Electronic Arts.           */
/* This software is in the public domain.                       */
/*                                                               */
/* This version for the Commodore-Amiga computer.               */
/*                                                               */
/*****************************************************************/
#include "iff/intuall.h"
#include "iff/gio.h"
#include "iff/ilbm.h"
#include "iff/putpict.h"

#define MaxDepth 5
static IFFP ifferror = 0;

#define CkErr(expression)  {if (ifferror == IFF_OKAY) ifferror = (expression);}

/*****************************************************************************/
/* IffErr                                                                    */
/*                                                                           */
/* Returns the iff error code and resets it to zero                         */
/*                                                                           */
/*****************************************************************************/
IFFP IffErr()
   {
   IFFP i;
   i = ifferror;
   ifferror = 0;
   return(i);
   }

/*****************************************************************************/
/* PutPict()                                                                 */
/*                                                                           */
/* Put a picture into an IFF file                                            */
/* Pass in mask == NULL for no mask.                                         */
/*                                                                           */
/* Buffer should be big enough for one packed scan line                     */
/* Buffer used as temporary storage to speed-up writing.                    */
/* A large buffer, say 8KB, is useful for minimizing Write and Seek calls.  */
/* (See /iff/gio.h & /iff/gio.c).                                           */
/*****************************************************************************/

BOOL PutPict(file, bm, pageW, pageH, colorMap, buffer, bufsize)
      LONG file; struct BitMap *bm;
        WORD pageW,pageH;
        WORD *colorMap;
      BYTE *buffer;  LONG bufsize;
   {
   BitMapHeader bmHdr;
   GroupContext fileContext, formContext;
```

```
   ifferror = InitBMHdr(&bmHdr,
      bm,
      mskNone,
      cmpByteRun1,
      0,
      pageW,
      pageH );

/* use buffered write for speedup, if it is big-enough for both
 * PutBODY's buffer and a gio buffer.*/
#define BODY_BUFSIZE 512
   if (ifferror == IFF_OKAY  &&  bufsize > 2*BODY_BUFSIZE) {
      if (GWriteDeclare(file, buffer+BODY_BUFSIZE, bufsize-BODY_BUFSIZE) < 0)
         ifferror = DOS_ERROR;
      bufsize = BODY_BUFSIZE;
      }

   CkErr(OpenWIFF(file, &fileContext, szNotYetKnown) );
   CkErr(StartWGroup(&fileContext, FORM, szNotYetKnown, ID_ILBM, &formContext) );

   CkErr(PutCk(&formContext, ID_BMHD, sizeof(BitMapHeader), (BYTE *)&bmHdr));

   if (colorMap!=NULL)
      CkErr( PutCMAP(&formContext, colorMap, (UBYTE)bm->Depth) );
   CkErr( PutBODY(&formContext, bm, NULL, &bmHdr, buffer, bufsize) );

   CkErr( EndWGroup(&formContext) );
   CkErr( CloseWGroup(&fileContext) );
   if (GWriteUndeclare(file) < 0  &&  ifferror == IFF_OKAY)
      ifferror = DOS_ERROR;
   return( (BOOL)(ifferror != IFF_OKAY) );
   }
```

```
/** raw2ilbm.c ****************************************************************
/*  Read in a "raw" bitmap (dump of the bitplanes in a screen)  */
/*  Display it, and write it out as an ILBM file.              */
/*  23-Jan-86                                                   */
/*                                                              */
/*  Usage from CLI: 'Raw2ILBM  source dest fmt(low,med,hi)      */
/*     nplanes'                                                 */
/*  Supports the three common Amiga screen formats.             */
/*              'low' is 320x200,                               */
/*              'med' is 640x200,                               */
/*              'hi' is 640x400.                                */
/*               'nplanes' is the number of bitplanes.          */
/*  The default is low-resolution, 5 bitplanes                  */
/*              (32 colors per pixel).                          */
/*                                                              */
/* By Jerry Morrison and Steve Shaw, Electronic Arts.           */
/* This software is in the public domain.                       */
/*                                                              */
/* This version for the Commodore-Amiga computer.               */
/*                                                              */
/****************************************************************/

#include "iff/intuall.h"
#include "libraries/dos.h"
#include "libraries/dosextens.h"
#include "iff/ilbm.h"
#include "iff/putpict.h"

#define MIN(a,b)  ((a)<(b)?(a):(b))
#define MAX(a,b)  ((a)>(b)?(a):(b))

/* general usage pointers */
LONG IconBase;  /* Actually, "struct IconBase *" if you've got some ".h" file*/
struct GfxBase *GfxBase;


/* Globals for displaying an image */
struct RastPort rP;
struct RasInfo rasinfo;
struct View v = {0};
struct ViewPort vp = {0};
struct View *oldView = 0;       /* so we can restore it */

/* ---------------------------------- */
DisplayPic(bm, colorMap) struct BitMap *bm; UWORD *colorMap;  {

    oldView = GfxBase->ActiView;        /* so we can restore it */

    InitView(&v);
    InitVPort(&vp);
    v.ViewPort = &vp;
    InitRastPort(&rP);
    rP.BitMap = bm;
    rasinfo.BitMap = bm;

    /* Always show the upper left-hand corner of this picture. */
```

```
    rasinfo.RxOffset = 0;
    rasinfo.RyOffset = 0;

    vp.DWidth = bm->BytesPerRow*8;      /* Physical display WIDTH */
    vp.DHeight = bm->Rows;      /* Display height */

    /* Always display it in upper left corner of screen.*/

    if (vp.DWidth <= 320) vp.Modes = 0;
       else vp.Modes = HIRES;
    if (vp.DHeight > 200) {
       v.Modes |= LACE;
       vp.Modes |= LACE;
       }
    vp.RasInfo = &rasinfo;
    MakeVPort(&v,&vp);
    MrgCop(&v);
    LoadView(&v);       /* show the picture */
    WaitBlit();
    WaitTOF();
    if (colorMap) LoadRGB4(&vp, colorMap,(1 << bm->Depth));
    }

UnDispPict() {
    if (oldView) {
       LoadView(oldView);       /* switch back to old view */
       FreeVPortCopLists(&vp);
       FreeCprList(v.LOFCprList);
       }
    }

PrintS(msg)  char *msg; {  printf(msg);   }

void GoodBye(msg)  char *msg; {  PrintS(msg);  PrintS("\n");  exit(0);  }

struct BitMap bitmap = {0};
SHORT cmap[32];

AllocBitMap(bm) struct BitMap *bm; {
    int i;
    LONG psz = bm->BytesPerRow*bm->Rows;
    UBYTE *p = (UBYTE *)AllocMem(bm->Depth*psz, MEMF_CHIP|MEMF_PUBLIC);
    for (i=0; i<bm->Depth; i++)  {
       bm->Planes[i] = p;
       p += psz;
       }
    }

FreeBitMap(bm) struct BitMap *bm;  {
    if (bitmap.Planes[0])  {
       FreeMem(bitmap.Planes[0],
               bitmap.BytesPerRow * bitmap.Rows * bitmap.Depth);
       }
    }

BOOL LoadBitMap(file,bm,cols)
```

```
    LONG file;
    struct BitMap *bm;
    SHORT *cols;
    {
    SHORT i;
    LONG nb,plsize;
    plsize = bm->BytesPerRow*bm->Rows;
    for (i=0; i<bm->Depth; i++) {
        nb =  Read(file, bm->Planes[i], plsize);
        if (nb<plsize) BltClear(bm->Planes[i],plsize,1);
        }
    if (cols) {
        nb = Read(file, cols, (1<<bm->Depth)*2);        /* load color map */
        return( (BOOL) (nb == (1<<bm->Depth)*2) );
        }
    return((BOOL) FALSE);
    }


/** main() ******************************************************************/

UBYTE defSwitch[] = "b";

#define BUFSIZE  16000

static SHORT maxDepth[3] = {5,4,4};

void main(argc, argv)  int argc;  char **argv;  {
    SHORT fmt,depth,pwidth,pheight;
    UBYTE *buffer;
    BOOL hadCmap;
    LONG file;
    if( !(GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",0)) )
        GoodBye("No graphics.library");
    if( !(IconBase = OpenLibrary("icon.library",0)) )
        GoodBye("No icon.library");
    if (argc) {
        if (argc < 3) {
            printf(
"Usage from CLI: 'Raw2ILBM  source dest fmt(low,med,hi) nplanes'\n");
            goto bailout;
            }
        fmt = 0;
        depth = 5;
        if (argc>3)
            switch(*argv[3]) {
                case 'l': fmt = 0; break;
                case 'm': fmt = 1; break;
                case 'h': fmt = 2; break;
                }
        if (argc>4) depth = *argv[4]-'0';
        depth = MAX(1, MIN(maxDepth[fmt],depth));
        pwidth = fmt? 640: 320;
        pheight = (fmt>1)? 400: 200;
        InitBitMap(&bitmap, depth, pwidth, pheight);
        AllocBitMap(&bitmap);
```

```
        file = Open(argv[1], MODE_OLDFILE);

        if (file)  {
            DisplayPic(&bitmap,NULL);
            hadCmap = LoadBitMap(file,&bitmap, cmap);
            if (hadCmap) LoadRGB4(&vp, cmap, 1<<bitmap.Depth);
            Close(file);
            file = Open(argv[2], MODE_NEWFILE);
            buffer = (UBYTE *)AllocMem(BUFSIZE, MEMF_CHIP|MEMF_PUBLIC);
            PutPict(file, &bitmap, pwidth, pheight,
                hadCmap? cmap: NULL, buffer, BUFSIZE),
            Close(file);
            FreeMem(buffer,BUFSIZE);
            }
        else printf(" Couldn't open file '%s' \n",argv[2]),
        }

    UnDispPict();
    FreeBitMap(&bitmap);

    bailout:
        CloseLibrary(GfxBase);
        CloseLibrary(IconBase);
        exit(0);
    }
```

```
/** ReadPict.c ****************************************************************
 *
 * Read an ILBM raster image file.                                 23-Jan-86.
 *
 * By Jerry Morrison, Steve Shaw, and Steve Hayes, Electronic Arts.
 * This software is in the public domain.
 *
 * USE THIS AS AN EXAMPLE PROGRAM FOR AN IFF READER.
 *
 * The IFF reader portion is essentially a recursive-descent parser.
 ******************************************************************************/

#define LOCAL   static

#include "iff/intuall.h"
#include "libraries/dos.h"
#include "libraries/dosextens.h"
#include "iff/ilbm.h"
#include "iff/readpict.h"

/* This example's max number of planes in a bitmap. Could use MaxAmDepth. */
#define EXDepth 5
#define maxColorReg  (1<<EXDepth)
#define MIN(a,b)  ((a)<(b)?(a):(b))

#define SafeFreeMem(p,q)  {if(p)FreeMem(p,q);}

/* Define the size of a temporary buffer used in unscrambling the ILBM rows.*/
#define bufSz 512

/*------------ ILBM reader -----------------------------------------------*/
/* ILBMFrame is our "client frame" for reading FORMs ILBM in an IFF file.
 * We allocate one of these on the stack for every LIST or FORM encountered
 * in the file and use it to hold BMHD & CMAP properties. We also allocate
 * an initial one for the whole file.
 * We allocate a new GroupContext (and initialize it by OpenRIFF or
 * OpenRGroup) for every group (FORM, CAT, LIST, or PROP) encountered. It's
 * just a context for reading (nested) chunks.
 *
 * If we were to scan the entire example file outlined below:
 *    reading            proc(s)              new                 new
 *
 * --whole file--     ReadPicture+ReadIFF   GroupContext        ILBMFrame
 * CAT                ReadICat                GroupContext
 *   LIST             GetLiILBM+ReadIList       GroupContext        ILBMFrame
 *     PROP ILBM      GetPrILBM                   GroupContext
 *       CMAP         GetCMAP
 *       BMHD         GetBMHD
 *     FORM ILBM      GetFoILBM                   GroupContext        ILBMFrame
 *       BODY         GetBODY
 *     FORM ILBM      GetFoILBM                   GroupContext        ILBMFrame
 *       BODY         GetBODY
 *   FORM ILBM        GetFoILBM                 GroupContext        ILBMFrame
 */

/* NOTE: For a small version of this program, set Fancy to 0.
```

```
 * That'll compile a program that reads a single FORM ILBM in a file, which
 * is what DeluxePaint produces. It'll skip all LISTs and PROPs in the input
 * file. It will, however, look inside a CAT for a FORM ILBM.
 * That's suitable for 90% of the uses.
 *
 * For a fancier version that handles LISTs and PROPs, set Fancy to 1.
 * That'll compile a program that dives into a LIST, if present, to read
 * the first FORM ILBM. E.g. a DeluxePrint library of images is a LIST of
 * FORMs ILBM.
 *
 * For an even fancier version, set Fancy to 2. That'll compile a program
 * that dives into non-ILBM FORMs, if present, looking for a nested FORM ILBM.
 * E.g. a DeluxeVideo C.S. animated object file is a FORM ANBM containing a
 * FORM ILBM for each image frame. */
#define Fancy  0

/* Global access to client-provided pointers.*/
LOCAL Allocator *gAllocator = NULL;
LOCAL struct BitMap *gBM = NULL;          /* client's bitmap.*/
LOCAL ILBMFrame *giFrame = NULL;          /* "client frame".*/

/** GetFoILBM() ***************************************************************
 *
 * Called via ReadPicture to han+le every FORM encountered in an IFF file.
 * Reads FORMs ILBM and skips all others.
 * Inside a FORM ILBM, it stops once it reads a BODY. It complains if it
 * finds no BODY or if it has no BMHD to decode the BODY.
 *
 * Once we find a BODY chunk, we'll allocate the BitMap and read the image.
 *
 ******************************************************************************/
LOCAL BYTE bodyBuffer[bufSz];
IFFP GetFoILBM(parent)  GroupContext *parent;  {
   /*compilerBug register*/ IFFP iffp;
   GroupContext formContext;
   ILBMFrame ilbmFrame;         /* only used for non-clientFrame fields.*/
   register int i;
   LONG plsize; /* Plane size in bytes. */
   int nPlanes; /* number of planes in our display image */

    /* Handle a non-ILBM FORM. */
    if (parent->subtype != ID_ILBM) {
#if Fancy >= 2
        /* Open a non-ILBM FORM and recursively scan it for ILBMs.*/
        iffp = OpenRGroup(parent, &formContext);
        CheckIFFP();
        do {
            iffp = GetF1ChunkHdr(&formContext);
            } while (iffp >= IFF_OKAY);
        if (iffp == END_MARK)
            iffp = IFF_OKAY;    /* then continue scanning the file */
        CloseRGroup(&formContext);
        return(iffp);
#else
        return(IFF_OKAY); /* Just skip this FORM and keep scanning the file.*/
#endif
```

```
      }

   ilbmFrame = *(ILBMFrame *)parent->clientFrame;
   iffp = OpenRGroup(parent, &formContext);
   CheckIFFP();

   do switch (iffp = GetFChunkHdr(&formContext)) {
      case ID_BMHD: {
        ilbmFrame.foundBMHD = TRUE;
        iffp = GetBMHD(&formContext, &ilbmFrame.bmHdr);
        break; }
      case ID_CMAP: {
        ilbmFrame.nColorRegs = maxColorReg;  /* we have room for this many */
        iffp = GetCMAP(
           &formContext, (WORD *)&ilbmFrame.colorMap[0], &ilbmFrame.nColorRegs),
                       /* was &ilbmFrame.colorMap, (fixed) robp. */
        break; }
      case ID_BODY: {
        if (!ilbmFrame.foundBMHD)  return(BAD_FORM);   /* No BMHD chunk! */

        nPlanes = MIN(ilbmFrame.bmHdr.nPlanes, EXDepth);
        InitBitMap(
           gBM,
           nPlanes,
           ilbmFrame.bmHdr.w,
           ilbmFrame.bmHdr.h);
        plsize = RowBytes(ilbmFrame.bmHdr.w) * ilbmFrame.bmHdr.h;
        /* Allocate all planes contiguously.  Not really necessary,
         * but it avoids writing code to back-out if only enough memory
         * for some of the planes.
         * WARNING: Don't change this without changing the code that
         * Frees these planes.
         */
        if (gBM->Planes[0] =
               (PLANEPTR)(*gAllocator)(nPlanes * plsize))
           {
           for (i = 1; i < nPlanes; i++)
               gBM->Planes[i] = (PLANEPTR) gBM->Planes[0] + plsize*i;
           iffp = GetBODY(
               &formContext,
               gBM,
               NULL,
               &ilbmFrame.bmHdr,
               bodyBuffer,
               bufSz);
           if (iffp == IFF_OKAY) iffp = IFF_DONE;      /* Eureka */
           *giFrame = ilbmFrame;  /* Copy fields to client's frame.*/
           }
        else
           iffp = CLIENT_ERROR;        /* not enough RAM for the bitmap */
        break; }
     case END_MARK: { iffp = BAD_FORM; break; } /* No BODY chunk! */
     } while (iffp >= IFF_OKAY);  /* loop if valid ID of ignored chunk or a
                           * subroutine returned IFF_OKAY (no errors).*/

  if (iffp != IFF_DONE)  return(iffp);
```

```
    /* If we get this far, there were no errors. */
    CloseRGroup(&formContext);
    return(iffp);
    }

