Parameter RAM Utilities

This chapter describes how your application can access and modify the information used by the system software at system startup time. Various user settings, such as the volume setting for the built-in speaker, need to be present at the next system startup. This startup information is stored in battery-powered parameter RAM, located in the computer's real-time clock chip. The Parameter RAM Utilities available in the Macintosh Operating System allow you to manipulate startup information stored in parameter RAM.

Because you can use Toolbox routines to indirectly access most of the useful information stored in parameter RAM, you should not need to use the utility routines described in this chapter. However, if you should discover some important need to directly manipulate the startup information in parameter RAM, you can use the Parameter RAM Utilities routines.

To use this chapter, you should already understand how to read and change the values of low-memory global variables. See the chapter "Memory Manager" in *Inside Macintosh: Memory* for a discussion on how to read and write system global variables.

This chapter

- introduces the kinds of information stored in parameter RAM
- describes some of the values stored in parameter RAM

About Parameter RAM

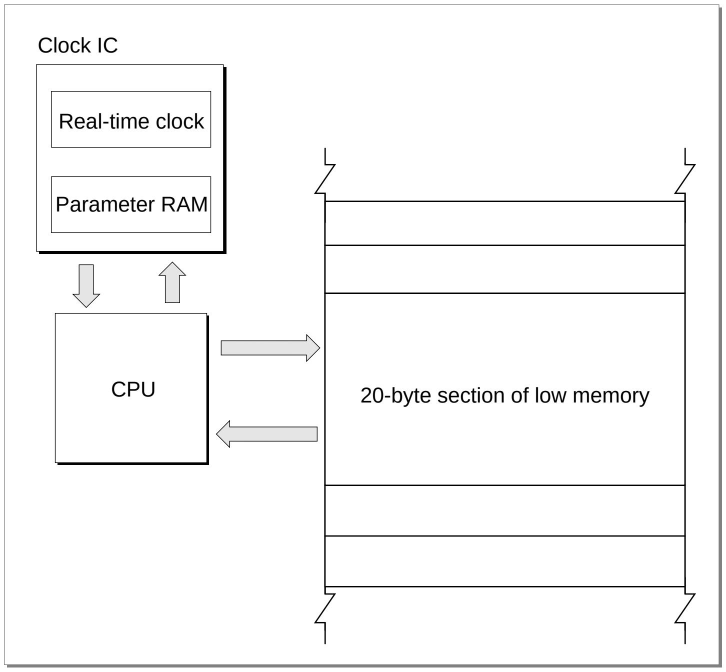
Most user settings that need to be present at system startup are stored in ***parameter RAM***. Parameter RAM takes up 256 bytes of battery-powered RAM: 20 bytes are documented in this chapter, and 236 bytes are reserved by the system software. The 236 bytes of parameter RAM are also known as ***extended parameter RAM*** . The parameter RAM is located in the computer's real-time clock chip, together with the date and time setting. No matter what system disk is used at system startup, parameter RAM ensures that certain settings remain the same on a given computer from one session to another.

Much of the information stored in parameter RAM is used exclusively by the system software. For example, system software uses 2 bits of parameter RAM to keep track of how many times menu items should blink after being selected. Other values stored in parameter RAM are useful to applications. For example, parameter RAM stores the suggested time interval that your application should use when determining whether two mouse clicks constitute a double-click. You can access this double-click time indirectly by using the Toolbox Event Manager's `GetDblTime` function. Whenever possible, you should use Toolbox routines to access parameter RAM values.

▲ **WARNING**
The operating-system routines described in this chapter let you directly manipulate values in parameter RAM; however, because the organization of parameter RAM is subject to change, you should rarely use them. Instead, use the appropriate Toolbox routines to *indirectly* manipulate values in parameter RAM. ▲

The 20 bytes of parameter RAM that are commonly accessible by applications are copied into low memory at system startup. Figure 7-1 illustrates the interaction between parameter RAM and low memory. Parameter RAM is read into low memory at system startup, and any modifications to this low-memory copy of parameter RAM are written back to the clock chip.

