Devices

COLLABORATORS			
	TITLE : Devices		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 28, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Devices **1**

1.1 Amiga® RKM Devices: 15 Resources 1

1.2 15 Resources / The Amiga Resources 1

1.3 15 Resources / Resource Interface 2

1.4 15 Resources / BattClock Resource 3

1.5 15 Resources / BattMem Resource 4

1.6 15 Resources / CIA Resource 5

1.7 15 Resources / Disk Resource 7

1.8 15 Resources / FileSystem Resource 9

1.9 15 Resources / Misc Resource 9

1.10 15 / Misc Resource / C Example Of Allocating Misc Resources 10

1.11 15 Resources / Potgo Resource 11

Chapter 1

Devices

1.1 Amiga® RKM Devices: 15 Resources

The Amiga's low-level hardware control functions are collectively referred to as "Resources". Most applications will never need to use the hardware at the resource level - the Amiga's device interface is much more convenient and provides for multitasking. However, some high performance applications, such as MIDI time stamping, may require direct access to the Amiga hardware registers.

NEW FEATURES FOR VERSION 2.0

Feature	Description
BattClock	New resource
BattMem	New resource
FileSystem	New resource

Compatibility Warning:

The new features for 2.0 are not backwards compatible.

The Amiga Resources	BattMem Resource	FileSystem Resource
Resource Interface	CIA Resource	Misc Resource
BattClock Resource	Disk Resource	Potgo Resource

1.2 15 Resources / The Amiga Resources

There are currently seven standard resources in the Amiga system. The following lists the name of each resource and its function.

battclock.resource
grants access to the battery-backed clock chip.

battmem.resource
grants access to non-volatile RAM.

cia.resource

```
   grants access to the interrupts and timer bits of the 8520 CIA
   (Complex Interface Adapter) chips.

disk.resource
   grants temporary exclusive access to the disk hardware.

FileSystem.resource
   grants access to the file system.

misc.resource
   grants exclusive access to functional blocks of chip registers.  At
   present, definitions have been made for the serial and parallel
   hardware only.

potgo.resource
   manages the bits of the proportional I/O pins on the game controller
   ports.

The resources allow you direct access to the hardware in a way that is
compatible with multitasking.  They also allow you to temporarily bar
other tasks from using the resource.  You may then use the associated
hardware directly for your special purposes.  If applicable, you must
return the resource back to the system for other tasks to use when you are
finished with it.

See the Amiga Hardware Reference Manual for detailed information on the
actual hardware involved.

   Look Before You Leap.
   ---------------------
   Resources are just one step above direct hardware manipulation. You
   are advised to try the higher level device and library approach
   before resorting to the hardware.
```

1.3 15 Resources / Resource Interface

```
Resources provide functions that you call to do low-level operations with
the hardware they access.  In order to use the functions of a resource,
you must obtain a pointer to the resource.  This is done by calling the
OpenResource() function with the resource name as its argument.

OpenResource() returns a pointer to the resource you request or NULL
if it does not exist.

    #include <resources/filesysres.h>

    struct FileSysResource *FileSysResBase = NULL;

    if (!(FileSysResBase = OpenResource(FSRNAME)))
        printf("Cannot open %s\n",FSRNAME);

There is no CloseResource() function.  When you are done with a resource,
you are done with it.  However, as you will see later in this chapter,
some resources provide functions to allocate parts of the hardware they
access. In those cases, you will have to free those parts for anyone else
```

to use them.

Each resource has at least one include file in the resources subdirectory of the include directory. Some of the include files contain only the name of the resource; others list structures and bit definitions used by the resource. The include files will be listed at the end of this chapter.

Calling a resource function is the same as calling any other function on the Amiga. You have to know what parameters it accepts and the return value, if any. The Autodocs for each resource lists the functions and their requirements.

```
#include <hardware/cia.h>
#include <resources/cia.h>

struct Library *CIAResource = NULL;

void main()
{

WORD mask = 0;

if (!(CIAResource = OpenResource(CIABNAME)))
    printf("Cannot open %s\n",CIABNAME);
else
   {
    /* What is the interrupt enable mask? */
     mask = AbleICR(CIAResource,0);

    printf("\nThe CIA interrupt enable mask: %x \n",mask);
   }
}
```

Looks Can Be Deceiving.

Some resources may look like libraries and act like libraries, but be assured they are not libraries.

1.4 15 Resources / BattClock Resource

The battery-backed clock (BattClock) keeps Amiga time while the system is powered off. The time from the BattClock is loaded into the Amiga system clock as part of the boot sequence.