/** Notes on extending GetFoILBM ****************************************
 *
 * To read more kinds of chunks, just add clauses to the switch statement.
 * To read more kinds of property chunks (GRAB, CAMG, etc.) add clauses to
 * the switch statement in GetPrILBM, too.
 *
 * To read a FORM type that contains a variable number of data chunks--e.g.
 * a FORM FTXT with any number of CHRS chunks--replace the ID_BODY case with
 * an ID_CHRS case that doesn't set iffp = IFF_DONE, and make the END_MARK
 * case do whatever cleanup you need.
 *
 ***********************************************************************/

/** GetPrILBM() ************************************************************
 *
 * Called via ReadPicture to handle every PROP encountered in an IFF file.
 * Reads PROPs ILBM and skips all others.
 *
 ***********************************************************************/
#if Fancy
IFFP GetPrILBM(parent)  GroupContext *parent;  {
    /*compilerBug register*/ IFFP iffp;
    GroupContext propContext;
    ILBMFrame *ilbmFrame = (ILBMFrame *)parent->clientFrame;

    if (parent->subtype != ID_ILBM)
       return(IFF_OKAY); /* just continue scaning the file */

    iffp = OpenRGroup(parent, &propContext);
    CheckIFFP();

    do switch (iffp = GetPChunkHdr(&propContext)) {
       case ID_BMHD: {
         ilbmFrame->foundBMHD = TRUE;
         iffp = GetBMHD(&propContext, &ilbmFrame->bmHdr);
         break; }
       case ID_CMAP: {
         ilbmFrame->nColorRegs = maxColorReg; /* we have room for this many */
         iffp = GetCMAP(
           &propContext, (WORD *)&ilbmFrame->colorMap, &ilbmFrame->nColorRegs);
         break; }
       } while (iffp >= IFF_OKAY);  /* loop if valid ID of ignored chunk or a
                           * subroutine returned IFF_OKAY (no errors).*/

    CloseRGroup(&propContext);
    return(iffp == END_MARK ? IFF_OKAY : iffp);
    }
#endif

/** GetLiILBM() ************************************************************
```

```
 *
 * Called via ReadPicture to handle every LIST encountered in an IFF file.
 *
 ***************************************************************************/
#if Fancy
IFFP GetLiILBM(parent)  GroupContext *parent;  {
    ILBMFrame newFrame; /* allocate a new Frame */

    newFrame = *(ILBMFrame *)parent->clientFrame;  /* copy parent frame */

    return( ReadIList(parent, (ClientFrame *)&newFrame) );
    }
#endif

/** ReadPicture() ***********************************************************/
IFFP ReadPicture(file, bm, iFrame, allocator)
   LONG file;
   struct BitMap *bm;
   ILBMFrame *iFrame;   /* Top level "client frame".*/

        /* **** ERROR IN SOURCE CODE, WAS jFrame, now iFrame */
        /* fixed */

   Allocator *allocator;
   {
   IFFP iffp = IFF_OKAY;

#if Fancy
   iFrame->clientFrame.getList = GetLiILBM;
   iFrame->clientFrame.getProp = GetPrILBM;
#else
   iFrame->clientFrame.getList = SkipGroup;
   iFrame->clientFrame.getProp = SkipGroup;
#endif
   iFrame->clientFrame.getForm = GetFoILBM;
   iFrame->clientFrame.getCat  = ReadICat ;

   /* Initialize the top-level client frame's property settings to the
    * program-wide defaults. This example just records that we haven't read
    * any BMHD property or CMAP color registers yet. For the color map, that
    * means the default is to leave the machine's color registers alone.
    * If you want to read a property like GRAB, init it here to (0, 0). */
   iFrame->foundBMHD  = FALSE;
   iFrame->nColorRegs = 0;

   gAllocator = allocator;
   gBM = bm;
   giFrame = iFrame;
  /* Store a pointer to the client's frame in a global variable so that
   * GetFoILBM can update client's frame when done.  Why do we have so
   * many frames & frame pointers floating around causing confusion?
   * Because IFF supports PROPs which apply to all FORMs in a LIST,
   * unless a given FORM overrides some property.
   * When you write code to read several FORMs,
   * it is ssential to maintain a frame at each level of the syntax
   * so that the properties for the LIST don't get overwritten by any
```

```
   * properties specified by individual FORMs.
   * We decided it was best to put that complexity into this one-FORM example,
   * so that those who need it later will have a useful starting place.
   */

   iffp = ReadIFF(file, (ClientFrame *)iFrame);
   return(iffp);
   `
```

```
/** RemAlloc.c ***************************************************************/
/*  ChipAlloc(), ExtAlloc(), RemAlloc(), RemFree().                          */
/*  ALLOCators which REMember the size allocated, for simpler freeing.       */
/*                                                                           */
/* Date      Who Changes                                                     */
/* --------- --- ----------------------------------------------------------*/
/* 16-Jan-86 sss Created from DPaint/DAlloc.c                                */
/* 23-Jan-86 jhm Include Compiler.h, check for size > 0 in RemAlloc.         */
/* 25-Jan-86 sss Added ChipNoClearAlloc,ExtNoClearAlloc                      */
/*                                                                           */
/* By Jerry Morrison and Steve Shaw, Electronic Arts.                        */
/* This software is in the public domain.                                    */
/*                                                                           */
/* This version for the Commodore-Amiga computer.                            */
/*                                                                           */
/*****************************************************************************/
#ifndef COMPILER_H
#include "iff/compiler.h"
#endif

#include "exec/nodes.h"
#include "exec/memory.h"
#include "iff/remalloc.h"

/** RemAlloc ****************************************************************/
UBYTE *RemAlloc(size,flags) LONG size, flags; {
    register LONG *p = NULL;    /* (LONG *) for the sake of p++, below */
    register LONG asize = size+4;
    if (size > 0)
        p = (LONG *)AllocMem(asize,flags);
    if (p != NULL)
        *p++ = asize;   /* post-bump p to point at clients area*/
    return((UBYTE *)p);
    }

/** ChipAlloc ***************************************************************/
UBYTE *ChipAlloc(size) LONG size; {
    return(RemAlloc(size, MEMF_CLEAR|MEMF_PUBLIC|MEMF_CHIP));
    }

/** ChipNoClearAlloc ********************************************************/
UBYTE *ChipNoClearAlloc(size) LONG size; {
    return(RemAlloc(size, MEMF_PUBLIC|MEMF_CHIP));
    }

/** ExtAlloc ****************************************************************/
UBYTE *ExtAlloc(size) LONG size; {
    return(RemAlloc(size, MEMF_CLEAR|MEMF_PUBLIC));
    }

/** ExtNoClearAlloc *********************************************************
UBYTE *ExtNoClearAlloc(size) LONG size; {
    return(RemAlloc(size, MEMF_PUBLIC));
    }

/** RemFree *****************************************************************/
```

```
UBYTE *RemFree(p) UBYTE *p; {
    if (p != NULL) {
        p -= 4;
        FreeMem(p, *((LONG *)p));
        }
    return(NULL);
    }
```

```
/*********************************************************************
 *
 * Read an ILBM file and display as a screen/window until closed.
 *   Simulated close gadget in upper left corner of window.
 *   Clicking below title bar area toggles screen bar for dragging.
 *
 *  By Carolyn Scheppner   CBM  03/15/86
 *
 * Based on early ShowILBM.c     11/12/85
 *  By Jerry Morrison, Steve Shaw, and Steve Hayes, Electronic Arts.
 *  This software is in the public domain.
 *
 * >>NOTE<<: This example must be linked with additional IFF rtn files.
 *           See linkage information below.
 *
 * The IFF reader portion is essentially a recursive-descent parser.
 * This program will look into a CAT or LIST to find a FORM ILBM, but it
 * won't look inside another FORM type for a nested FORM ILBM.
 *
 * The display portion is specific to the Commodore Amiga computer.
 *
 * Linkage Information:
 *
 * FROM     LStartup.obj, SeeILBM.o, iffr.o, ilbmr.o, unpacker.o
 * TO       SeeILBM
 * LIBRARY  LC.lib, Amiga.lib
 *
 ***********************************************************************/

#include <exec/types.h>
#include <exec/memory.h>
#include <libraries/dos.h>
#include <graphics/gfxbase.h>
#include <graphics/rastport.h>
#include <graphics/gfx.h>
#include <graphics/view.h>
#include <workbench/startup.h>
#include <intuition/intuition.h>
#include <lattice/stdio.h>

#include "iff/ilbm.h"

/* This example's max number of planes in a bitmap. Could use MaxAmDepth. */
#define EXDepth 5
#define maxColorReg (1<<EXDepth)
#define MIN(a,b)  ((a)<(b)?(a):(b))

#define SafeFreeMem(p,q)  {if(p)FreeMem(p,q);}

/* Define the size of a temp buffer used in unscrambling the ILBM rows.*/
#define bufSz 512

/* general usage pointers */
struct GfxBase         *GfxBase;
struct IntuitionBase *IntuitionBase;
struct IntuiMessage   *message;
```

```
/* Globals for displaying an image */
struct Screen   *screen1;
struct Window   *window1;
struct RastPort *rport1;
struct ViewPort *vport1;

struct BitMap   tBitMap;       /* Temp BitMap struct for small pics  */

/* For WorkBench startup */
extern struct WBStartup *WBenchMsg;
BOOL    fromWB;
struct FileLock *startLock, *newLock;

/* Other globals */
int    i, error;
BYTE   c;
BOOL   TBtoggle, Done;
ULONG class, code, pBytes;

/* Structures for new Screen, new Window */

struct   TextAttr       TextFont = {
   "topaz.font",                /* Font Name    */
   TOPAZ_EIGHTY,                /* Font Height */
   FS_NORMAL,                   /* Style        */
   FPF_ROMFONT,                 /* Preferences */
   };

struct   NewScreen      ns = {
   0, 0,                                /* LeftEdge and TopEdge    */
   0, 0,                                /* Width and Height        */
   0,                                   /* Depth                   */
   -1, -1,                              /* DetailPen and BlockPen */
   NULL,                                /* Special display modes  */
   CUSTOMSCREEN,                        /* Screen Type             */
   &TextFont,                           /* Use my font             */
   NULL,                                /* Title                   */
   NULL,                                /* No gadgets yet          */
   NULL,                                /* Ptr to CustomBitmap     */
   };

struct   NewWindow      nw = {
   0, 0,                                /* LeftEdge and TopEdge */
   0, 0,                                /* Width and Height */
   -1, -1                               /* DetailPen and BlockPen */
   MOUSEBUTTONS,                        /* IDCMP Flags */
   ACTIVATE
   |BACKDROP
   |BORDERLESS,                         /* Flags */
   NULL, NULL,                          /* Gadget and Image pointers */
   NULL,                                /* Title string */
   NULL,                                /* Put Screen ptr here */
   NULL,                                /* SuperBitMap pointer */
   0, 0,                                /* MinWidth and MinHeight */
   0, 0,                                /* MaxWidth and MaxHeight */
   CUSTOMSCREEN,                        /* Type of window */
   };

USHORT  allBgColor[32];


/* Message strings for IFFP codes. */
char MsgOkay[]        = {"(IFF_OKAY) No FORM ILBM in the file." };
char MsgEndMark[]     = {"(END_MARK) How did you get this message?" };
char MsgDone[]        = {"(IFF_DONE) All done."};
char MsgDos[]         = {"(DOS_ERROR) The DOS returned an error." };
char MsgNot[]         = {"(NOT_IFF) Not an IFF file." };
char MsgNoFile[]      = {"(NO_FILE) No such file found." };
char MsgClientError[] = {"(CLIENT_ERROR) ShowILBM bug or insufficient RAM."};
char MsgForm[]        = {"(BAD_FORM) A malformed FORM ILBM." };
char MsgShort[]       = {"(SHORT_CHUNK) A malformed FORM ILBM." };
char MsgBad[]         = {"(BAD_IFF) A mangled IFF file." };

/* THESE MUST APPEAR IN RIGHT ORDER!! */
char *IFFPMessages[-LAST_ERROR+1] = {
    /*IFF_OKAY*/  MsgOkay,
    /*END_MARK*/  MsgEndMark,
    /*IFF_DONE*/  MsgDone,
    /*DOS_ERROR*/ MsgDos,
    /*NOT_IFF*/   MsgNot,
    /*NO_FILE*/   MsgNoFile,
    /*CLIENT_ERROR*/ MsgClientError,
    /*BAD_FORM*/  MsgForm,
    /*SHORT_CHUNK*/  MsgShort,
    /*BAD_IFF*/   MsgBad
    };

/*------------ ILBM reader -----------------------------------------------*/
/* ILBMFrame is our "client frame" for reading FORMs ILBM in an IFF file.
 * We allocate one of these on the stack for every LIST or FORM encountered
 * in the file and use it to hold BMHD & CMAP properties. We also allocate
 * an initial one for the whole file.
 * We allocate a new GroupContext (and initialize it by OpenRIFF or
 * OpenRGroup) for every group (FORM, CAT, LIST, or PROP) encountered. It's
 * just a context for reading (nested) chunks.
 *
 * If we were to scan the entire example file outlined below:
 *    reading          proc(s)               new            new
 *
 * --whole file--   ReadPicture+ReadIFF   GroupContext     ILBMFrame
 * CAT              ReadICat               GroupContext
 *   LIST           GetLiILBM+ReadIList      GroupContext     ILBMFrame
 *     PROP ILBM    GetPrILBM                  GroupContext
 *       CMAP       GetCMAP
 *       BMHD       GetBMHD
 *     FORM ILBM    GetFoILBM                  GroupContext     ILBMFrame
 *       BODY       GetBODY
 *     FORM ILBM    GetFoILBM                  GroupContext     ILBMFrame
 *       BODY       GetBODY
 *   FORM ILBM      GetFoILBM                GroupContext     ILBMFrame
 */
```

```
typedef struct {
   ClientFrame clientFrame;
   UBYTE foundBMHD;
   UBYTE nColorRegs;
   BitMapHeader bmHdr;
   Color4 colorMap[maxColorReg];
   /* If you want to read any other property chunks, e.g. GRAB or CAMG, add
    * fields to this record to store them. */
   } ILBMFrame;


/* NOTE: For a simple version of this program, set Fancy to 0.
 * That'll compile a program that skips all LISTs and PROPs in the input
 * file. It will look in CATs for FORMs ILBM. That's suitable for most uses.
 *
 * For a fancy version that handles LISTs and PROPs, set Fancy to 1. */

#define Fancy  1

/* Modified by C. Scheppner */
/*  iFrame    made global -  moved from ReadPicture() */
/*  ilbmFrame made global -  moved from GetFoILBM()   */

ILBMFrame   iFrame;        /* top level client frame */
ILBMFrame   ilbmFrame;     /* global bitmap frame    */


/** main() ******************************************************************/
main(argc, argv)
   int argc;
   char **argv;
   {
   LONG              file;
   IFFP              iffp = NO_FILE;
   struct WBArg      *arg;
   char              *filename;

   fromWB = (argc==0) ? TRUE : FALSE;

   if(argc>1)                   /* Passed filename via command line  */
      {
      filename = argv[1];
      }
   else if ((argc==0)&&(WBenchMsg->sm_NumArgs > 1))
      {                         /* Passed filename via  WorkBench */
      arg = WBenchMsg->sm_ArgList;
      arg++;
      filename = (char *)arg->wa_Name;
      newLock    = (struct FileLock *)arg->wa_Lock;
      startLock  = (struct FileLock *)CurrentDir(newLock);
      }
   else if (argc==1)            /* From CLI but no filename */
      cleanexit("Usage: 'SeeILBM filename'\n");
   else                         /* From WB but no filename */
      cleanexit("\nClick ONCE on SeeILBM\nSHIFT and DoubleClick on Pic\n");
   if(!(GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",0)))
      cleanexit("Can't open graphics library\n");

   if(!(IntuitionBase=
           (struct IntuitionBase *)OpenLibrary("intuition.library",0)))
      cleanexit("Can't open graphics library\n");

   if(file = Open(filename, MODE_OLDFILE))
      {
      printf("\nCLICK PIC TOP LEFT TO END DISPLAY\n");
      printf("CLICK LOWER TO TOGGLE DRAG BAR\n");
      Delay(150);  /* wait about 3 seconds to give person time to read it */

      iffp = ReadPicture(file);
      Close(file);
      if (iffp == IFF_DONE)
         {
         error = DisplayPic(&ilbmFrame);
         if(error)  cleanexit("Can't open screen or window\n");