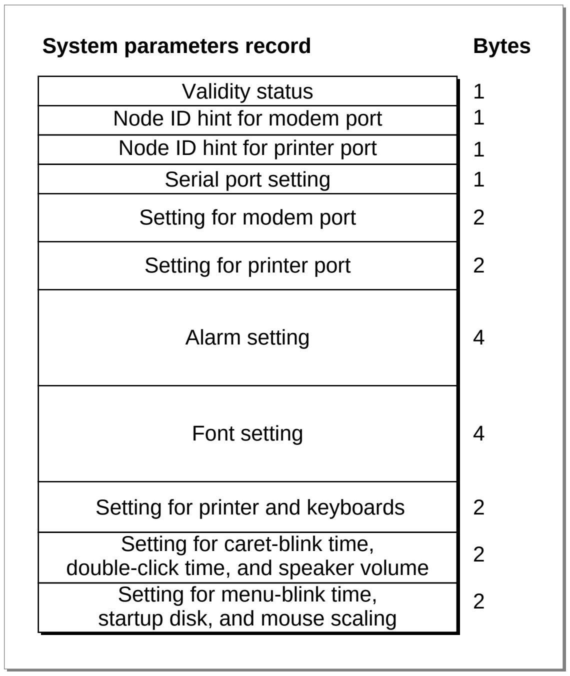
Figure 7-1 Interaction between parameter RAM and low memory



The 20 accessible bytes of parameter RAM are described by the ***system parameters record***, which is defined by a data structure of type `SysParmType`.

Figure 7-2 shows the general structure of the system parameters record, which contains 11 fields.

Figure 7-2 The format of the system parameter record


System parameters record
Bytes
Validity status
1
Node ID hint for modem port
1
Node ID hint for printer port
1
Serial port setting
1
Setting for modem port
2
Setting for printer port
2
Alarm setting
4
Font setting
4
Setting for printer and keyboards
2
Setting for caret-blink time, double-click time, and speaker volume
2
Setting for menu-blink time, startup disk, and mouse scaling
2

A system parameters record contains 11 fields. See page 7-31 for the exact structure of each field.

The first field of the system parameters record contains information about the validity status of the clock chip. Whenever a write to the clock chip is successful, the value $A8 is stored in this field. The status is examined when the clock chip is read at system startup.

The second and third fields contain information about the node ID for the modem port and printer port.

The fourth field tells which device or devices may use each of the serial ports.

The fifth field contains the baud rate, data bits, stop bits, and parity for the modem port. Bits 0–9 define the baud rate; bits 10 and 11 define the number of data bits; bits 12 and 13 define the parity; and bits 14 and 15 define the number of stop bits.

The sixth field contains the baud rate, data bits, stop bits, and parity for the printer port. As with the modem port, bits 0–9 define the baud rate; bits 10 and 11 define the number of data bits; bits 12 and 13 define the parity; and bits 14 and 15 define the number of stop bits.

The seventh field contains the time at which the alarm clock should sound. The time is defined in terms of seconds since midnight, January 1, 1904.

The eighth field contains the default application font number minus 1.

The ninth field contains the settings for the printer and for the keyboard. Bit 0 designates whether the currently chosen printer (if any) is connected to the printer port (0) or the modem port (1). Bits 1–7 are reserved for future use. Bits 8–11 of this field contain the ***auto-key rate***, the rate at which a character key repeats when it's held down; this value is stored in 2-tick units. Bits 12–15 contain the ***auto-key threshold***, the length of time a key must be held down before it begins to repeat; this value is stored in 4-tick units.

The tenth field contains miscellaneous user settings. Bits 0–3 contain the caret-blink time, and bits 4–7 contain the double-click time; both values are stored in four-tick units. The ***caret-blink time*** is the interval between blinks of a caret that marks the insertion point in text. The ***double-click time*** is the greatest interval between a mouse-up and mouse-down event that would qualify two mouse clicks as a double click. Bits 8–10 contain the speaker volume setting, which ranges from silent (0) to loud (7).