The battclock resource provides access to the BattClock. Three functions allow you to read the BattClock value, reset it and set it to a value you desire.

	BattClock Resource Functions
ReadBattClock()	Read the time from the BattClock and returns it as the number of seconds since 12:00 AM, January 1, 1978.
ResetBattClock()	Reset the BattClock to 12:00 AM, January 1,

	1978.
WriteBattClock()	Set the BattClock to the number of seconds you pass it relative to 12:00 AM, January 1, 1978.

The utility.library contains time functions which convert the number of seconds since 12:00 AM, January 1, 1978 to a date and time we can understand, and vice versa. You will find these functions useful when dealing with the BattClock. The example program below uses the Amiga2Date() utility function to convert the value returned by ReadBattClock(). See the "Utility Library" chapter of the Amiga ROM Kernel Reference Manual: Libraries for a discussion of the utility.library and the Amiga ROM Kernel Reference Manual: Includes and Autodocs for a listing of its functions.

So, You Want to Be A Time Lord?

This resource will allow you to set the BattClock to any value you desire. Keep in mind that this time will endure a reboot and could adversely affect your system.

Read_BattClock.c

Additional programming information on the battclock resource can be found in the include files and the Autodocs for the battclock resource and the utility library.

1.5 15 Resources / BattMem Resource

The battery-backed memory (BattMem) preserves a small portion of Amiga memory while the system is powered off. Some of the information stored in this memory is used during the system boot sequence.

The battmem resource provides access to the BattMem. Four functions allow you to use the BattMem.

BattMemResource Functions	
ObtainBattSemaphore()	Obtain exclusive access to the BattMem.
ReadBattMem()	Read a bitstring from the BattMem. You specify the bit position and the number of bits you wish to read.
ReleaseBattSemaphore()	Relinquish exclusive access to the BattMem.
WriteBattMem()	Write a bitstring to the BattMem. You specify the bit position and the number of bits you wish to write.

The system considers BattMem to be a set of bits rather than bytes. This is done to conserve the limited space available. All bits are reserved, and applications should not read, or write undefined bits. Writing bits should be done with extreme caution since the settings will survive

power-down/power-up. You can find the bit definitions in the BattMem include files resources/battmembitsamiga.h, resources/battmembitsamix.h and resources/battmembitsshared.h. They should be consulted before you do anything with the resource.

You Don't Need This Resource.

The BattMem resource is basically for system use only. There is generally no reason for applications to use it. It is documented here simply for completeness.

Additional information on the battmem resource can be found in the include files and the Autodocs for the battmem resource.

BattMem Resource Information

INCLUDES	resources/battmem.i resources/battmembitsamiga.h resources/battmembitsamix.h resources/battmembitsshared.h
AUTODOCS	battmem.doc

1.6 15 Resources / CIA Resource

The CIA resource provides access to the timers and timer interrupt bits of the 8520 Complex Interface Adapter (CIA) A and B chips. This resource is intended for use by high performance timing applications such as MIDI time stamping and SMPTE time coding.

Four functions allow you to interact with the CIA hardware.

CIA Resource Functions

AbleICR()	Enable or disable Interrupt Control Register interrupts. Can also return the current or previous enabled interrupt mask.
AddICRVector()	Allocate one of the CIA timers by assigning an interrupt handler to an interrupt bit and enabling the interrupt of one of the timers. If the timer you request is not available, a pointer to the interrupt structure that owns it will be returned.
RemICRVector()	Remove an interrupt handler from an interrupt bit and disable the interrupt.
SetICR()	Cause or clear one or more interrupts, or return the current or previous interrupt status.

Each CIA chip has two interval timers within it - Timer A and Timer B - that may be available. The CIA chips operate at different interrupt levels with the CIA-A timers at interrupt level 2 and the CIA-B timers at interrupt level 6.

```
   Choose A Timer Wisely.
   ----------------------
   The timer you use should be based solely on interrupt level and
   availability. If the timer you request is not available, try for
   another.  Whatever you do, do not base your decision on what you
   think the timer is used for by the system.
```

You allocate a timer by calling AddICRVector(). This is the only way you should access a timer. If the function returns zero, you have successfully allocated that timer. If it is unavailable, the owner interrupt will be returned.

```
   /* allocate CIA-A Timer A */
   inta = AddICRVector (CIAResource, CIAICRB_TA, &tint);

   if (inta)  /* if allocate was not successful */
       printf("Error: Could not allocate timer\n");
   else
       {
       ...ready for timing
       }
```

The timer is deallocated by calling RemICRVector(). This is the only way you should deallocate a timer.

```
   RemICRVector(CIAResource, CIAICRB_TA, &tint);
```