         TBtoggle   = FALSE;      /* Title bar toggle */
         Done       = FALSE;      /* Close flag       */
         while (!Done)
            {
            if(1<<window1->UserPort->mp_SigBit)   chkmsg();
            }
         }
      else cleanexit(IFFPMessages[-iffp]);
      }
   else cleanexit("Picture file not found.\n");

   cleanup();
   }


chkmsg()
   {
   while(message=(struct IntuiMessage *)GetMsg(window1->UserPort))
      {
      class = message->Class;
      code  = message->Code;
      ReplyMsg(message);
      switch(class)
         {
         case MOUSEBUTTONS:
            if ((code == SELECTDOWN)&&
                  (window1->MouseX < 10)&&(window1->MouseY<10))
               {
               Done = TRUE;
               }
            else if ((code == SELECTDOWN)&&
                        (window1->MouseY>10)&&(TBtoggle==FALSE))
               {
               TBtoggle = TRUE;
```

```
               ShowTitle(screen1,TRUE);
               }
            else if ((code == SELECTDOWN)&&
                        (window1->MouseY>10)&&(TBtoggle==TRUE))
               {
               TBtoggle = FALSE;
               ShowTitle(screen1,FALSE);
               }
            break;
         default:
            printf("Unknown IDCMP message\n");
         }
      }
   }


cleanexit(errstr)
   char  *errstr;
   {
   printf("\n %s \n",errstr);
   cleanup();
   if (fromWB)    /* Wait so user can read messages */
      {
      printf("\nPRESS RETURN TO CLOSE THIS WINDOW\n");
      while ((c=getchar()) != '\n');
      }
   exit();
   }

cleanup()
   {
   /* tBitMap planes were deallocated in DisplayPic() */
   if (window1)  CloseWindow(window1);
   if (screen1)  CloseScreen(screen1);
   if (IntuitionBase)  CloseLibrary(IntuitionBase);
   if (GfxBase)        CloseLibrary(GfxBase);
   if (newLock != startLock)  CurrentDir(startLock);
   }


/** getBitMap() ************************************************************
 *
 * Open screen or temp bitmap.
 *   Returns ptr destBitMap  or  0 = error
 *
 **************************************************************************/
struct BitMap *getBitMap(ptilbmFrame)
   ILBMFrame *ptilbmFrame;
   {
   int     i, nPlanes, plsize;
   SHORT  sWidth, sHeight, dWidth, dHeight;
   struct BitMap *destBitMap;

   sWidth  = ptilbmFrame->bmHdr.w;
   sHeight = ptilbmFrame->bmHdr.h;
   dWidth  = ptilbmFrame->bmHdr.pageWidth;
   dHeight = ptilbmFrame->bmHdr.pageHeight;
   nPlanes   = MIN(ptilbmFrame->bmHdr.nPlanes, EXDepth);

   for (i = 0; i < ptilbmFrame->nColorRegs; i++)
      {
      allBgColor[i] = ptilbmFrame->colorMap[0];
      }

   ns.Width  = dWidth;
   ns.Height = dHeight;
   ns.Depth  = nPlanes;

   if (ptilbmFrame->bmHdr.pageWidth <= 320)
      ns.ViewModes = 0;
   else
      ns.ViewModes = HIRES;

   if (ptilbmFrame->bmHdr.pageHeight > 200)
      ns.ViewModes |= LACE;

   if ((screen1 = (struct Screen *)OpenScreen(&ns))==NULL)    return(0);

   vport1 = &screen1->ViewPort;
   LoadRGB4(vport1, &allBgColor[0], ptilbmFrame->nColorRegs);

   nw.Width  = dWidth;
   nw.Height = dHeight;
   nw.Screen = screen1;

   if ((window1 = (struct Window *)OpenWindow(&nw))==NULL)    return(0),

   ShowTitle(screen1, FALSE);

   if ((sWidth == dWidth) && (sHeight == dHeight))
      {
      destBitMap = (struct BitMap *)screen1->RastPort.BitMap;
      }
   else
      {
      InitBitMap( &tBitMap,
                  nPlanes,
                  sWidth,
                  sHeight);

      plsize = RowBytes(ptilbmFrame->bmHdr.w) * ptilbmFrame->bmHdr.h;
      if (tBitMap.Planes[0] =
       (PLANEPTR)AllocMem(nPlanes * plsize, MEMF_CHIP))
         {
         for (i = 1; i < nPlanes; i++)
            tBitMap.Planes[i] = (PLANEPTR)tBitMap.Planes[0] + plsize*i;
         destBitMap = &tBitMap;
         }
      else
         {
         return(0);  /* can't allocate temp BitMap */
         }
```

```
      }
   return(destBitMap);          /* destBitMap allocated */
   }


/** DisplayPic() ***********************************************************
 *
 * Display loaded bitmap.  If tBitMap, first transfer to screen.
 *
 ***************************************************************************/
DisplayPic(ptilbmFrame)
   ILBMFrame *ptilbmFrame;
   {
   int    i, row, byte, nrows, nbytes;
   struct BitMap  *tbp, *sbp; /* temp and screen BitMap ptrs */
   UBYTE  *tpp, *spp;         /* temp and screen plane ptrs  */

   if (tBitMap.Planes[0])     /* transfer from tBitMap if nec. */
      {
      tbp = &tBitMap;
      sbp = screen1->RastPort.BitMap;
      nrows  = MIN(tbp->Rows, sbp->Rows);
      nbytes = MIN(tbp->BytesPerRow, sbp->BytesPerRow);

      for (i = 0; i < sbp->Depth; i++)
         {
         tpp = (UBYTE *)tbp->Planes[i];
         spp = (UBYTE *)sbp->Planes[i];
         for (row = 0; row < nrows; row++)
            {
            tpp = tbp->Planes[i] + (row * tbp->BytesPerRow);
            spp = sbp->Planes[i] + (row * sbp->BytesPerRow);
            for (byte = 0; byte < nbytes; byte++)
               {
               *spp++ = *tpp++;
               }
            }
         }
      /*  Can now deallocate the temp BitMap  */
      FreeMem(tBitMap.Planes[0],
                 tBitMap.BytesPerRow * tBitMap.Rows * tBitMap.Depth);
      }

   vport1 = &screen1->ViewPort;
   LoadRGB4(vport1, ptilbmFrame->colorMap, ptilbmFrame->nColorRegs);

   return(0);
   }


/** GetLiILBM() ************************************************************
 *
 * Called via ReadPicture to handle every LIST encountered in an IFF file.
 *
 ***************************************************************************/
#if Fancy
```

```
IFFP GetLiILBM(parent)
GroupContext *parent; {
    ILBMFrame newFrame;    /* allocate a new Frame */

    newFrame = *(ILBMFrame *)parent->clientFrame;  /* copy parent frame */

    return( ReadIList(parent, (ClientFrame *)&newFrame) );
    }
#endif


/** GetPrILBM() ***********************************************************
 *
 * Called via ReadPicture to handle every PROP encountered in an IFF file.
 * Reads PROPs ILBM and skips all others.
 *
 **************************************************************************/
#if Fancy
IFFP GetPrILBM(parent)
GroupContext *parent; {
   /*compilerBug register*/ IFFP iffp;
   GroupContext propContext;
   ILBMFrame *ilbmFrame = (ILBMFrame *)parent->clientFrame;

   if (parent->subtype != ID_ILBM)
      return(IFF_OKAY);   /* just continue scaning the file */

   iffp = OpenRGroup(parent, &propContext);
   CheckIFFP();

   do switch (iffp = GetPChunkHdr(&propContext)) {
      case ID_BMHD: {
         ilbmFrame->foundBMHD = TRUE;
         iffp = GetBMHD(&propContext, &ilbmFrame->bmHdr);
         break; }
      case ID_CMAP: {
         ilbmFrame->nColorRegs = maxColorReg; /* room for this many */
         iffp = GetCMAP( &propContext, (WORD *)ilbmFrame->colorMap,
                          &ilbmFrame->nColorRegs);
         break; }

      } while (iffp >= IFF_OKAY);/* loop if valid ID of ignored chunk or
                                  * subrtn returned IFF_OKAY (no errors).*/

   CloseRGroup(&propContext);
   return(iffp == END_MARK ? IFF_OKAY : iffp);
   }
#endif

/** GetFoILBM() ***********************************************************
 *
 * Called via ReadPicture to handle every FORM encountered in an IFF file.
 * Reads FORMs ILBM and skips all others.
 * Inside a FORM ILBM, it stops once it reads a BODY. It complains if it
 * finds no BODY or if it has no BMHD to decode the BODY.
```

```
 *
 * Once we find a BODY chunk, we'll allocate the BitMap and read the image.
 *
 * Modified by C. Scheppner:  ilbmFrame moved above main making it
 *                            global so main can call DisplayPic()
 *
 ***************************************************************************/
IFFP GetFoILBM(parent)
   GroupContext *parent;
   {
   IFFP iffp;
   GroupContext formContext;
   BYTE buffer[bufSz];
   struct BitMap *destBitMap;

   if (parent->subtype != ID_ILBM)
      return(IFF_OKAY);   /* just continue scaning the file */

   ilbmFrame = *(ILBMFrame *)parent->clientFrame;
   iffp = OpenRGroup(parent, &formContext);
   CheckIFFP();

   do switch (iffp = GetFChunkHdr(&formContext)) {
      case ID_BMHD: {
         ilbmFrame.foundBMHD = TRUE;
         iffp = GetBMHD(&formContext, &ilbmFrame.bmHdr);
         break; }
      case ID_CMAP: {
         ilbmFrame.nColorRegs = maxColorReg;  /* we have room for this many */
         iffp = GetCMAP(&formContext, (WORD *)ilbmFrame.colorMap,
                         &ilbmFrame.nColorRegs);
         break; }
      case ID_BODY: {
         if (!ilbmFrame.foundBMHD)  return(BAD_FORM);   /* No BMHD chunk! */

         if(destBitMap=(struct BitMap *)getBitMap(&ilbmFrame))
            {
            iffp = GetBODY( &formContext,
                            destBitMap,
                            NULL,
                            &ilbmFrame.bmHdr,
                            buffer,
                            bufSz);
            if (iffp == IFF_OKAY) iffp = IFF_DONE;   /* Eureka */
            }
         else
            iffp = CLIENT_ERROR;   /* not enough RAM for the bitmap */
         break; }

      case END_MARK: {
         iffp = BAD_FORM;
         break; }

   } while (iffp >= IFF_OKAY);  /* loop if valid ID of ignored chunk or a
           * subroutine returned IFF_OKAY (no errors).*/
   if (iffp != IFF_DONE)  return(iffp);

   CloseRGroup(&formContext);
   return(iffp);
   }

/** Notes on extending GetFoILBM ****************************************
 *
 * To read more kinds of chunks, just add clauses to the switch statement.
 * To read more kinds of property chunks (GRAB, CAMG, etc.) add clauses to
 * the switch statement in GetPrILBM, too.
 *
 * To read a FORM type that contains a variable number of data chunks--e.g.
 * a FORM FTXT with any number of CHRS chunks--replace the ID_BODY case with
 * an ID_CHRS case that doesn't set iffp = IFF_DONE, and make the END_MARK
 * case do whatever cleanup you need.
 *
 *************************************************************************/


/** ReadPicture() *********************************************************
 *
 * Read a picture from an IFF file, given a file handle open for reading.
 *
 * Modified by Carolyn Scheppner   CBM   03-86
 *   iFrame made global (above main)
 *   Close(file) moved to main
 *
 *************************************************************************/
IFFP ReadPicture(file)
      LONG file;
   {
   IFFP iffp = IFF_OKAY;

#if Fancy
   iFrame.clientFrame.getList = GetLiILBM;
   iFrame.clientFrame.getProp = GetPrILBM;
#else
   iFrame.clientFrame.getList = SkipGroup;
   iFrame.clientFrame.getProp = SkipGroup;
#endif
   iFrame.clientFrame.getForm = GetFoILBM;
   iFrame.clientFrame.getCat  = ReadICat ;

   /* Initialize the top-level client frame's property settings to the
    * program-wide defaults. This example just records that we haven't read
    * any BMHD property or CMAP color registers yet. For the color map, that
    * means the default is to leave the machine's color registers alone.
    * If you want to read a property like GRAB, init it here to (0, 0). */
   iFrame.foundBMHD  = FALSE;
   iFrame.nColorRegs = 0;

   iffp = ReadIFF(file, (ClientFrame *)&iFrame);
   return(iffp);
   }
```

```
/** ShowILBM.c ***************************************************************
 *
 * Read an ILBM raster image file and display it.       24-Jan-86.
 *
 * By Jerry Morrison, Steve Shaw, and Steve Hayes, Electronic Arts.
 * This software is in the public domain.
 *
 * USE THIS AS AN EXAMPLE PROGRAM FOR AN IFF READER.
 *
 * The IFF reader portion is essentially a recursive-descent parser.
 * The display portion is specific to the Commodore Amiga computer.
 *
 * NOTE: This program displays an image, pauses, then exits.
 *
 * Usage from CLI:
 *   showilbm picture1 [picture2] ...
 *
 * Usage from WorkBench:
 * Click on ShowILBM, hold down shift key, click on each picture to show,
 * Double-click on final picture to complete the selection, release the
 * shift key.
 *
 ******************************************************************************/

/* If you are constructing a Makefile, here are the names of the files
 * that you'll need to compile and link with to use showilbm:

        showilbm.c
        readpict.c
        remalloc.c
        ilbmr.c
        iffr.c
        unpacker.c
        gio.c

        and you'll have to get movmem() from lc.lib

 * robp.
 * ************************************************************************* */


#include "iff/intuall.h"
#include "libraries/dos.h"
#include "libraries/dosextens.h"
#include "iff/ilbm.h"
#include "workbench/workbench.h"
#include "workbench/startup.h"
#include "iff/readpict.h"
#include "iff/remalloc.h"

#define LOCAL static

#define MIN(a,b)  ((a)<(b)?(a):(b))
#define MAX(a,b)  ((a)>(b)?(a):(b))

/* general usage pointers */
```

```
struct GfxBase *GfxBase;
LONG IconBase;  /* Actually, "struct IconBase *" if you've got some ".h" file*/

/* For displaying an image */
LOCAL struct RastPort rP;
LOCAL struct BitMap bitmap0;
LOCAL struct RasInfo rasinfo;
LOCAL struct View v = {0};
LOCAL struct ViewPort vp = {0};

LOCAL ILBMFrame iFrame;

/* Define the size of a temporary buffer used in unscrambling the ILBM rows.*/
#define bufSz 512

/* Message strings for IFFP codes. */
LOCAL char MsgOkay[]        = {
        "(IFF_OKAY) Didn't find a FORM ILBM in the file." };
LOCAL char MsgEndMark[]     = { "(END_MARK) How did you get this message?" };
LOCAL char MsgDone[]        = { "(IFF_DONE) All done."};
LOCAL char MsgDos[]         = { "(DOS_ERROR) The DOS returned an error." };
LOCAL char MsgNot[]         = { "(NOT_IFF) Not an IFF file." };
LOCAL char MsgNoFile[]      = { "(NO_FILE) No such file found." };
LOCAL char MsgClientError[] = {
        "(CLIENT_ERROR) ShowILBM bug or insufficient RAM."};
LOCAL char MsgForm[]        = { "(BAD_FORM) A malformed FORM ILBM." };
LOCAL char MsgShort[]       = { "(SHORT_CHUNK) A malformed FORM ILBM." };
LOCAL char MsgBad[]         = { "(BAD_IFF) A mangled IFF file." };

/* THESE MUST APPEAR IN RIGHT ORDER!! */
LOCAL char *IFFPMessages[-(int)LAST_ERROR+1] = {
    /*IFF_OKAY*/  MsgOkay,
    /*END_MARK*/  MsgEndMark,
    /*IFF_DONE*/  MsgDone,
    /*DOS_ERROR*/ MsgDos,
    /*NOT_IFF*/   MsgNot,
    /*NO_FILE*/   MsgNoFile,
    /*CLIENT_ERROR*/ MsgClientError,
    /*BAD_FORM*/  MsgForm,
    /*SHORT_CHUNK*/  MsgShort,
    /*BAD_IFF*/   MsgBad
    };