The last field contains more miscellaneous user settings. Bits 2 and 3 contain a value from 0 to 3 designating the ***menu-blink time*** , which is how many times a menu item blinks when the user chooses it. Because system software automatically calls both standard and nonstandard menu definition procedures the appropriate number of times, you should not need to worry about that value in parameter RAM. Bit 4 indicates whether the preferred system startup disk is in an internal (0) or external (1) drive. If there is any problem using the disk in the specified drive, the other drive is used. Bit 6 designates whether mouse scaling is on (1) or off (0). If mouse scaling is on, cursor movement doubles if the user moves the mouse more than a certain number of pixels between vertical retrace interrupts.

The global variable `SysParam` contains the address of the start of the system parameters record. Other global variables allow you to access individual fields of the system parameters record directly. These global variables all begin with the letters `SP` and point directly into the system parameters record stored in low memory. Other global variables referencing memory locations outside of the system parameters record are used to store copies of individual fields of the system parameters record.

▲ **WARNING**
The default values for parameter RAM vary depending on the version of the system software. Therefore, do not rely on any one default value being the same for all machines. ▲

Though default values can vary, most of the U.S. system software "shares" default values. The default values for parameter RAM, for U.S. system software, are shown in Table 7-1.

Table 7-1 Default values for parameter RAM (for U.S. system software)

Description	Default value
Validity status	$A8
Node ID hint for modem port	0
Node ID hint for printer port	0
Serial port use	0 (both ports)
Modem port configuration	9600 baud, 8 data bits, no parity, 2 stop bits
Printer port configuration	9600 baud, 8 data bits, no parity, 2 stop bits
Alarm setting	0 (midnight, January 1, 1904)
Application font minus 1	2 (indicating Geneva)
Auto-key threshold	6 (24 ticks)
Auto-key rate	3 (6 ticks)
Printer connection	0 (printer port)
Caret-blink time	8 (32 ticks)
Double-click time	8 (32 ticks)
Speaker volume	3 (medium)
Menu-blink time	3
Preferred system start-up disk	0 (internal drive)
Mouse scaling	1 (on)

In System 7, a user can clear the current settings in the parameter RAM and restore the default values by holding down the x-Option-P-R keys at system startup. When system software detects this key combination, it resets parameter RAM to the default values and then restarts the computer again. Clearing the current settings in the parameter RAM also causes system software to change other settings not stored in parameter RAM to default values. These settings include the desktop pattern and the color depth of the default monitor.

Using the Parameter RAM Utilities

The Parameter RAM Utilities provide two functions—`GetSysPPtr` and `WriteParam`—that allow you to directly manipulate parameter RAM. The `GetSysPPtr` function lets you access the low-memory copy of the parameter RAM, and the `WriteParam` function lets you write the modified low-memory copy back to parameter RAM. A third function, `InitUtil`, is used by the system software only. At system startup, this function reads the values from parameter RAM into low memory.

You may find it necessary to read the values in parameter RAM or even change them. You read from and write to parameter RAM using the `GetSysPPtr` and `WriteParam` functions.

Many of the values held in parameter RAM are also copied at system startup into other low-memory locations. Therefore, to change a value in parameter RAM, you must change all low-memory copies representing the value before you call `WriteParam` to write the values back to the clock chip. For example, the global variable `SPVolCtl` points to the location within the system parameters record that stores the speaker volume, and the global variable `SdVolume` references a copy of this information stored elsewhere in low memory. You could change one without changing the other, although ordinarily you change both simultaneously.