Your application should not make any assumptions regarding which interval timers (if any) are available for use; other tasks or critical operating system routines may be using the interval timers. In fact, in the latest version of the operating system, the timer device may dynamically allocate one of the interval timers.

```
   Time Is Of The Essence!
   -----------------------
   There are a limited number of free CIA interval timers.  Applications
   which use the interval timers may not be able to run at the same time
   if all interval timers are in use.  As a general rule, you should use
   the timer device for most interval timing.
```

You read from and write to the CIA interrupt control registers using SetICR() and AbleICR(). SetICR() is useful for sampling which cia interrupts (if any) are active. It can also be used to clear and generate interrupts. AbleICR() is used to disable and enable a particular CIA interrupt. Additional information about these functions can be found in the Amiga ROM Kernel Reference Manual: Includes and Autodocs.

Things to keep in mind:

1. Never directly read from or write to the CIA interrupt control registers. Always use SetICR() and AbleICR().

2. Your interrupt routine will be called with a pointer to your data area in register A1, and a pointer to the code being called in register A5. No other registers are set up for you. You must observe the standard convention of preserving all registers except D0-D1 and

A0-A1.

3. Never turn off all level 2 or level 6 interrupts. The proper way to disable interrupts for an interval timer that you've successfully allocated is via the AbleICR() function.

4. Interrupt handling code should be written in assembly code and, if possible, should signal a task to do most of the work.

5. Do not make assumptions about which CIA interval timers (if any) are available for use. The only proper way to own an interval timer is via the AddICRVector() function.

6. Do not use SetICR(), AbleICR() and RemICRVector() to affect timers or other CIA hardware which your task does not own.

Changes in the CIA resource:

* In pre-V36 versions of the operating system, SetICR() could return FALSE for a particular interrupt just prior to processing the interrupt. SetICR() now returns TRUE for a particular interrupt until sometime after the interrupt has been processed.

* Applications which only need to read a CIA interval timer should use the ReadEClock() function of the timer device. See "Timer Device" chapter of this manual for more information on ReadEClock().

* The timer device may dynamically allocate a free CIA interval timer. Do not make any assumptions regarding which interval timers are in use unless you are taking over the machine completely.

Cia_Interval.c

Additional programming information on the CIA resource can be found in the include files and the Autodocs for the CIA resource and the 8520 spec. The includes files and Autodocs are in the Amiga ROM Kernel Reference Manual: Includes and Autodocs and the 8520 spec is in the Amiga Hardware Reference Manual.

CIA Resource Information

INCLUDES	resources/cia.h resources/cia.i hardware/cia.h hardware/cia.i
AUTODOCS	cia.doc
HARDWARE	8520 specification

1.7 15 Resources / Disk Resource

The Disk resource obtains exclusive access to the floppy disk hardware There are four disk/MFM units available, units 0-3.

Six functions are available for dealing with the floppy disk hardware.

	Disk Resource Functions
AllocUnit()	Allocate one of the units of the disk resource.
FreeUnit()	Deallocate an allocated disk unit.
GetUnit()	Allocate the disk for a driver.
GetUnitID()	Return the drive ID of a specified drive unit.
GiveUnit()	Free the disk.
ReadUnitID()	Reread and return the drive ID of a specified unit.

The disk resource provides both a gross and a fine unit allocation scheme. AllocUnit() and FreeUnit() are used to claim a unit for long term use, and GetUnit() and GiveUnit() are used to claim a unit and the disk hardware for shorter periods.

The trackdisk device uses and abides by both allocation schemes. Because a trackdisk unit is never closed for Amiga 3.5" drives (the file system keeps them open) the associated resource units will always be allocated for these drives. GetUnit() and GiveUnit() can still be used, however, by other applications that have not succeeded with AllocUnit().

You must not change the state of of a disk that the trackdisk device is using unless you either

a) force its removal before giving it up, or

b) return it to the original track (with no changes to the track), or

c) CMD_STOP the unit before GetUnit(), update the current track number and CMD_START it after GiveUnit(). This option is only available under V36 and higher versions of the operating system.

ReadUnitID() is provided to handle drives which use the unit number in a dynamic manner. Subsequent GetUnit() calls will return the value obtained by ReadUnitID().

It is therefore possible to prevent the trackdisk device from using units that have not yet been mounted by successfully performing an AllocUnit() for that unit. It is also possible to starve trackdisk usage by performing a GetUnit(). The appropriate companion routine (FreeUnit() or GiveUnit()) should be called to restore the resource at the end of its use.