/** DisplayPic() *************************************************************
 *
 * Interface to Amiga graphics ROM routines.
 *
 *****************************************************************************/
DisplayPic(bm, ptilbmFrame)
    struct BitMap *bm;  ILBMFrame *ptilbmFrame;  {
    int i;
    struct View *oldView = GfxBase->ActiView;   /* so we can restore it */

    InitView(&v);
    InitVPort(&vp);
    v.ViewPort = &vp;
    InitRastPort(&rP);
    rP.BitMap = bm;
    rasinfo.BitMap = bm;

    /* Always show the upper left-hand corner of this picture. */
    rasinfo.RxOffset = 0;
    rasinfo.RyOffset = 0;

    vp.DWidth = MAX(ptilbmFrame->bmHdr.w, 4*8);
    vp.DHeight = ptilbmFrame->bmHdr.h;

#if 0
    /* Specify where on screen to put the ViewPort. */
    vp.DxOffset = ptilbmFrame->bmHdr.x;
    vp.DyOffset = ptilbmFrame->bmHdr.y;
#else
    /* Always display it in upper left corner of screen.*/
#endif

    if (ptilbmFrame->bmHdr.pageWidth <= 320)
        vp.Modes = 0;
    else vp.Modes = HIRES;
    if (ptilbmFrame->bmHdr.pageHeight > 200) {
        v.Modes |= LACE;
        vp.Modes |= LACE;
        }
    vp.RasInfo = &rasinfo;
    MakeVPort(&v,&vp);
    MrgCop(&v);
    LoadView(&v);       /* show the picture */
    WaitBlit();
    WaitTOF();
    LoadRGB4(&vp, ptilbmFrame->colorMap, ptilbmFrame->nColorRegs);

    for (i = 0; i < 5*60; ++i)  WaitTOF();      /* Delay 5 seconds. */

    LoadView(oldView);  /* switch back to old view */
    }

/** stuff for main0() ********************************************************/
LOCAL struct WBStartup *wbStartup = 0;  /* 0 unless started from WorkBench.*/

PrintS(msg)  char *msg; {
    if (!wbStartup)  printf(msg);
    }

void GoodBye(msg)  char *msg; {
/*    PrintS(msg);  PrintS("\n");  */
    printf(msg);   printf("\n");    /* If linked with Lstartup.obj and
                                     * NOT compiled with -dTINY, this
                                     * outputs the message to the window
                                     * that Lattice opens.
                                     * ... carolyn.
                                     */
    exit(0);
    }
```

```
/** OpenArg() ****************************************************************
 *  Given a "workbench argument" (a file reference) and an I/O mode.
 *  It opens the file.
 *****************************************************************************/
LONG OpenArg(wa, openmode)  struct WBArg *wa;   int openmode; {
    LONG olddir;
    LONG file;
    if (wa->wa_Lock)    olddir = CurrentDir(wa->wa_Lock);
    file = Open(wa->wa_Name, openmode);
    if (wa->wa_Lock)    CurrentDir(olddir);
    return(file);
    }

/** main0() ******************************************************************/
void main0(wa)  struct WBArg *wa;  {
    LONG file;
    IFFP iffp = NO_FILE;

    /* load and display the picture */
    file = OpenArg(wa, MODE_OLDFILE);
    if (file)
        iffp = ReadPicture(file, &bitmap0, &iFrame, ChipAlloc);
        /* Allocates BitMap using ChipAlloc().*/
    Close(file);
    if (iffp == IFF_DONE)
        DisplayPic(&bitmap0, &iFrame);

/*    PrintS(" ");  PrintS(IFFPMessages[-iffp]);  PrintS("\n");   */
    printf(" ");  printf(IFFPMessages[-iffp]);  printf("\n");
        /* see note near definition of PrintS */

    /* clean-up */
    if (bitmap0.Planes[0])  {
        RemFree(bitmap0.Planes[0]);
                /* ASSUMES allocated all planes via a single ChipAlloc call.*/
        FreeVPortCopLists(&vp);
        FreeCprList(v.LOFCprList);
        }
    }

extern struct WBStartup *WBenchMsg;     /* added: Carolyn Scheppner */

/** main() *******************************************************************/

void main(argc, argv)  int argc;  char **argv;  {
    struct WBArg wbArg, *wbArgs;
    LONG olddir;
/*sss    struct Process *myProcess; */

    if( !(GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",0)) )
        GoodBye("No graphics.library");
    if( !(IconBase = OpenLibrary("icon.library",0)) )
        GoodBye("No icon.library");
    if (!argc) {
        /* Invoked via workbench */
/*      wbStartup = (struct WBStartup *)argv;   */
        wbStartup = WBenchMsg;  /* modified by Carolyn Scheppner */
        wbArgs = wbStartup->sm_ArgList;
        argc = wbStartup->sm_NumArgs;
        while (argc >= 2) {
            olddir = CurrentDir(wbArgs[1].wa_Lock);
            main0(&wbArgs[1]);
            argc--;    wbArgs = &wbArgs[1];
            }
#if 0
        /* [TBD] We want to get an error msg to the Workbench user... */
        if (argc < 2) {
            printf ("Usage from workbench:\n");
            printf (" Click mouse on Show-ILBM, Then hold 'SHIFT' key\n");

        /* BOTH OF THESE WERE "PrintS", see note near PrintS definition */

            GoodBye(" while double-click on file to display.");
            }
#endif
    }
    else {
        /* Invoked via CLI.  Make a lock for current directory.
         * Eventually, scan name, separate out directory reference?*/
        if (argc < 2)
            GoodBye("Usage from CLI: 'Show-ILBM filename'");
/*sss   myProcess = (struct Process *)FindTask(0); */
        wbArg.wa_Lock = 0; /*sss myProcess->pr_CurrentDir; */
        while (argc >= 2) {
            wbArg.wa_Name = argv[1];
            printf("Showing file ");   printf(wbArg.wa_Name);   printf(" ...");
        /* THESE WERE "PrintS", see note near PrintS definition */
            main0(&wbArg);
            printf("\n");
        /* THIS WAS "PrintS", see note near PrintS definition */
            argc--;    argv = &argv[1];
            }
        }
    CloseLibrary(GfxBase);
    CloseLibrary(IconBase);
    exit(0);
    }
```

```
/*----------------------------------------------------------------------*
 * unpacker.c Convert data from "cmpByteRun1" run compression. 11/15/85
 *
 * By Jerry Morrison and Steve Shaw, Electronic Arts.
 * This software is in the public domain.
 *
 *      control bytes:
 *       [0..127]   : followed by n+1 bytes of data.
 *       [-1..-127] : followed by byte to be repeated (-n)+1 times.
 *       -128       : NOOP.
 *
 * This version for the Commodore-Amiga computer.
 *----------------------------------------------------------------------*/
#include "iff/packer.h"


/*----------- UnPackRow ----------------------------------------------*/

#define UGetByte()      (*source++)
#define UPutByte(c)     (*dest++ = (c))

/* Given POINTERS to POINTER variables, unpacks one row, updating the source
 * and destination pointers until it produces dstBytes bytes. */
BOOL UnPackRow(pSource, pDest, srcBytes0, dstBytes0)
        BYTE **pSource, **pDest;  WORD srcBytes0, dstBytes0; {
    register BYTE *source = *pSource;
    register BYTE *dest   = *pDest;
    register WORD n;
    register BYTE c;
    register WORD srcBytes = srcBytes0, dstBytes = dstBytes0;
    BOOL error = TRUE;  /* assume error until we make it through the loop */
    WORD minus128 = -128;  /* get the compiler to generate a CMP.W */

    while( dstBytes > 0 )  {
        if ( (srcBytes -= 1) < 0 )  goto ErrorExit;
        n = UGetByte();

        if (n >= 0) {
            n += 1;
            if ( (srcBytes -= n) < 0 )  goto ErrorExit;
            if ( (dstBytes -= n) < 0 )  goto ErrorExit;
            do {  UPutByte(UGetByte());  } while (--n > 0);
            }

        else if (n != minus128) {
            n = -n + 1;
            if ( (srcBytes -= 1) < 0 )  goto ErrorExit;
            if ( (dstBytes -= n) < 0 )  goto ErrorExit;
            c = UGetByte();
            do {  UPutByte(c);  } while (--n > 0);
            }
        }
    error = FALSE;      /* success! */

  ErrorExit:
    *pSource = source;  *pDest = dest;
    return(error);
    }
```

```
;  Linking information for the files contained in this appendix
;  provided in the form of "program.with" files, so that you can
;  copy them individually, and give the command:
;  ALINK with program.with

;  iffchecg.with
FROM lib:Lstartup.obj,iffcheck.o,iffr.o,gio.o
LIBRARY lib:lc.lib,lib:amiga.lib
TO iffcheck

;  iffcheck.with
FROM lib:Lstartup.obj,iffcheck.o,iffr.o
LIBRARY lib:lc.lib,lib:amiga.lib
TO iffcheck

;  ilbm2raw.with
FROM lib:Lstartup.obj, ilbm2raw.o, readpict.o, ilbmr.o, unpacker.o, iffr.o*
remalloc.o
LIBRARY lib:lc.lib, lib:amiga.lib
TO ilbm2raw

;  ilbmdump.with
FROM lib:Lstartup.obj, ilbmdump.o, readpict.o, ilbmr.o, unpacker.o, iffr.o*
remalloc.o, bmprintc.o
LIBRARY lib:lc.lib, lib:amiga.lib
TO ilbmdump

;  raw2ilbg.with
FROM lib:Lstartup.obj, raw2ilbm.o, putpict.o, ilbmw.o, packer.o, iffw.o, gio.o
LIBRARY lib:lc.lib, lib:amiga.lib
TO raw2ilbm

;  raw2ilbm.with
FROM lib:Lstartup.obj, raw2ilbm.o, putpict.o, ilbmw.o, packer.o, iffw.o
LIBRARY lib:lc.lib, lib:amiga.lib
TO raw2ilbm

;  showilbg.with
FROM lib:Lstartup.obj,showilbm.o,readpict.o,ilbmr.o*
unpacker.o,iffr.o,remalloc.o gio.o
LIBRARY lib:lc.lib,lib:amiga.lib
TO showilbm

;  showilbm.with
FROM lib:Lstartup.obj,showilbm.o,readpict.o,ilbmr.o*
unpacker.o,iffr.o,remalloc.o
TO showilbm
LIBRARY lib:lc.lib,lib:amiga.lib
```

Appendix C

MEMORY MAP—DISK FORMAT

This appendix contains a software memory map and disk format information.

Software Memory Map

The software memory map is for the software developer who may be accessing the system hardware registers directly. It is simply the hardware memory map appropriate to the *Amiga ROM Kernel Reference Manuals.*

A true software memory map, showing system utilization of the various sections of RAM and free space, is not provided. The system is dynamically allocated and linked such that it would not be possible to show precisely which parts of RAM are utilized by the ROM Kernel. User code, if written with the Amiga Assembler or Amiga C and linked by the Amiga linker is relocatable and can load and execute wherever there is a large enough area of memory in which it can fit.

Therefore, aside from specifying that Exec manages the lowest parts of the 68000 memory space (exception and trap vectors), no actual software memory utilization map can be provided.

Disk Format Information

This information will be useful to the developer who wants to take over the entire machine—that is, instead of using AmigaDOS to load the application's code, the application boots directly after Kickstart, without using either AmigaDOS or Intuition. This appendix provides two pieces of information: the disk boot block format and the format of the actual data on the disk.

SYSTEM MEMORY MAP -

This system memory map for the ROM Kernel manual is a combination of the Appendicies D (mem.map) and F (8520 info) from the Amiga PC Hardware Manual. Actual bit assignments for the 8520's and current additional details can be found in the Amiga Hardware Manual.

```
Note:  If you select to read or write an address that is not
         specifically decoded, you WILL generate an address error.
--------------------------------------------------------------------

ADDRESS RANGE  NOTES
-------------  --------------------------------------
               [ 256k RAM ]

000000-3FFFFF  RAM space for 256k RAM

040000-080000  Not used if extra 256k board not installed.
               Do NOT access in this range - dangerous side effects.

    ---------------------------------------------
               [ 512k RAM ]

000000-07FFFF  RAM space for 512k RAM

080000-1FFFFF  Do NOT access in this range.

    ---------------------------------------------

200000-9FFFFF  Expansion space ( 8 megabytes )

    --------------------------------------------

A00000-BEFFFF  External decoder expansion space

               Reserved for future use.
               See note (1) below.
     -------------------------------------------

BFD000-BFDF00  8520-B (accessed only at
   =      =
                         EVEN byte addresses)

          The underlined digit chooses which of the  16
          internal  registers  of  the 8520  is  to  be
          accessed. See also note (5)
```

```
ADDRESS RANGE  NOTES
-------------  --------------------------------------
BFE001-BFEF01  8520-A (accessed only at
   =      =
                         ODD byte addresses)

         The  underlined digit chooses which of the 16
         internal  registers  of  the 8520  is  to  be
         accessed.

         Register Names are given in note (2) below.

Other addresses in the range of:

C00000-DFEFFF         Reserved for future use

DFF000-DFFFFF         Special purpose chips, where
                      the last three digits specify
                      the chip register WORD address.

                      The chip addresses are specified
                      in separate pages immediately
                      following  this  overall  memory
                      map.

E00000-E7FFFF         Reserved for future use.

E80000-EFFFFF         Expansion Slot decoding, see
                      note (1) below.

F00000-E7FFFF         Reserved for future use.

F80000-FFFFFF         SYSTEM ROM or kickstart RAM.