▲ **WARNING**
It is not recommended that you directly manipulate parameter RAM. Your application should, if at all possible, use the routines provided by the Toolbox to read the information stored in parameter RAM. ▲

The global variable `SysParam` points to the beginning of the system parameters record stored in low memory. You can access the system parameters record directly by using this global variable, or you can use the `GetSysPPtr` routine to return a pointer to the system parameters record. Thus, you can access the low-memory system parameters record like this:

```
WITH GetSysPPtr^ DO
BEGIN
   ...   {access the system parameters record directly here}
END;
```

IMPORTANT
Though system software automatically copies parameter RAM into low memory at startup, it does not automatically do the reverse. Therefore, after you make a change to the information in the low-memory system parameters record, you must use the `WriteParam` function to copy values from that record back to the clock chip to make the change permanent. ▲

At startup, system software calls the `InitUtil` function (which you should never need to call yourself) to copy the values stored in parameter RAM into low memory. (It then copies those values into other appropriate global variables.) When you make changes to the low-memory copy of parameter RAM, you must call the `WriteParam` function to record your changes in the clock chip.

Parameter RAM Utilities Reference

This section describes the data structure and routines that are specific to the Parameter RAM Utilities. The section “Data Structures” shows the Pascal data structure for the system parameters record. The section “Routines” describes the routines that are used to access and manipulate the startup information stored in parameter RAM.

Data Structures

This section describes the systems parameter record, which contains the current settings for startup information stored in parameter RAM. For information about parameter RAM default values, see Table 7-1 on page 7-29.

The System Parameters Record

The `SysParmType` data type describes a system parameters record.

```
TYPE SysParmType =
PACKED RECORD
   valid:      Byte;    {validity status}
   aTalkA:     Byte;    {node ID hint for modem port}
   aTalkB:     Byte;    {node ID hint for printer port}
   config:     Byte;    {use types for serial ports}
   portA:      Integer; {modem port configuration}
   portB:      Integer; {printer port configuration}
   alarm:      LongInt; {alarm setting}
   font:       Integer; {application font number minus 1}
   kbdPrint:   Integer; {printer connection, auto-key settings}
   volClik:    Integer; {caret blink, double click, speaker vol.}
   misc:       Integer; {menu blink, startup disk, mouse scaling }
END;
SysPPtr = ^SysParmType;
```

Field descriptions

`valid`	Contains information about the validity status of the clock chip. Whenever a write to the clock chip is successful, the value $A8 is stored in this field. The status is examined when the clock chip is read at system startup.
`aTalkA`	Contains the node ID hint for the modem port.
`aTalkB`	Contains the node ID hint for the printer port.
`config`	Indicates which device or devices may use each of the serial ports.
`portA`	Contains the baud rate, data bits, parity, and stop bits for the modem port. Bits 0–9 define the baud rate; bits 10 and 11 define the number of data bits; bits 12 and 13 define the parity; and bits 14 and 15 define the number of stop bits.
`portB`	Contains the baud rate, data bits, parity, and stop bits for the printer port. Bits 0–9 define the baud rate; bits 10 and 11 define the number of data bits; bits 12 and 13 define the parity; and bits 14 and 15 define the number of stop bits.
`alarm`	Contains the time at which the alarm clock should sound. The time is defined in terms of seconds since midnight, January 1, 1904.
`font`	Adding 1 to this field produces the font number of the default application font.

`kbdPrint`	Contains the settings for the printer and for the keyboard. Bit 0 designates whether the currently chosen printer (if any) is connected to the printer port (0) or the modem port (1). Bits 1–7 are reserved for future use. Bits 8–11 of this field contain the auto-key rate, whose value is stored in 2-tick units. Bits 12–15 contain the auto-key threshold, whose value is stored in 4-tick units.
`volClik`	Contains miscellaneous user settings, including the caret-blink time, double-click time, and the speaker volume setting.
`misc`	Contains more miscellaneous user settings. Bits 2 and 3 contain a value from 0 to 3 designating the menu-blink time. Because system software automatically calls both standard and nonstandard menu definition procedures many times, you should not need to worry about that value in parameter RAM. Bit 4 indicates whether the preferred startup disk is in an internal (0) or external (1) drive. If there is any problem with using the disk in the specified drive, the other drive is used. Bit 6 designates whether mouse scaling is on (1) or off (0).