Get_Disk_Unit_ID.c

Additional programming information on the disk resource can be found in the include files and the Autodocs for the disk resource.

Disk Resource Information

INCLUDES	resources/disk.h resources/disk.i
AUTODOCS	disk.doc

1.8 15 Resources / FileSystem Resource

The FileSystem resource returns the filesystems that are available on the Amiga. It has no functions. Opening the FileSystem resource returns a pointer to a List structure containing the current filesystems in the Amiga.

```
Get_Filesys.c
```

Additional programming information on the FileSystem resource can be found in the include files and the Autodocs for the FileSystem resource in the Amiga ROM Kernel Reference Manual: Includes and Autodocs and the "Expansion" chapter of the Amiga ROM Kernel Reference Manual: Libraries.

FileSystem Resource Information

INCLUDES	resources/filesysres.h resources/filesysres.i
AUTODOCS	filesysres.doc
LIBRARIES	expansion library

1.9 15 Resources / Misc Resource

The misc resource oversees usage of the serial data port, the serial communication bits, the parallel data and handshake port, and the parallel communication bits. Before using serial or parallel port hardware, it first must be acquired from the misc resource.

The misc resource provides two functions for allocating and freeing the serial and parallel hardware.

Misc Resource Functions

AllocMiscResource()	Allocate one of the serial or parallel misc resources.
FreeMiscResource()	Deallocate one of the serial or parallel misc resources.

Once you've successfully allocated one of the misc resources, you are free to write directly to its hardware locations. Information on the serial and parallel hardware can be found in the Amiga Hardware Reference Manual and the hardware/custom.h include file.

The two examples below are assembly and C versions of the same code for locking the serial misc resources and waiting for CTRL-C to be pressed before releasing them.

Assembly Example Of Allocating Misc Resources
C Example Of Allocating Misc Resources

1.10 15 / Misc Resource / C Example Of Allocating Misc Resources

Allocate_Misc.c

The example below will try to open the serial device and execute the SDCMD_QUERY command. If it cannot open the serial device, it will do an AllocMiscResource() on the serial port and return the name of the owner.

Query_Serial.c

Take Over Everything.

There are two serial.device resources to take over, MR_SERIALBITS and MR_SERIALPORT. You should get both resources when you take over the serial port to prevent other tasks from using them. The parallel.device also has two resources to take over. See the resources/misc.h include file for the relevant definitions and structures.

Under V1.3 and earlier versions of the Amiga system software the MR_GETMISCRESOURCE routine will always fail if the serial device has been used at all by another task (even if that task has finished using the resource. In other words, once a printer driver or communication package has been activated, it will keep the associated resource locked up preventing your task from using it. Under these conditions, you must get the resource back from the system yourself.

You do this by calling the function FlushDevice() as follows:

```
/*
 * A safe way to expunge ONLY a certain device.  The serial.device holds
 * on to the misc serial resource until a general expunge occurs.
 * This code attempts to flush ONLY the named device out of memory and
 * nothing else.  If it fails, no status is returned since it would have
 * no valid use after the Permit().
 */

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

#include <clib/exec_protos.h>

void FlushDevice(char *);

extern struct ExecBase *SysBase;

void FlushDevice(char *name)
{
```

```
struct Device *devpoint;

Forbid();

if (devpoint=(struct Device *)FindName(&SysBase->DeviceList,name) )
    RemDevice(devpoint);

Permit();
}
```

Additional programming information on the misc resource can be found in the include files and the Autodocs for the misc resource.

Misc Resource Information

INCLUDES	resources/misc.h
	resources/misc.i
	hardware/custom.h
	hardware/custom.i
AUTODOCS	misc.doc

1.11 15 Resources / Potgo Resource

The potgo resource is used to get control of the hardware POTGO register connected to the proportional I/O pins on the game controller ports. There are two registers, POTGO (write-only) and POTINP (read-only). These pins could also be used for digital I/O.

The potgo resource provides three functions for working with the POTGO hardware.

Potgo Resource Functions

AllocPotBits()	Allocate bits in the POTGO register.
FreePotBits()	Free previously allocated bits in the POTGO register.
WritePotgo()	Set and clear bits in the POTGO register. The bits must have been allocated before calling this function.

The example program shown below demonstrates how to use the ptogo resource to track mouse button presses on port 1.

Read_Potinp.c

Additional programming information on the potgo resource can be found in the include files and the Autodocs for the potgo resource.

Potgo Resource Information

INCLUDES	resources/potgo.h

```
                    resources/potgo.i
                    utility/hooks.h
                    utility/hooks.i

AUTODOCS            potgo.doc
```