================================================================
```

NOTES FOR THE SYSTEM MEMORY MAP:

(1) Expansion Slot decoding:

Boards designed to respond in this range must adhere to the auto-configuration guidelines to be published in Dec. 1985 by Commodore-Amiga Inc.

(2) The names of the registers within the 8520's are as follows. The address at which each are to be accessed (per note 1 above) is given here in this list.

Address for:

8520-A	8520-B	NAME	EXPLANATION
			(write)/(read mode)
BFE001	BFD000	PRA	Peripheral Data Register A
BFE101	BFD100	PRB	Peripheral Data Register B
BFE201	BFD200	DDRB	Data Direction Register "A"
BFE301	BFD300	DDRA	Data Direction Register "B"
BFE401	BFD400	TALO	TIMER A Low Register
BFE501	BFD500	TAHI	TIMER A High Register
BFE601	BFD600	TBLO	TIMER B Low Register
BFE701	BFD700	TBHI	TIMER B High Register
BFE801	BFD800	TODLO	Low TOD Clock
BFE901	BFD900	TODMID	Mid TOD Clock
BFEA01	BFDA00	TODHI	High TOD Clock Register
BFEB01	BFDB00	-----	Unused
BFEC01	BFDC00	SDR	Serial Data Register
BFED01	BFDD00	ICR	Interrupt Control Register
BFEE01	BFDE00	CRA	Control Register A
BFEF01	BFDF00	CRB	Control Register B

SPECIAL PURPOSE CHIP ADDRESSES

The following are the "offset" addresses for the special purpose chips in the Amiga PC. Each address figure shown below must be added to the base address of hex DFF000. Again the addressing follows the convention shown in note 1 of the system memory map.

Each register is located at an even WORD (16-bit) boundary.

NAME	OFFSET	R/W	EXPLANATION
ADKCON	09E	W	Audio, Disk, Control write
ADKCONR	010	R	Audio, disk, Control read
AUD0DAT	0AA	W	Audio channel 0 Data
AUD0LCH	0A0	W	Audio channel 0 location (High 3 bits)
AUD0LCL	0A2	W	Audio channel 0 location (Low 16 bits)
AUD0LEN	0A4	W	Audio Channel 0 length
AUD0PER	0A6	W	Audio channel 0 Period
AUD0VOL	0A8	W	Audio Channel 0 Volume

NAME	OFFSET	R/W	EXPLANATION
AUD1DAT	0BA	W	Audio channel 1 Data
AUD1LCH	0B0	W	Audio channel 1 location (High 3 bits)
AUD1LCL	0B2	W	Audio channel 1 location (Low 16 bits)
AUD1LEN	0B4	W	Audio Channel 1 length
AUD1PER	0B6	W	Audio channel 1 Period
AUD1VOL	0B8	W	Audio Channel 1 Volume
AUD2DAT	0CA	W	Audio channel 2 Data
AUD2LCH	0C0	W	Audio channel 2 location (High 3 bits)
AUD2LCL	0C2	W	Audio channel 2 location (Low 16 bits)
AUD2LEN	0C4	W	Audio Channel 2 length
AUD2PER	0C6	W	Audio channel 2 Period
AUD2VOL	0C8	W	Audio Channel 2 Volume
AUD3DAT	0DA	W	Audio channel 3 Data
AUD3LCH	0D0	W	Audio channel 3 location (High 3 bits)
AUD3LCL	0D2	W	Audio channel 3 location (Low 16 bits)
AUD3LEN	0D4	W	Audio Channel 3 length
AUD3PER	0D6	W	Audio channel 3 Period
AUD3VOL	0D8	W	Audio Channel 3 Volume
BLTAFWM	044	W	Blitter first word mask for source A
BLTALWM	046	W	Blitter last word mask for source A
BLTCON0	040	W	Blitter control register 0
BLTCON1	042	W	Blitter control register 1
BLTSIZE	058	W	Blitter start and size (window width, height)
BLTADAT	074	W	Blitter source A data reg
BLTAMOD	064	W	Blitter Modulo A
BLTAPTH	050	W	Blitter Pointer to src or dst.A (High 3 bits)
BLTAPTL	052	W	Blitter Pointer A (Low 16 bits)
BLTBDAT	072	W	Blitter source B data reg
BLTBMOD	062	W	Blitter Modulo B
BLTBPTH	04C	W	Blitter Pointer to src or dst.B (High 3 bits)
BLTBPTL	04E	W	Blitter Pointer B (Low 16 bits)
BLTCDAT	070	W	Blitter source C data reg
BLTCMOD	060	W	Blitter Modulo C
BLTCPTH	048	W	Blitter Pointer to src or dst.C (High 3 bits)
BLTCPTL	04A	W	Blitter Pointer C (Low 16 bits)
BLTDMOD	066	W	Blitter Modulo D
BLTDPTH	054	W	Blitter Pointer to src or dst.D (High 3 bits)
BLTDPTL	056	W	Blitter Pointer D (Low 16 bits)
BPL1MOD	108	W	Bit plane modulo (odd planes)
BPL2MOD	10A	W	Bit Plane modulo (even planes)

NAME	OFFSET	R/W	EXPLANATION
BPLCON0	100	W	Bit plane control reg.(misc control bits)
BPLCON1	102	W	Bit plane control reg.(priority control)
BPLCON2	104	W	Bit Plane control reg.(horiz scroll control)
BPL1DAT	110	W	Bit plane 1 data (Parallel to serial convert)
BPL2DAT	112	W	Bit plane 2 data (Parallel to serial convert)
BPL3DAT	114	W	Bit plane 3 data (Parallel to serial convert)
BPL4DAT	116	W	Bit plane 4 data (Parallel to serial convert)
BPL5DAT	118	W	Bit plane 5 data (Parallel to serial convert)
BPL6DAT	11A	W	Bit plane 6 data (Parallel to serial convert)
BPL1PTH	0E0	W	Bit plane 1 pointer (High 3 bits)
BPL1PTL	0E2	W	Bit plane 1 pointer (Low 16 bits)
BPL2PTH	0E4	W	Bit plane 2 pointer (High 3 bits)
BPL2PTL	0E6	W	Bit plane 2 pointer (Low 16 bits)
BPL3PTH	0E8	W	Bit plane 3 pointer (High 3 bits)
BPL3PTL	0EA	W	Bit plane 3 pointer (Low 16 bits)
BPL4PTH	0EC	W	Bit plane 4 pointer (High 3 bits)
BPL4PTL	0EE	W	Bit plane 4 pointer (Low 16 bits)
BPL5PTH	0F0	W	Bit plane 5 pointer (High 3 bits)
BPL5PTL	0F2	W	Bit plane 5 pointer (Low 16 bits)
BPL6PTH	0F4	W	Bit plane 6 pointer (High 3 bits)
BPL6PTL	0F6	W	Bit plane 6 pointer (Low 16 bits)
CLXCON	098	W	Collision control
CLXDAT	00E	R	Collision data reg.(Read and clear)
COLORxx	180	W	Color table xx (32 WORD ENTRIES, START COLOR 00)
COP1LCH	080	W	Coprocessor first location reg (High 3 bits)
COP1LCL	082	W	Coprocessor first location reg. (Low 16 bits)
COP2LCH	084	W	Coprocessor secnd location reg. (High 3 bits)
COP2LCL	086	W	Coprocessor second location reg (Low 16 bits)
COPINS	08C	W	Coprocessor inst. fetch identify
COPJMP1	088	S	Coprocessor restart at first location
COPJMP2	08A	S	Coprocessor restart at second location
DDFSTOP	094	W	Display bit plane data fetch stop(hor pos)
DDFSTRT	092	W	Display bit plane data fetch start(hor pos)

NAME	OFFSET	R/W	EXPLANATION
DIWSTOP	090	W	Disp Window Stop (lower right vert-hor pos)
DIWSTRT	08E	W	Disp Window Start (upper left vert-hor pos)
DMACON	096	W	DMA control write(clear or set)
DMACONR	002	R	DMA control (and blitter status) read
DSKBYTR	01A	R	Disk Data byte and status read
DSKDAT	026	W	Disk DMA Data write
DSKDATR	008	ER	Disk DMA Data read (early read dummy address)
DSKLEN	024	W	Disk length
DSKPTH	020	W	Disk pointer (High 3 bits)
DSKPTL	022	W	Disk pointer (Low 16 bits)
INTENA	09A	W	Interrupt Enable bits (clear or set bits)
INTENAR	01C	R	Interrupt Enable bits Read
INTREQ	09C	W	Interrupt Request bits (clear or set)
INTREQR	01E	R	Interrupt request bits (read)
JOY0DAT	00A	R	Joystick-mouse 0 data (vert,horiz)
JOY1DAT	00C	R	Joystick-mouse 1 data (vert,horiz)
POT0DAT	012	R	Pot counter data left pair (vert,horiz)
POT1DAT	014	R	Pot counter data right pair (vert,horiz)
POTGO	034	W	Pot Port (4 bit) Direction and Data,
POTINP	016	R	Pot pin data read
REFPTR	028	W	Refresh pointer
SERDAT	030	W	Serial Port Data and stop bits write
SERDATR	018	R	Serial Port Data and Status read
SERPER	032	W	Serial Port Period and control
SPR0POS	140	W	Sprite 0 Vert-Horiz start position data
SPR0CTL	142	W	Sprite 0 Vert stop position and control data
SPR0DATA	144	W	Sprite 0 image data register A
SPR0DATB	146	W	Sprite 0 image data register B
SPR1POS	148	W	Sprite 1 Vert-Horiz start position data
SPR1CTL	14A	W	Sprite 1 Vert stop position and control data
SPR1DATA	14C	W	Sprite 1 image data register A
SPR1DATB	14E	W	Sprite 1 image data register B

Dec 9 15:18 1985 Software.Memory.Map Page 8

NAME	OFFSET	R/W	EXPLANATION
SPR2POS	150	W	Sprite 2 Vert-Horiz start position data
SPR2CTL	152	W	Sprite 2 Vert stop position and control data
SPR2DATA	154	W	Sprite 2 image data register A
SPR2DATB	156	W	Sprite 2 image data register B
SPR3POS	158	W	Sprite 3 Vert-Horiz start position data
SPR3CTL	15A	W	Sprite 3 Vert stop position and control data
SPR3DATA	15C	W	Sprite 3 image data register A
SPR3DATB	15E	W	Sprite 3 image data register B
SPR4POS	160	W	Sprite 4 Vert-Horiz start position data
SPR4CTL	162	W	Sprite 4 Vert stop position and control data
SPR4DATA	164	W	Sprite 4 image data register A
SPR4DATB	166	W	Sprite R image data register B
SPR5POS	168	W	Sprite 5 Vert-Horiz start position data
SPR5CTL	16A	W	Sprite 5 Vert stop position and control data
SPR5DATA	16C	W	Sprite 5 image data register A
SPR5DATB	16E	W	Sprite 5 image data register B
SPR6POS	170	W	Sprite 6 Vert-Horiz start position data
SPR6CTL	172	W	Sprite 6 Vert stop position and control data
SPR6DATA	174	W	Sprite 6 image data register A
SPR6DATB	176	W	Sprite 6 image data register B
SPR7POS	178	W	Sprite 7 Vert-Horiz start position data
SPR7CTL	17A	W	Sprite 7 Vert stop position and control data
SPR7DATA	17C	W	Sprite 7 image data register A
SPR7DATB	17E	W	Sprite 7 image data register B
SPR0PTH	120	W	Sprite 0 pointer (High 3 bits)
SPR0PTL	122	W	Sprite 0 pointer (Low 16 bits)
SPR1PTH	124	W	Sprite 1 pointer (High 3 bits)
SPR1PTL	126	W	Sprite 1 pointer (Low 16 bits)
SPR2PTH	128	W	Sprite 2 pointer (High 3 bits)
SPR2PTL	12A	W	Sprite 2 pointer (Low 16 bits)
SPR3PTH	12C	W	Sprite 3 pointer (High 3 bits)
SPR3PTL	12E	W	Sprite 3 pointer (Low 16 bits)
SPR4PTH	130	W	Sprite 4 pointer (High 3 bits)
SPR4PTL	132	W	Sprite 4 pointer (Low 16 bits)
SPR5PTH	134	W	Sprite 5 pointer (High 3 bits)
SPR5PTL	136	W	Sprite 5 pointer (Low 16 bits)
SPR6PTH	138	W	Sprite 6 pointer (High 3 bits)
SPR6PTL	13A	W	Sprite 6 pointer (Low 16 bits)
SPR7PTH	13C	W	Sprite 7 pointer (High 3 bits)
SPR7PTL	13E	W	Sprite 7 pointer (Low 16 bits)

Dec 9 15:18 1985 Software.Memory.Map Page 9

NAME	OFFSET	R/W	EXPLANATION
STREQU	038	S	Strobe for horiz sync with VB and EQU
STRHOR	03C	S	Strobe for horiz sync
STRLONG	03E	S	Strobe for identification of long horzontal line.
STRVBL	03A	S	Strobe for horiz sync with VB (Vert. Blank)
VHPOSR	004	R	Read Vert and horiz Position of beam
VHPOSW	02A	W	Write Vert and horiz Position of beam
VPOSR	006	R	Read Vert most sig. bit (and frame flop)
VPOSW	02C	W	Write Vert most sig. bit (and frame flop)

The following listing of the chip special purpose addresses is provided in numerical order by chip address for the convenience of software developers who may prefer this ordering sequence.

DMACONR	002	R	DMA control (and blitter status) read
VHPOSR	004	R	Read Vert and horiz Position of beam
VPOSR	006	R	Read Vert most sig. bit (and frame flop)
DSKDATR	008	ER	Disk DMA Data read (early read dummy address)
JOY0DAT	00A	R	Joystick-mouse 0 data (vert,horiz)
JOY1DAT	00C	R	Joystick-mouse 1 data (vert,horiz)
CLXDAT	00E	R	Collision data reg.(Read and clear)
ADKCONR	010	R	Audio, disk, Control read
POT0DAT	012	R	Pot counter data left pair (vert,horiz)
POT1DAT	014	R	Pot counter data right pair (vert,horiz)
POTINP	016	R	Pot pin data read
SERDATR	018	R	Serial Port Data and Status read
DSKBYTR	01A	R	Disk Data byte and status read
INTENAR	01C	R	Interrupt Enable bits Read
INTREQR	01E	R	Interrupt request bits (read)
DSKPTH	020	W	Disk pointer (High 3 bits)
DSKPTL	022	W	Disk pointer (Low 16 bits)
DSKLEN	024	W	Disk length
DSKDAT	026	W	Disk DMA Data write
REFPTR	028	W	Refresh pointer
VHPOSW	02A	W	Write Vert and horiz Position of beam
VPOSW	02C	W	Write Vert most sig. bit (and frame flop)
SERDAT	030	W	Serial Port Data and stop bits write
SERPER	032	W	Serial Port Period and control
POTGO	034	W	Pot Port (4 bit) Direction and Data,
STREQU	038	S	Strobe for horiz sync with VB and EQU
STRVBL	03A	S	Strobe for horiz sync with VB (Vert. Blank)
STRHOR	03C	S	Strobe for horiz sync
STRLONG	03E	S	Strobe for ident. of long horzontal line.
BLTCON0	040	W	Blitter control register 0
BLTCON1	042	W	Blitter control register 1
BLTAFWM	044	W	Blitter first word mask for source A
BLTALWM	046	W	Blitter last word mask for source A
BLTCPTH	048	W	Blitter Pointer to src or dst.C (High 3 bits)

```
BLTCPTL     04A  W   Blitter Pointer C (Low 16 bits)
BLTBPTH     04C  W   Blitter Pointer to src or dst.B (High 3 bits)
BLTBPTL     04E  W   Blitter Pointer B (Low 16 bits)
BLTAPTH     050  W   Blitter Pointer to src cr dst.A (High 3 bits)
BLTAPTL     052  W   Blitter Pointer A (Low  6 bits)
BLTDPTH     054  W   Blitter Pointer to src or dst.D (High 3 bits)
BLTDPTL     056  W   Blitter Pointer D (Low 16 bits)
BLTSIZE     058  W   Blitter start and size (window width, height)
BLTCMOD     060  W   Blitter Modulo C
BLTBMOD     062  W   Blitter Modulo B
BLTAMOD     064  W   Blitter Modulo A
BLTDMOD     066  W   Blitter Modulo D
BLTCDAT     070  W   Blitter source C data reg
BLTBDAT     072  W   Blitter source B data reg
BLTADAT     074  W   Blitter source A data reg
COP1LCH     080  W   Coprocessor first location reg (High 3 bits)
COP1LCL     082  W   Coprocessor first location reg.(Low 16 bits)
COP2LCH     084  W   Coprocessor secnd location reg.(High 3 bits)
COP2LCL     086  W   Coprocessor second location reg (low 16 bits)
COPJMP1     088  S   Coprocessor restart at first location
COPJMP2     08A  S   Coprocessor restart at second location
COPINS      08C  W   Coprocessor inst. fetch identify
DIWSTRT     08E  W   Disp Window Start (upper left vert-hor pos)
DIWSTOP     090  W   Disp Window Stop (lower right vert-hor pos)
DDFSTRT     092  W   Display bit plane data fetch start(hor pos)
DDFSTOP     094  W   Display bit plane data fetch stop(hor pos)
DMACON      096  W   DMA control write(clear or set)
CLXCON      098  W   Collision control
INTENA      09A  W   Interrupt Enable bits (clear or set  bits)
INTREQ      09C  W   Interrupt Request bits (clear or set)
ADKCON      09E  W   Audio, Disk, Control write
AUD0LCH     0A0  W   Audio channel 0 location (High 3 bits)
AUD0LCL     0A2  W   Audio channel 0 location (Low 16 bits)
AUD0LEN     0A4  W   Audio Channel 0 length
AUD0PER     0A6  W   Audio Channel 0 Period
AUD0VOL     0A8  W   Audio Channel 0 Volume
AUD0DAT     0AA  W   Audio channel 0 Data
AUD1LCH     0B0  W   Audio channel 1 location (High 3 bits)
AUD1LCL     0B2  W   Audio channel 1 location (Low 16 bits)
AUD1LEN     0B4  W   Audio Channel 1 length
AUD1PER     0B6  W   Audio channel 1 Period
AUD1VOL     0B8  W   Audio Channel 1 Volume
AUD1DAT     0BA  W   Audio channel 1 Data
AUD2LCH     0C0  W   Audio channel 2 location (High 3 bits)
AUD2LCL     0C2  W   Audio channel 2 location (Low 16 bits)
AUD2LEN     0C4  W   Audio Channel 2 length
AUD2PER     0C6  W   Audio channel 2 Period
AUD2VOL     0C8  W   Audio Channel 2 Volume
AUD2DAT     0CA  W   Audio channel 2 Data
AUD3LCH     0D0  W   Audio channel 3 location (High 3 bits)
AUD3LCL     0D2  W   Audio channel 3 location (Low 16 bits)
AUD3LEN     0D4  W   Audio Channel 3 length
AUD3PER     0D6  W   Audio channel 3 Period
AUD3VOL     0D8  W   Audio Channel 3 Volume
AUD3DAT     0DA  W   Audio channel 3 Data
BPL1PTH     0E0  W   Bit plane 1 pointer (High 3 bits)
```