Routines

The Parameter RAM Utilities provide two functions for use by your application and one function for use by system software. At startup, system software uses the `InitUtil` function to read parameter RAM values into low memory. You can access the values through a system parameters record of type `SysParmType` described in the previous section. To obtain a pointer to the low-memory system parameters record, call the `GetSysPPtr` function. To copy the values in the system parameters record back into the clock chip, call the `WriteParam` function.

▲ **WARNING**
The organization of parameter RAM is subject to change. Therefore, you should not manipulate parameter RAM values directly using the operating-system routines described in this chapter; instead, use the appropriate Toolbox routines. ▲

InitUtil

System software uses the `InitUtil` function at startup time to copy values from parameter RAM and date and time information into low memory. Your application should never need to use this function.

```
FUNCTION InitUtil: OSErr;
```

DESCRIPTION

The `InitUtil` function copies the contents of parameter RAM into 20 bytes of low memory and calls the Date, Time, and Measurement Utilities' `ReadDateTime` function to copy the date and time from the clock chip into a separate low-memory location.

If the validity status in parameter RAM is not $A8 when `InitUtil` is called, `InitUtil` returns a non-zero result code. In this case, the default values are read into the low-memory copy of parameter RAM; these values are then written to the clock chip.

ASSEMBLY-LANGUAGE INFORMATION

The registers on exit for the `InitUtil` function are

Registers on exit

D0	Result code

RESULT CODES

`noErr`	0	No error
`prInitErr`	–88	Validity status not $A8

SEE ALSO

For more information about the `ReadDateTime` function, see the chapter "Date, Time, and Measurement Utilities" in this book.

GetSysPPtr

You can use the `GetSysPPtr` function to obtain a pointer to the low-memory copy of parameter RAM.

```
FUNCTION GetSysPPtr: SysPPtr;
```

DESCRIPTION

The `GetSysPPtr` function returns a pointer to the low-memory copy of parameter RAM. The copied parameter RAM values are accessible through the system parameters record.

You can examine the values stored in the various fields of this record, or you can change them and call the `WriteParam` function to copy your changes back into parameter RAM.

SPECIAL CONSIDERATIONS

Because of the organization of parameter RAM is subject to change, you should not use the `GetSysPPtr` function to change the values in parameter RAM. Instead use the appropriate Toolbox routines to modify values in parameter RAM.

ASSEMBLY-LANGUAGE INFORMATION

The global variable `SysParam` contains the address of the start of the system parameters record. Other global variables allow you to access individual fields of the system parameters record directly. These global variables all begin with the letters `SP` and point directly into the system parameters record stored in low memory. Other global variables referencing memory locations outside of the system parameters record are used to store copies of individual fields of the system parameters record.

SEE ALSO

For information about the system parameters record, see page 7-31. For a list of global variables associated with the system parameters record, see "Global Variables" on page 7-38. The `WriteParam` function is described next.

WriteParam

You can use the `WriteParam` function to write the modified values in the system parameters record to parameter RAM.

```
FUNCTION WriteParam: OSErr;
```

DESCRIPTION

The `WriteParam` function writes the modified values in the system parameters record to parameter RAM. Your application should call this function only after making changes to the system parameters record (returned by the `GetSysPPtr` function described in the previous section).

The `WriteParam` function also attempts to verify the values written by reading them back in and comparing them to the values in the low-memory copy.

SPECIAL CONSIDERATIONS

Because the organization of parameter RAM is subject to change, you should not use the `WriteParam` function to change the values in parameter RAM. Instead use the appropriate Toolbox routines to modify values in parameter RAM.