```
BPL1PTL     0E2  W   Bit plane 1 pointer (Low 16 bits)
BPL2PTH     0E4  W   Bit plane 2 pointer (High 3 bits)
BPL2PTL     0E6  W   Bit plane 2 pointer (Low 16 bits)
BPL3PTH     0E8  W   Bit plane 3 pointer (High 3 bits)
BPL3PTL     0EA  W   Bit plane 3 pointer (Low 16 bits)
BPL4PTH     0EC  W   Bit plane 4 pointer (High 3 bits)
BPL4PTL     0EE  W   Bit plane 4 pointer (Low 16 bits)
BPL5PTH     0F0  W   Bit plane 5 pointer (High 3 bits)
BPL5PTL     0F2  W   Bit plane 5 pointer (Low 16 bits)
BPL6PTH     0F4  W   Bit plane 6 pointer (High 3 bits)
BPL6PTL     0F6  W   Bit plane 6 pointer (Low 16 bits)
BPLCON0     100  W   Bit plane control reg.(misc control bits)
BPLCON1     102  W   Bit plane control reg.(priority control)
BPLCON2     104  W   Bit Plane control reg.(horiz scroll control)
BPL1MOD     108  W   Bit plane modulo (odd planes)
BPL2MOD     10A  W   Bit Plane modulo (even planes)
BPL1DAT     110  W   Bit plane 1 data (Parallel to serial convert)
BPL2DAT     112  W   Bit plane 2 data (Parallel to serial convert)
BPL3DAT     114  W   Bit plane 3 data (Parallel to serial convert)
BPL4DAT     116  W   Bit plane 4 data (Parallel to serial convert)
BPL5DAT     118  W   Bit plane 5 data (Parallel to serial convert)
BPL6DAT     11A  W   Bit plane 6 data (Parallel to serial convert)
SPR0PTH     120  W   Sprite 0 pointer (High 3 bits)
SPR0PTL     122  W   Sprite 0 pointer (Low 16 bits)
SPR1PTH     124  W   Sprite 1 pointer (High 3 bits)
SPR1PTL     126  W   Sprite 1 pointer (Low 16 bits)
SPR2PTH     128  W   Sprite 2 pointer (High 3 bits)
SPR2PTL     12A  W   Sprite 2 pointer (Low 16 bits)
SPR3PTH     12C  W   Sprite 3 pointer (High 3 bits)
SPR3PTL     12E  W   Sprite 3 pointer (Low 16 bits)
SPR4PTH     130  W   Sprite 4 pointer (High 3 bits)
SPR4PTL     132  W   Sprite 4 pointer (Low 16 bits)
SPR5PTH     134  W   Sprite 5 pointer (High 3 bits)
SPR5PTL     136  W   Sprite 5 pointer (Low 16 bits)
SPR6PTH     138  W   Sprite 6 pointer (High 3 bits)
SPR6PTL     13A  W   Sprite 6 pointer (Low 16 bits)
SPR7PTH     13C  W   Sprite 7 pointer (High 3 bits)
SPR7PTL     13E  W   Sprite 7 pointer (Low 16 bits)
SPR0POS     140  W   Sprite 0 Vert-Horiz start position data
SPR0CTL     142  W   Sprite 0 Vert stop position and control data
SPR0DATA    144  W   Sprite 0 image data register A
SPR0DATB    146  W   Sprite 0 image data register B
SPR1POS     148  W   Sprite 1 Vert-Horiz start position data
SPR1CTL     14A  W   Sprite 1 Vert stop position and control data
SPR1DATA    14C  W   Sprite 1 image data register A
SPR1DATB    14E  W   Sprite 1 image data register B
SPR2POS     150  W   Sprite 2 Vert-Horiz start position data
SPR2CTL     152  W   Sprite 2 Vert stop position and control data
SPR2DATA    154  W   Sprite 2 image data register A
SPR2DATB    156  W   Sprite 2 image data register B
SPR3POS     158  W   Sprite 3 Vert-Horiz start position data
SPR3CTL     15A  W   Sprite 3 Vert stop position and control data
SPR3DATA    15C  W   Sprite 3 image data register A
SPR3DATB    15E  W   Sprite 3 image data register B
SPR4POS     160  W   Sprite 4 Vert-Horiz start position data
SPR4CTL     162  W   Sprite 4 Vert stop position and control data
```

```
SPR4DATA   164  W   Sprite 4 image data register A
SPR4DATB   166  W   Sprite 4 image data register B
SPR5POS    168  W   Sprite 5 Vert-Horiz start position data
SPR5CTL    16A  W   Sprite 5 Vert stop position and control data
SPR5DATA   16C  W   Sprite 5 image data register A
SPR5DATB   16E  W   Sprite 5 image data register B
SPR6POS    170  W   Sprite 6 Vert-Horiz start position data
SPR6CTL    172  W   Sprite 6 Vert stop position and control data
SPR6DATA   174  W   Sprite 6 image data register A
SPR6DATB   176  W   Sprite 6 image data register B
SPR7POS    178  W   Sprite 7 Vert-Horiz start position data
SPR7CTL    17A  W   Sprite 7 Vert stop position and control data
SPR7DATA   17C  W   Sprite 7 image data register A
SPR7DATB   17E  W   Sprite 7 image data register B
COLORxx    180  W   Color table xx (32 WORD ENTRIES, START COLOR 00)
```

THE BOOT PROCESS

The first two sectors are read into the system at an arbitrary position; therefore, the code MUST be PC-relative. The first three longwords are as in devices/bootblock.h. The type should be BBID_DOS; the checksum must be correct (as in additive carry wraparound sum of 0xffffffff). Execution starts at location 12 of the sectors that were read in.

The code is called with an open disk I/O request in A1 (see the TrackDisk chapter for the format of this IORequest block). The boot code is free to use it as it wishes (it may trash A1, but must not trash the io block itself).

The boot code returns two values: D0 and A0. D0 is a failure code -- if it is non-zero then a system alert will be called, then the boot code falls into the debugger.

If D0 is null then A0 contains the start address to jump to. The strap module will free the boot sectors, close the I/O block, do any other cleanup that is required, and jump to the location pointed to by A0.

COMMODORE-AMIGA DISK FORMAT

The following are details about how the bits on the Commodore-Amiga disk are actually written.

Gross Data Organization:

```
        3 1/2 inch disk
        double-sided
        80 cylinders/160 tracks
```

Per-track Organization:

```
        Nulls written as a gap, then 11 sectors of data.
        No gaps written between sectors.
```

Per-sector Organization:

```
        All data is MFM encoded.  This is the pre-encoded contents
        of each sector:

                two bytes of 00 data    (MFM = AAAA each)
                two bytes of A1*        ( "standard sync byte" -- MFM
                                        encoded A1 without a clock pulse )
                                        (MFM = 4489 each)
                one byte  of format-byte
                                        (Amiga 1.0 format = FF)
                one byte  of track number
                one byte  of sector number
                one byte  of sectors until end of write (NOTE 1)
```

```
                [above 4 bytes treated as one longword
                  for purposes of MFM encoding]

        16  bytes of OS recovery info (NOTE 2)
                [treated as a block of 16 bytes for encoding]
        four bytes of header checksum
                [treated as a longword for encoding]
        four bytes of data-area checksum
                [treated as a longword for encoding]
        512 bytes of data
                [treated as a block of 512 bytes for encoding]
```

NOTES:

NOTE 1.

The track number and sector number are constant for each particular sector. However, the sector offset byte changes each time we rewrite the track.

The Amiga does a full track read starting at a random position on the track and going for slightly more than a full track read to assure that all data gets into the buffer. The data buffer is examined to determine where the first sector of data begins as compared to the start of the buffer. The track data is block moved to the beginning of the buffer so as to align some sector with the first location in the buffer.

Because we start reading at a random spot, the read data may be divided into three chunks: a series of sectors, the track gap, and another series of sectors. The sector offset value tells the disk software how many more sectors remain before the gap. From this the software can figure out the buffer memory location of the last byte of legal data in the buffer. It can then search past the gap for the next sync byte and, having found it, can block move the rest of the disk data so that all 11 sectors of data are contiguous.

Example:

```
    first-ever write of the track from a buffer like this:

    <GAP> |sector0|sector1|sector2|.....|sector10|

    sector offset values:

          11     10      9    ....    1

       (if I find this one at the start of my read buffer,
         then I know there are this many more sectors
         with no intervening gaps before I hit a gap).
```

```
        sample read of this track:

        <junk>|sector9|sector10|<gap>|sector0|...|sector8|<junk>

        value of 'sectors till end of write':

              2      1     ....    11        3

        result of track realligning:

        <GAP>|sector9|sector10|sector0|...|sector8|

        new sectors till end of write:

             11     10      9    ...    1

        so that when the track is rewritten, the sector offsets
        are adjusted to match the way the data was written.

    NOTE 2. This is operating systems dependent data and relates
        to how AmigaDos assigns sectors to files.

        Reserved for future use.


    GENERAL:

        When data is MFM encoded, the encoding is performed on
        the basis of a data block-size.  In the sector encoding
        described above, there are bytes individually encoded;
        three segments of 4 bytes of data each, treated as
        longwords; one segment of 16 bytes treated as a block; two
        segments of longwords for the header and data checksums;
        and the data area of 512 bytes treated as a block.

        When the data is encoded, the odd bits are encoded first,
        then the even bits of the block.

        (Make a block of bytes formed from all odd bits of the block,
         encode as MFM.

         Make a block of bytes formed from all even bits of the block,
         encode as MFM.   Even bits are shifted left one bit position
         before being encoded.)
```

```
SOURCE CODE FOR DATA ENCODE/DECODE

decodeBlock( mfmbuffer, userbuffer, numwords )
WORD *mfmbuffer;          /* the encoded data */
WORD *userbuffer;         /* where to put the decoded data */
int numwords;             /* the number of WORDS of data (not bytes) */
{
    WORD *oddptr, *evenptr, oddbits, evenbits;

    oddptr = mfmbuffer;

    /* the even region starts right after the odd one */
    evenptr = &mfmbuffer[numwords];

    while( numwords-- > 0 ) {
        /* mask off the mfm clock bits, and shift the word */
        oddbits = ((*oddptr++ << 1) & 0xAAAA);

        /* even bits are already in the right place.  Just mask off clock */
        evenbits = ((*evenptr++) & 0x5555);

        /* recombine the two sections */
        *userbuffer++ = oddbits | evenbits;
    }
}

encodeBlock( mfmbuffer, userbuffer, numwords )
WORD *mfmbuffer;          /* where to put the encoded data */
WORD *userbuffer;         /* the user data, before encoding */
int numwords;             /* the number of WORDS of data (not bytes) */
{
    WORD *oddptr, *evenptr;
    WORD *ubuf;


    oddptr = mfmbuffer;

    /* the even region starts right after the odd one */
    evenptr = &mfmbuffer[numwords];

    /* mfmencode takes one word of mfm data can correctly sets
     * the clock bits
     */

    /* encode the odd bits */
    for( ubuf = userbuffer, i = numwords; i > 0; i-- ) {
        oddptr++ = mfmencode( (*ubuf++ >> 1) & 0x5555 );
    }

    /* encode the even bits */
    for( ubuf = userbuffer, i = numwords; i > 0; i-- ) {
        evenptr++ = mfmencode( *ubuf++ & 0x5555 );
    }
}
```

Appendix D

LIBRARY BASE OFFSETS

This appendix is primarily for the convenience of assembly language developers in debugging their code.

The first three columns in each list contain three different ways of expressing the same information, namely the negative offset from the library base of the entry point to the particular routine. Normally these values will be symbolically encoded rather than hard-coded into your assembly language source code; the Amiga linker resolves the symbol references. The file *amiga.lib* contains the routine names preceded by "_LVO," which stands for "Library Vector Offset."

The fourth column combines the routine name, its parameters, and the registers in which the routine expects the parameters to be loaded (in the same order as the list of names). This would be used, for example, if you wanted to call the graphics library routine **ReadPixel()**. At the assembly-language level, you would do the following. Note that this is a code fragment only, and it assumes that the graphics library has not yet been opened.

```
         XREF     _LVOOpenLibrary
         XREF     _LVOReadPixel
         XREF     ExecBase

GFXNAME  DC.B     'graphics.library'

         MOVE.L   A6,-(SP)              ;Save current A6 value

         MOVE.L   ExecBase,A6           ;Base address of exec.library
         LEA      GFXNAME(PC),A1        ;name of graphics library
         MOVE.L   VERSION,D0            ;version number (currently 31)
         JSR      _LVOOpenLibrary(A6)   ;call the function
         MOVE.L   (SP)+,A6              ;restore A6
         MOVE.L   D0,GfxBase            ;save the value for later

         * later in the code ....

         MOVE.L   A6,-(SP)              ;Save current A6 value
         MOVE.L   GfxBase,A6            ;Base address of graphics lib.
         MOVE.L   myrastport,A1         ;address of my rast port
         MOVE.L   xposition,D0          ;x coordinate
         MOVE.L   yposition,D1          ;y coordinate
         JSR      _LVOReadPixel(A6)     ;call the function
         MOVE.L   (SP)+,A6              ;restore A6
         MOVE.L   D0,valuefound         ;value returns in D0
```

After a call to a library routine, the contents of register A6 will be kept the same as as it was at the entry to the library routine. Therefore, if you are making repeated calls to the same library, the base address loading need only be performed once.

You can make your code more readable by defining the following macros:

```
CALLSYS   MACRO
          CALLLIB    _LVO\1
          ENDM

LINKSYS   MACRO
          LINKLIB    _LVO\1
          ENDM

XLIB      MACRO
          XREF       _LVO\1
          ENDM
```

where **CALLLIB** and **LINKLIB** are already defined in *exec/libraries.i.* In place of:

```
XREF      _LVOOpenLibrary

JSR       _LVOOpenLibrary(A6)
```

you would use:

```
XLIB      OpenLibrary

CALLSYS   OpenLibrary
```

If you are not sure that the address in A6 will be correct at entry to a particular routine, you can use the **LINKSYS** macro. This macro will save the current A6, use A6 to call the library, then restore A6 at exit. For this alternate macro, you specify both the name of the library base and the offset value for the routine within that library. The offset value is resolved by the linker. Here is the code from the above example with macros used to make the code more compact.

```
; (macro definitions elsewhere)

XLIB        OpenLibrary
XLIB        ReadPixel

LEA         GFXNAME,A1        ;name of graphics library
MOVE.L      VERSION,D0        ;version number (currently 31)
CALLSYS     ExecBase,OpenLibrary
MOVE.L      D0,GfxBase        ;save the value for later

; later in the code ....

MOVE.L      myrastport,A1     ;address of my rast port
MOVE.L      xposition,D0      ;x coordinate
MOVE.L      yposition,D1      ;y coordinate
CALLSYS     GfxBase,ReadPixel
MOVE.L      D0,valuefound     ;value returns in D0
```

```
clist.lib.offsets


 30 0xffe2 -0x001e InitCLPool(cLPool,size)(A0,D0)
 36 0xffdc -0x0024 AllocCList(cLPool)(A1)
 42 0xffd6 -0x002a FreeCList(cList)(A0)
 48 0xffd0 -0x0030 FlushCList(cList)(A0)
 54 0xffca -0x0036 SizeCList(cList)(A0)
 60 0xffc4 -0x003c PutCLChar(cList,byte)(A0,D0)
 66 0xffbe -0x0042 GetCLChar(cList)(A0)
 72 0xffb8 -0x0048 UnGetCLChar(cList,byte)(A0,D0)
 78 0xffb2 -0x004e UnPutCLChar(cList)(A0)
 84 0xffac -0x0054 PutCLWord(cList,word)(A0,D0)
 90 0xffa6 -0x005a GetCLWord(cList)(A0)
 96 0xffa0 -0x0060 UnGetCLWord(cList,word)(A0,D0)
102 0xff9a -0x0066 UnPutCLWord(cList)(A0)
108 0xff94 -0x006c PutCLBuf(cList,buffer,length)(A0,A1,D1)
114 0xff8e -0x0072 GetCLBuf(cList,buffer,maxLength)(A0,A1,D1)
120 0xff88 -0x0078 MarkCList(cList,offset)(A0,D0)
126 0xff82 -0x007e IncrCLMark(cList)(A0)
132 0xff7c -0x0084 PeekCLMark(cList)(A0)
138 0xff76 -0x008a SplitCList(cList)(A0)
144 0xff70 -0x0090 CopyCList(cList)(A0)
150 0xff6a -0x0096 SubCList(cList,index,length)(A0,D0/D1)
156 0xff64 -0x009c ConcatCList(sourceCList,destCList)(A0,A1)



console.lib.offsets


 42 0xffd6 -0x002a CDInputHandler(events,device)(A0/A1)
 48 0xffd0 -0x0030 RawKeyConvert(events,buffer,length,keyMap)(A0/A1,D1/A2)



diskfont.lib.offsets


 30 0xffe2 -0x001e OpenDiskFont(textAttr)(A0)
 36 0xffdc -0x0024 AvailFonts(buffer,bufBytes,flags)(A0,D0/D1)



dos.lib.offsets


  0 0x0000 -0x0000 Open(name,accessMode)(D1/D2)
  6 0xfffa -0x0006 Close(file)(D1)
 12 0xfff4 -0x000c Read(file,buffer,length)(D1/D2/D3)
 18 0xffee -0x0012 Write(file,buffer,length)(D1/D2/D3)
 24 0xffe8 -0x0018 Input()
 30 0xffe2 -0x001e Output()
 36 0xffdc -0x0024 Seek(file,position,offset)(D1/D2/D3)
 42 0xffd6 -0x002a DeleteFile(name)(D1)
 48 0xffd0 -0x0030 Rename(oldName,newName)(D1/D2)
 54 0xffca -0x0036 Lock(name,type)(D1/D2)
 60 0xffc4 -0x003c UnLock(lock)(D1)
 66 0xffbe -0x0042 DupLock(lock)(D1)
 72 0xffb8 -0x0048 Examine(lock,fileInfoBlock)(D1/D2)
```

```
 78 0xffb2 -0x004e ExNext(lock,fileInfoBlock)(D1/D2)
 84 0xffac -0x0054 Info(lock,parameterBlock)(D1/D2)
 90 0xffa6 -0x005a CreateDir(name)(D1)
 96 0xffa0 -0x0060 CurrentDir(lock)(D1)
102 0xff9a -0x0066 IoErr()
108 0xff94 -0x006c CreateProc(name,pri,segList,stackSize)(D1/D2/D3/D4)
114 0xff8e -0x0072 Exit(returnCode)(D1)
120 0xff88 -0x0078 LoadSeg(fileName)(D1)
126 0xff82 -0x007e UnLoadSeg(segment)(D1)
132 0xff7c -0x0084 GetPacket(wait)(D1)
138 0xff76 -0x008a QueuePacket(packet)(D1)
144 0xff70 -0x0090 DeviceProc(name)(D1)
150 0xff6a -0x0096 SetComment(name,comment)(D1/D2)
156 0xff64 -0x009c SetProtection(name,mask)(D1/D2)
162 0xff5e -0x00a2 DateStamp(date)(D1)
168 0xff58 -0x00a8 Delay(timeout)(D1)
174 0xff52 -0x00ae WaitForChar(file,timeout)(D1/D2)
180 0xff4c -0x00b4 ParentDir(lock)(D1)
186 0xff46 -0x00ba IsInteractive(file)(D1)
192 0xff40 -0x00c0 Execute(string,file,file)(D1/D2/D3)


exec.lib.offsets


 30 0xffe2 -0x001e Supervisor()
 36 0xffdc -0x0024 ExitIntr()
 42 0xffd6 -0x002a Schedule()
 48 0xffd0 -0x0030 Reschedule()
 54 0xffca -0x0036 Switch()
 60 0xffc4 -0x003c Dispatch()
 66 0xffbe -0x0042 Exception()
 72 0xffb8 -0x0048 InitCode(startClass,version)(D0/D1)
 78 0xffb2 -0x004e InitStruct(initTable,memory,size)(A1/A2,D0)
 84 0xffac -0x0054 MakeLibrary(funcInit,structInit,libInit,dataSize,codeSize)
                          (A0/A1/A2,D0/D1)
 90 0xffa6 -0x005a MakeFunctions(target,functionArray,funcDispBase)(A0,A1,A2)
 96 0xffa0 -0x0060 FindResident(name)(A1)
102 0xff9a -0x0066 InitResident(resident,segList)(A1,D1)
108 0xff94 -0x006c Alert(alertNum,parameters)(D7,A5)
114 0xff8e -0x0072 Debug()
120 0xff88 -0x0078 Disable()
126 0xff82 -0x007e Enable()
132 0xff7c -0x0084 Forbid()
138 0xff76 -0x008a Permit()
144 0xff70 -0x0090 SetSR(newSR,mask)(D0/D1)
150 0xff6a -0x0096 SuperState()
156 0xff64 -0x009c UserState(sysStack)(D0)
162 0xff5e -0x00a2 SetIntVector(intNumber,interrupt)(D0/A1)
168 0xff58 -0x00a8 AddIntServer(intNumber,interrupt)(D0/A1)
174 0xff52 -0x00ae RemIntServer(intNumber,interrupt)(D0/A1)
180 0xff4c -0x00b4 Cause(interrupt)(A1)
186 0xff46 -0x00ba Allocate(freeList,byteSize)(A0,D0)
192 0xff40 -0x00c0 Deallocate(freeList,memoryBlock,byteSize)(A0/A1,D0)
198 0xff3a -0x00c6 AllocMem(byteSize,requirements)(D0/D1)
204 0xff34 -0x00cc AllocAbs(byteSize,location)(D0/A1)
210 0xff2e -0x00d2 FreeMem(memoryBlock,byteSize)(A1,D0)
216 0xff28 -0x00d8 AvailMem(requirements)(D1)
222 0xff22 -0x00de AllocEntry(entry)(A0)
228 0xff1c -0x00e4 FreeEntry(entry)(A0)
```

```
234 0xff16 -0x00ea Insert(list,node,pred)(A0/A1/A2)
240 0xff10 -0x00f0 AddHead(list,node)(A0/A1)
246 0xff0a -0x00f6 AddTail(list,node)(A0/A1)
252 0xff04 -0x00fc Remove(node)(A1)
258 0xfefe -0x0102 RemHead(list)(A0)
264 0xfef8 -0x0108 RemTail(list)(A0)
270 0xfef2 -0x010e Enqueue(list,node)(A0/A1)
276 0xfeec -0x0114 FindName(list,name)(A0/A1)
282 0xfee6 -0x011a AddTask(task,initPC,finalPC)(A1/A2/A3)
288 0xfee0 -0x0120 RemTask(task)(A1)
294 0xfeda -0x0126 FindTask(name)(A1)
300 0xfed4 -0x012c SetTaskPri(task,priority)(A1,D0)
306 0xfece -0x0132 SetSignal(newSignals,signalSet)(D0/D1)
312 0xfec8 -0x0138 SetExcept(newSignals,signalSet)(D0/D1)
318 0xfec2 -0x013e Wait(signalSet)(D0)
324 0xfebc -0x0144 Signal(task,signalSet)(A1,D0)
330 0xfeb6 -0x014a AllocSignal(signalNum)(D0)
336 0xfeb0 -0x0150 FreeSignal(signalNum)(D0)
342 0xfeaa -0x0156 AllocTrap(trapNum)(D0)
348 0xfea4 -0x015c FreeTrap(trapNum)(D0)
354 0xfe9e -0x0162 AddPort(port)(A1)
360 0xfe98 -0x0168 RemPort(port)(A1)
366 0xfe92 -0x016e PutMsg(port,message)(A0/A1)
372 0xfe8c -0x0174 GetMsg(port)(A0)
378 0xfe86 -0x017a ReplyMsg(message)(A1)
384 0xfe80 -0x0180 WaitPort(port)(A0)
390 0xfe7a -0x0186 FindPort(name)(A1)
396 0xfe74 -0x018c AddLibrary(library)(A1)
402 0xfe6e -0x0192 RemLibrary(library)(A1)
408 0xfe68 -0x0198 OldOpenLibrary(libName)(A1)
414 0xfe62 -0x019e CloseLibrary(library)(A1)
420 0xfe5c -0x01a4 SetFunction(library,funcOffset,funcEntry)(A1,A0,D0)
426 0xfe56 -0x01aa SumLibrary(library)(A1)
432 0xfe50 -0x01b0 AddDevice(device)(A1)
438 0xfe4a -0x01b6 RemDevice(device)(A1)
444 0xfe44 -0x01bc OpenDevice(devName,unit,ioRequest,flags)(A0,D0/A1,D1)
450 0xfe3e -0x01c2 CloseDevice(ioRequest)(A1)
456 0xfe38 -0x01c8 DoIO(ioRequest)(A1)
462 0xfe32 -0x01ce SendIO(ioRequest)(A1)
468 0xfe2c -0x01d4 CheckIO(ioRequest)(A1)
474 0xfe26 -0x01da WaitIO(ioRequest)(A1)
480 0xfe20 -0x01e0 AbortIO(ioRequest)(A1)
486 0xfe1a -0x01e6 AddResource(resource)(A1)
492 0xfe14 -0x01ec RemResource(resource)(A1)
498 0xfe0e -0x01f2 OpenResource(resName,version)(A1,D0)
504 0xfe08 -0x01f8 RawIOInit()
510 0xfe02 -0x01fe RawMayGetChar()
516 0xfdfc -0x0204 RawPutChar(char)(d0)
522 0xfdf6 -0x020a RawDoFmt()(A0/A1/A2/A3)
528 0xfdf0 -0x0210 GetCC()
534 0xfdea -0x0216 TypeOfMem(address)(A1);
540 0xfde4 -0x021c Procure(semaport,bidMsg)(A0/A1)
546 0xfdde -0x0222 Vacate(semaport)(A0)
552 0xfdd8 -0x0228 OpenLibrary(libName,version)(A1,D0)


graphics.lib.offsets

 30 0xffe2 -0x001e BltBitMap(srcBitMap,srcX,srcY,destBitMap,
                    destX,destY,sizeX,sizeY,minterm,mask,tempA)
                    (A0,D0/D1,A1,D2/D3/D4/D5/D6/D7/A2)
 36 0xffdc -0x0024 BltTemplate(source,srcX,srcMod,
                    destRastPort,destX,destY,sizeX,sizeY)
                    (A0,D0/D1/A1,D2/D3/D4/D5)
 42 0xffd6 -0x002a ClearEOL(rastPort)(A1)
 48 0xffd0 -0x0030 ClearScreen(rastPort)(A1)
 54 0xffca -0x0036 TextLength(RastPort,string,count)(A1,A0,D0)
 60 0xffc4 -0x003c Text(RastPort,string,count)(A1,A0,D0)
 66 0xffbe -0x0042 SetFont(RastPortID,textFont)(A1,A0)
 72 0xffb8 -0x0048 OpenFont(textAttr)(A0)
 78 0xffb2 -0x004e CloseFont(textFont)(A1)
 84 0xffac -0x0054 AskSoftStyle(rastPort)(A1)
 90 0xffa6 -0x005a SetSoftStyle(rastPort,style,enable)(A1,D0/D1)
 96 0xffa0 -0x0060 AddBob(bob,rastPort)(A0,A1)
102 0xff9a -0x0066 AddVSprite(vSprite,rastPort)(A0/A1)
108 0xff94 -0x006c DoCollision(rasPort)(A1)
114 0xff8e -0x0072 DrawGList(rastPort,viewPort)(A1,A0)
120 0xff88 -0x0078 InitGels(dummyHead,dummyTail,GelsInfo)(A0/A1/A2)
126 0xff82 -0x007e InitMasks(vSprite)(A0)
132 0xff7c -0x0084 RemIBob(bob,rastPort,viewPort)(A0/A1/A2)
138 0xff76 -0x008a RemVSprite(vSprite)(A0)
144 0xff70 -0x0090 SetCollision(type,routine,gelsInfo)(D0/A0/A1)
150 0xff6a -0x0096 SortGList(rastPort)(A1)
156 0xff64 -0x009c AddAnimOb(obj,animationKey,rastPort)(A0/A1/A2)
162 0xff5e -0x00a2 Animate(animationKey,rastPort)(A0/A1)
168 0xff58 -0x00a8 GetGBuffers(animationObj,rastPort,doubleBuffer)(A0/A1,D0)
174 0xff52 -0x00ae InitGMasks(animationObj)(A0)
180 0xff4c -0x00b4 GelsFuncE()
186 0xff46 -0x00ba GelsFuncF()
192 0xff40 -0x00c0 LoadRGB4(viewPort,colors,count)(A0/A1,D0)
198 0xff3a -0x00c6 InitRastPort(rastPort)(A1)
204 0xff34 -0x00cc InitVPort(viewPort)(A0)
210 0xff2e -0x00d2 MrgCop(view)(A1)
216 0xff28 -0x00d8 MakeVPort(view,viewPort)(A0/A1)
222 0xff22 -0x00de LoadView(view)(A1)
228 0xff1c -0x00e4 WaitBlit()
234 0xff16 -0x00ea SetRast(rastPort,color)(A1,D0)
240 0xff10 -0x00f0 Move(rastPort,x,y)(A1,D0/D1)
246 0xff0a -0x00f6 Draw(rastPort,x,y)(A1,D0/D1)
252 0xff04 -0x00fc AreaMove(rastPort,x,y)(A1,D0/D1)
258 0xfefe -0x0102 AreaDraw(rastPort,x,y)(A1,D0/D1)
264 0xfef8 -0x0108 AreaEnd(rastPort)(A1)
270 0xfef2 -0x010e WaitTOF()
276 0xfeec -0x0114 QBlit(blit)(A1)
282 0xfee6 -0x011a InitArea(areaInfo,vectorTable,vectorTableSize)(A0/A1,D0)
288 0xfee0 -0x0120 SetRGB4(viewPort,index,r,g,b)(A0,D0/D1/D2/D3)
294 0xfeda -0x0126 QBSBlit(blit)(A1)
300 0xfed4 -0x012c BltClear(memory,size,flags)(A1,D0/D1)
306 0xfece -0x0132 RectFill(rastPort,x1,y1,xu,yu)(A1,D0/D1/D2/D3)
312 0xfec8 -0x0138 BltPattern(rastPort,ras,x1,y1,maxX,maxY,fillBytes)
                    (a1,a0,D0/D1/D2/D3/D4)
318 0xfec2 -0x013e ReadPixel(rastPort,x,y)(A1,D0/D1)
324 0xfebc -0x0144 WritePixel(rastPort,x,y)(A1,D0/D1)
330 0xfeb6 -0x014a Flood(rastPort,mode,x,y)(A1,D2,D0/D1)
336 0xfeb0 -0x0150 PolyDraw(rastPort,count,polyTable)(A1,D0,A0)
342 0xfeaa -0x0156 SetAPen(rastPort,pen)(A1,D0)
348 0xfea4 -0x015c SetBPen(rastPort,pen)(A1,D0)
354 0xfe9e -0x0162 SetDrMd(rastPort,drawMode)(A1,D0)
360 0xfe98 -0x0168 InitView(view)(A1)
```