Note

If you accidentally use `WriteParam` to write incorrect values into parameter RAM, the user can clear the current settings in the parameter RAM and restore the default values by holding down the x-Option-P-R keys at system startup. ◆

ASSEMBLY-LANGUAGE INFORMATION

The registers on entry and exit for the `WriteParam` functions are

Registers on entry

A0	SysParam
D0	MinusOne

Registers on exit

D0	Result code

For historical reasons, you must set up register A0 with the global variable `SysParam` and register D0 with the global variable `MinusOne`. When `WriteParam` returns, register D0 contains the result code.

RESULT CODES

`noErr`	0	No error
`prWrErr`	–87	Parameter RAM written did not verify

SEE ALSO

For a description of the system parameters record, see page 7-31.

Summary of the Parameter RAM Utilities

Pascal Summary

Data Types

```
TYPE SysParmType =
   PACKED RECORD
      valid:      Byte;    {validity status}
      aTalkA:     Byte;    {node ID hint for modem port}
      aTalkB:     Byte;    {node ID hint for printer port}
      config:     Byte;    {use types for serial ports}
      portA:      Integer; {modem port configuration}
      portB:      Integer; {printer port configuration}
      alarm:      LongInt; {alarm setting}
      font:       Integer; {application font number minus 1}
      kbdPrint:   Integer; {printer connection, auto-key settings}
      volClik:    Integer; {caret blink, double click, speaker volume}
      misc:       Integer; {menu blink, startup disk, mouse scaling}
   END;

   SysPPtr        = ^SysParmType;
```

Routines

```
FUNCTION InitUtil           : OSErr;
FUNCTION GetSysPPtr         : SysPPtr;
FUNCTION WriteParam         : OSErr;
```

C Summary

Data Types

```
struct SysParmType {
      char     valid;          /*validity status*/
      char     aTalkA;         /*node ID hint for modem port*/
      char     aTalkB;         /*node ID hint for printer port*/
      char     config;         /*use types for serial ports*/
      short    portA;          /*modem port configuration*/
      short    portB;          /*printer port configuration*/
      long     alarm;          /*alarm setting*/
      short    font;           /*application font number minus 1*/
      short    kbdPrint;       /*printer connection, auto-key settings*/
      short    volClik;        /*caret blink, double click, speaker volume*/
      short    misc;           /*menu blink, startup disk, mouse scaling*/
   };

typedef struct SysParmType SysParmType;
typedef SysParmType *SysPPtr;
```

Routines

```
pascal OSErr InitUtil      (void);
SysPPtr GetSysPPtr         (void);
pascal OSErr WriteParam    (void);
```

Assembly-Language Summary

Data Structures

SysParmType Data Structure

0	`valid`	1 byte	validity status
1	`aTalkA`	1 byte	node ID hint for modem port
2	`aTalkB`	1 byte	node ID hint for printer port
3	`config`	1 byte	use types for serial ports
4	`portA`	word	modem port configuration
6	`portB`	word	printer port configuration
8	`alarm`	long	alarm setting
12	`font`	word	application font number minus 1
14	`kbdPrint`	word	printer connection, auto-key settings
16	`volClik`	word	caret blink, double click, speaker volume
18	`misc`	word	menu blink, system startup disk, mouse scaling

Global Variables

`GetParam`	System parameter scratch
`SPAlarm`	The alarm setting
`SPATalkA`	The node ID hint for modem port
`SPATalkB`	The node ID hint for printer port
`SPClikCaret`	The double-click and caret-blink times
`SPConfig`	The use types for serial ports
`SPFont`	The application font number minus 1
`SPKbd`	The auto-key threshold and rate
`SPMisc1`	Miscellaneous
`SPMisc2`	The setting for mouse scaling, the system startup disk, and menu-blink time
`SPPortA`	The modem port configuration
`SPPortB`	The printer port configuration
`SPPrint`	The printer connection
`SPValid`	The validity status of parameter RAM
`SPVolCtl`	The speaker volume
`SysParam`	The low-memory copy of parameter RAM

Result Codes

`noErr`	0	No error
`prWrErr`	–87	Parameter RAM written did not verify
`prInitErr`	–88	Validity status is not $A8