```
366 0xfe92 -0x016e CBump(copperList)(A1)
372 0xfe8c -0x0174 CMove(copperList,destination,data)(A1,D0/D1)
378 0xfe86 -0x017a CWait(copperList,x,y)(A1,D0/D1)
384 0xfe80 -0x0180 VBeamPos()
390 0xfe7a -0x0186 InitBitMap(bitMap,depth,width,height)(A0,D0/D1/D2)
396 0xfe74 -0x018c ScrollRaster(rastPort,dX,dY,minx,miny,maxx,maxy)
                          (A1,D0/D1/D2/D3/D4/D5)
402 0xfe6e -0x0192 WaitBOVP(viewport)(a0)
408 0xfe68 -0x0198 GetSprite(simplesprite,num)(a0,d0)
414 0xfe62 -0x019e FreeSprite(num)(d0)
420 0xfe5c -0x01a4 ChangeSprite(vp,simplesprite,data)(a0/a1/a2)
426 0xfe56 -0x01aa MoveSprite(viewport,simplesprite,x,y)(a0/a1,d0/d1)
432 0xfe50 -0x01b0 LockLayerRom(layer)(a5)
438 0xfe4a -0x01b6 UnlockLayerRom(layer)(a5)
444 0xfe44 -0x01bc SyncSBitMap(l)(a0)
450 0xfe3e -0x01c2 CopySBitMap(l1,l2)(a0/a1)
456 0xfe38 -0x01c8 OwnBlitter()()
462 0xfe32 -0x01ce DisownBlitter()()
468 0xfe2c -0x01d4 InitTmpRas(tmpras,buff,size)(a0/a1,d0)
474 0xfe26 -0x01da AskFont(rastPort,textAttr)(A1,A0)
480 0xfe20 -0x01e0 AddFont(textFont)(A1)
486 0xfe1a -0x01e6 RemFont(textFont)(A1)
492 0xfe14 -0x01ec AllocRaster(width,height)(D0/D1)
498 0xfe0e -0x01f2 FreeRaster(planeptr,width,height)(A0,D0/D1)
504 0xfe08 -0x01f8 AndRectRegion(rgn,rect)(A0/A1)
510 0xfe02 -0x01fe OrRectRegion(rgn,rect)(A0/A1)
516 0xfdfc -0x0204 NewRegion()()
522 0xfdf6 -0x020a **** Reserved for future use *****
528 0xfdf0 -0x0210 ClearRegion(rgn)(A0)
534 0xfdea -0x0216 DisposeRegion(rgn)(A0)
540 0xfde4 -0x021c FreeVPortCopLists(viewport)(a0)
546 0xfdde -0x0222 FreeCopList(coplist)(a0)
552 0xfdd8 -0x0228 ClipBlit(srcrp,srcX,srcY,
                        destrp,destX,destY,sizeX,sizeY,minterm)
                        (A0,D0/D1,A1,D2/D3/D4/D5/D6)
558 0xfdd2 -0x022e XorRectRegion(rgn,rect)(a0/a1)
564 0xfdcc -0x0234 FreeCprList(cprlist)(a0)
570 0xfdc6 -0x023a GetColorMap(entries)(d0)
576 0xfdc0 -0x0240 FreeColorMap(colormap)(a0)
582 0xfdba -0x0246 GetRGB4(colormap,entry)(a0,d0)
588 0xfdb4 -0x024c ScrollVPort(vp)(a0)
594 0xfdae -0x0252 UCopperListInit(copperlist,num)(a0,d0)
600 0xfda8 -0x0258 FreeGBuffers(animationObj,rastPort,doubleBuffer)(A0/A1,D0)
606 0xfda2 -0x025e BltBitMapRastPort(srcbm,srcx,srcy,destrp,destX,destY,
                                     sizeX,sizeY,minterm)
                                     (A0,D0/D1,A1,D2/D3/D4/D5/D6)


icon.lib.offsets


 30 0xffe2 -0x001e GetWBObject(name)(A0)
 36 0xffdc -0x0024 PutWBObject(name,object)(A0/A1)
 42 0xffd6 -0x002a GetIcon(name,icon,freelist)(A0/A1/A2)
 48 0xffd0 -0x0030 PutIcon(name,icon)(A0/A1)
 54 0xffca -0x0036 FreeFreeList(freelist)(A0)
 60 0xffc4 -0x003c FreeWBObject(WBObject)(A0)
 66 0xffbe -0x0042 AllocWBObject()()
 72 0xffb8 -0x0048 AddFreeList(freelist,mem,size)(A0/A1/A2)
 78 0xffb2 -0x004e GetDiskObject(name)(A0)
```

```
 84 0xffac -0x0054 PutDiskObject(name,diskobj)(A0,A1)
 90 0xffa6 -0x005a FreeDiskObject(diskobj)(A0)
 96 0xffa0 -0x0060 FindToolType(toolTypeArray,typeName)(A0/A1)
102 0xff9a -0x0066 MatchToolValue(typeString,value)(A0/A1)
108 0xff94 -0x006c BumpRevision(newname,oldname)(A0/A1)


intuition.lib.offsets


 30 0xffe2 -0x001e OpenIntuition()()
 36 0xffdc -0x0024 Intuition(ievent)(A0)
 42 0xffd6 -0x002a AddGadget(AddPtr,Gadget,Position)(A0/A1,D0)
 48 0xffd0 -0x0030 ClearDMRequest(Window)(A0)
 54 0xffca -0x0036 ClearMenuStrip(Window)(A0)
 60 0xffc4 -0x003c ClearPointer(Window)(A0)
 66 0xffbe -0x0042 CloseScreen(Screen)(A0)
 72 0xffb8 -0x0048 CloseWindow(Window)(A0)
 78 0xffb2 -0x004e CloseWorkBench()()
 84 0xffac -0x0054 CurrentTime(Seconds,Micros)(A0/A1)
 90 0xffa6 -0x005a DisplayAlert(AlertNumber,String,Height)(D0/A0,D1)
 96 0xffa0 -0x0060 DisplayBeep(Screen)(A0)
102 0xff9a -0x0066 DoubleClick(sseconds,smicros,cseconds,cmicros)(D0/D1/D2/D3)
108 0xff94 -0x006c DrawBorder(RPort,Border,LeftOffset,TopOffset)(A0/A1,D0/D1)
114 0xff8e -0x0072 DrawImage(RPort,Image,LeftOffset,TopOffset)(A0/A1,D0/D1)
120 0xff88 -0x0078 EndRequest(requester,window)(A0/A1)
126 0xff82 -0x007e GetDefPrefs(preferences,size)(A0,D0)
132 0xff7c -0x0084 GetPrefs(preferences,size)(A0,D0)
138 0xff76 -0x008a InitRequester(req)(A0)
144 0xff70 -0x0090 ItemAddress(MenuStrip,MenuNumber)(A0,D0)
150 0xff6a -0x0096 ModifyIDCMP(Window,Flags)(A0,D0)
156 0xff64 -0x009c ModifyProp(Gadget,Ptr,Req,Flags,HPos,VPos,HBody,VBody)
                          (A0/A1/A2,D0/D1/D2/D3/D4)
162 0xff5e -0x00a2 MoveScreen(Screen,dx,dy)(A0,D0/D1)
168 0xff58 -0x00a8 MoveWindow(window,dx,dy)(A0,D0/D1)
174 0xff52 -0x00ae OffGadget(Gadget,Ptr,Req)(A0/A1/A2)
180 0xff4c -0x00b4 OffMenu(Window,MenuNumber)(A0,D0)
186 0xff46 -0x00ba OnGadget(Gadget,Ptr,Req)(A0/A1/A2)
192 0xff40 -0x00c0 OnMenu(Window,MenuNumber)(A0,D0)
198 0xff3a -0x00c6 OpenScreen(OSargs)(A0)
204 0xff34 -0x00cc OpenWindow(OWargs)(A0)
210 0xff2e -0x00d2 OpenWorkBench()()
216 0xff28 -0x00d8 PrintIText(rp,itext,left,top)(A0/A1,D0/D1)
222 0xff22 -0x00de RefreshGadgets(Gadgets,Ptr,Req)(A0/A1/A2)
228 0xff1c -0x00e4 RemoveGadget(RemPtr,Gadget)(A0/A1)
234 0xff16 -0x00ea ReportMouse(Window,Boolean)(A0/D0)
240 0xff10 -0x00f0 Request(Requester,Window)(A0/A1)
246 0xff0a -0x00f6 ScreenToBack(Screen)(A0)
252 0xff04 -0x00fc ScreenToFront(Screen)(A0)
258 0xfefe -0x0102 SetDMRequest(Window,req)(A0/A1)
264 0xfef8 -0x0108 SetMenuStrip(Window,Menu)(A0/A1)
270 0xfef2 -0x010e SetPointer(Window,Pointer,Height,Width,Xoffset,Yoffset)
                          (A0/A1,D0/D1/D2/D3)
276 0xfeec -0x0114 SetWindowTitles(window,windowtitle,screentitle)(A0/A1/A2)
282 0xfee6 -0x011a ShowTitle(Screen,ShowIt)(A0,D0)
288 0xfee0 -0x0120 SizeWindow(window,dx,dy)(A0,D0/D1)
294 0xfeda -0x0126 ViewAddress()()
300 0xfed4 -0x012c ViewPortAddress(window)(A0)
306 0xfece -0x0132 WindowToBack(window)(A0)
```

```
312 0xfec8 -0x0138 WindowToFront(window)(A0)
318 0xfec2 -0x013e WindowLimits(window,minwidth,minheight,maxwidth,maxheight)
                          (A0,D0/D1/D2/D3)
324 0xfebc -0x0144 SetPrefs(preferences,size,flag)(A0,D0/D1)
330 0xfeb6 -0x014a IntuiTextLength(itext)(A0)
336 0xfeb0 -0x0150 WBenchToBack()()
342 0xfeaa -0x0156 WBenchToFront()()
348 0xfea4 -0x015c AutoRequest(Window,Body,PText,NText,PFlag,NFlag,W,H)
                          (A0,A1,A2,A3,D0,D1,D2,D3)
354 0xfe9e -0x0162 BeginRefresh(Window)(A0)
360 0xfe98 -0x0168 BuildSysRequest(Window,Body,PosText,NegText,Flags,W,H)
                          (A0,A1,A2,A3,D0,D1,D2)
366 0xfe92 -0x016e EndRefresh(Window,Complete)(A0,D0)
372 0xfe8c -0x0174 FreeSysRequest(Window)(A0)
378 0xfe86 -0x017a MakeScreen(Screen)(A0)
384 0xfe80 -0x0180 RemakeDisplay()()
390 0xfe7a -0x0186 RethinkDisplay()()
396 0xfe74 -0x018c AllocRemember(RememberKey,Size,Flags)(A0,D0,D1)
402 0xfe6e -0x0192 AlohaWorkbench(wbport)(A0)
408 0xfe68 -0x0198 FreeRemember(RememberKey,ReallyForget)(A0,D0)
414 0xfe62 -0x019e LockIBase(dontknow)(D0)
420 0xfe5c -0x01a4 UnlockIBase(IBLock)(A0)
```

layers.lib.offsets

```
 30 0xffe2 -0x001e InitLayers(li)(A0)
 36 0xffdc -0x0024 CreateUpfrontLayer(li,bm,x0,y0,x1,y1,flags,bm2)
                          (A0/A1,D0/D1/D2/D3/D4,A2)
 42 0xffd6 -0x002a CreateBehindLayer(li,bm,x0,y0,x1,y1,flags,bm2)
                          (A0/A1,D0/D1/D2/D3/D4,A2)
 48 0xffd0 -0x0030 UpfrontLayer(li,layer)(A0/A1)
 54 0xffca -0x0036 BehindLayer(li,layer)(A0/A1)
 60 0xffc4 -0x003c MoveLayer(li,layer,dx,dy)(A0/A1,D0/D1)
 66 0xffbe -0x0042 SizeLayer(li,layer,dx,dy)(A0/A1,D0/D1)
 72 0xffb8 -0x0048 ScrollLayer(li,layer,dx,dy)(A0/A1,D0/D1)
 78 0xffb2 -0x004e BeginUpdate(layer)(A0)
 84 0xffac -0x0054 EndUpdate(layer,flag)(A0,d0)
 90 0xffa6 -0x005a DeleteLayer(li,layer)(A0/A1)
 96 0xffa0 -0x0060 LockLayer(li,layer)(A0/A1)
102 0xff9a -0x0066 UnlockLayer(layer)(A0)
108 0xff94 -0x006c LockLayers(li)(A0)
114 0xff8e -0x0072 UnlockLayers(li)(A0)
120 0xff88 -0x0078 LockLayerInfo(li)(A0)
126 0xff82 -0x007e SwapBitsRastPortClipRect(rp,cr)(A0/A1)
132 0xff7c -0x0084 WhichLayer(li,x,y)(a0,d0/d1)
138 0xff76 -0x008a UnlockLayerInfo(li)(A0)
144 0xff70 -0x0090 NewLayerInfo()()
150 0xff6a -0x0096 DisposeLayerInfo(li)(a0)
156 0xff64 -0x009c FattenLayerInfo(li)(a0)
162 0xff5e -0x00a2 ThinLayerInfo(li)(a0)
168 0xff58 -0x00a8 MoveLayerInFrontOf(layer_to_move,layer_to_be_infront_of)
                          (a0/a1)
```

mathffp.lib.offsets

```
 30 0xffe2 -0x001e SPFix(float)(D0)
 36 0xffdc -0x0024 SPFlt(integer)(D0)
 42 0xffd6 -0x002a SPCmp(leftFloat,rightFloat)(D1,D0)
 48 0xffd0 -0x0030 SPTst(float)(D1)
 54 0xffca -0x0036 SPAbs(float)(D0)
 60 0xffc4 -0x003c SPNeg(float)(D0)
 66 0xffbe -0x0042 SPAdd(leftFloat,rightFloat)(D1,D0)
 72 0xffb8 -0x0048 SPSub(leftFloat,rightFloat)(D1,D0)
 78 0xffb2 -0x004e SPMul(leftFloat,rightFloat)(D1,D0)
 84 0xffac -0x0054 SPDiv(leftFloat,rightFloat)(D1,D0)
```

mathieeedoubbas.lib.offsets

```
 30 0xffe2 -0x001e IEEEDPFix(integer,integer)(D0,D1)
 36 0xffdc -0x0024 IEEEDPFlt(integer)(D0)
 42 0xffd6 -0x002a IEEEDPCmp(integer,integer,integer,integer)(D0,D1,D2,D3)
 48 0xffd0 -0x0030 IEEEDPTst(integer,integer)(D0,D1)
 54 0xffca -0x0036 IEEEDPAbs(integer,integer)(D0,D1)
 60 0xffc4 -0x003c IEEEDPNeg(integer,integer)(D0,D1)
 66 0xffbe -0x0042 IEEEDPAdd(integer,integer,integer,integer)(D0,D1,D2,D3)
 72 0xffb8 -0x0048 IEEEDPSub(integer,integer,integer,integer)(D0,D1,D2,D3)
 78 0xffb2 -0x004e IEEEDPMul(integer,integer,integer,integer)(D0,D1,D2,D3)
 84 0xffac -0x0054 IEEEDPDiv(integer,integer,integer,integer)(D0,D1,D2,D3)
```

mathtrans.lib.offsets

```
 30 0xffe2 -0x001e SPAtan(float)(D0)
 36 0xffdc -0x0024 SPSin(float)(D0)
 42 0xffd6 -0x002a SPCos(float)(D0)
 48 0xffd0 -0x0030 SPTan(float)(D0)
 54 0xffca -0x0036 SPSincos(leftFloat,rightFloat)(D1,D0)
 60 0xffc4 -0x003c SPSinh(float)(D0)
 66 0xffbe -0x0042 SPCosh(float)(D0)
 72 0xffb8 -0x0048 SPTanh(float)(D0)
 78 0xffb2 -0x004e SPExp(float)(D0)
 84 0xffac -0x0054 SPLog(float)(D0)
 90 0xffa6 -0x005a SPPow(leftFloat,rightFloat)(D1,D0)
 96 0xffa0 -0x0060 SPSqrt(float)(D0)
102 0xff9a -0x0066 SPTieee(float)(D0)
108 0xff94 -0x006c SPFieee(integer)(D0)
114 0xff8e -0x0072 SPAsin(float)(D0)
120 0xff88 -0x0078 SPAcos(float)(D0)
126 0xff82 -0x007e SPLog10(float)(D0)
```

potgo.lib.offsets

```
  6 0xfffa -0x0006 AllocPotBits(bits)(D0)
 12 0xfff4 -0x000c FreePotBits(bits)(D0)
 18 0xffee -0x0012 WritePotgo(word,mask)(D0,D1)
```

timer.lib.offsets

```
 42 0xffd6 -0x002a AddTime(dest,src)(A0/A1)
 48 0xffd0 -0x0030 SubTime(dest,src)(A0/A1)
 54 0xffca -0x0036 CmpTime(dest,src)(A0/A1)


translator.lib.offsets


 30 0xffe2 -0x001e Translate(inputString,inputLength,outputBuffer,bufferSize)
                                (A0,D0/A1,D1)
```

Index

> $24.95 FPT USA

Amiga™ Technical Reference Series

Amiga ROM Kernel Reference Manual: Exec

The Amiga Computer is an exciting new high-performance microcomputer with superb graphics, sound, and multitasking capabilities. Its technologically advanced hardware, designed around the Motorola 68000 microprocessor, includes three sophisticated custom chips that control graphics, audio, and peripherals. The Amiga's unique system software is contained in 192K of read-only memory (ROM), providing programmers with unparalleled power, flexibility, and convenience in designing and creating programs.

The AMIGA ROM KERNEL REFERENCE MANUAL: *Exec,* written by the technical staff at Commodore-Amiga, Inc., is a detailed description of Exec, the ROM-based multitasking system executive that controls the Amiga's underlying system execution environment. It includes:

- an overview of the capabilities of Exec
- key descriptions of how Exec handles multitasking and inter-task communication
- explanations of Exec's exception handling and memory allocation
- a complete listing of the ROM routines that make up Exec

For the serious programmer working in assembly language, C, or Pascal who wants to take full advantage of the Amiga's impressive capabilities, the AMIGA ROM KERNEL REFERENCE MANUAL: *Exec* is an essential reference.

Written by the technical staff at Commodore-Amiga, Inc., who designed the Amiga's hardware and system software, the AMIGA ROM KERNEL REFERENCE MANUAL: *Exec* is the definitive source of information on the system execution environment of this revolutionary microcomputer.

Cover design by Marshall Henrichs
Cover photograph by Jack Haeger

Addison-Wesley Publishing Company, Inc.

ISBN 0-201-11099-7