# OS-9 for 68K OEM System-State Debuggin Addendum

## Version 1.1

www.radisys.com

World Headquarters
5445 NE Dawson Creek Drive • Hillsboro, OR
97124 USA
Phone: 503-615-1100 • Fax: 503-615-1121
Toll-Free: 800-950-0044

International Headquarters
Gebouw Flevopoort • Televisieweg 1A
NL-1322 AC • Almere, The Netherlands
Phone: 31 36 5365595 • Fax: 31 36 5365620

RadiSys Microware Communications Software Division, Inc.
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000

Revision C
August 1999

## Copyright and publication information

This manual reflects version 3.0 of OS-9 for 68K. Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Microware Communications Software Division, Inc.

## Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

## Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

August 1999

# Table of Contents

# Chapter 1: Creating Low-Level Serial Device Drivers and Timer Modules

This chapter includes the following topics:

- **Overview**
- **The Console Device Record**
- **Low-Level Serial I/O Module Services**
- **Initializing the Low-Level Serial Device Drivers**
- **Building the Low-Level Serial Device Drivers**
- **Low-Level Timer Module**

# Overview

The distribution package contains source code for several low-level serial modules you can configure and use in your system without modification. If your target has a serial device for which no I/O module already exists, use the example sources as a guide to writing your own. If both the console port and communications port use the same type of hardware interface, you only need to build one low-level I/O module.

The distributed low-level serial I/O module sources are in `MWOS/SRC/ROM/SERIAL`. Create a subdirectory for your own source code if you are building you own I/O module.

In addition to the directories listed earlier, each example port directory contains `<target>/ROM/IO<device>` directories containing makefiles used to build the low-level I/O module used in the port. You need to create such a directory and makefile for your serial devices in your ports directory. Use the example makefiles as a guide.

# The Console Device Record

A console device (consdev) structure is maintained for each low level serial I/O device included with the low-level system modules. This structure is used to access the services of the I/O module, and to maintain lists of such devices. The definition of consdev appears in the header file, MWOS/SRC/DEFS/ROM/rom.h, and appears here for illustration.

```
struct consdev {
   idver infoid;                                   /* structure version tag */
   void    *cons_addr;                             /* port address of I/O device*/
   u_int32 (*cons_probe)(Rominfo, Consdev),        /* h/w probe service */
           (*cons_init)(Rominfo, Consdev),         /* initialization service */
           (*cons_term)(Rominfo, Consdev);         /* de-initialization service*/
   u_char  (*cons_read)(Rominfo, Consdev);         /* read service */
   u_int32 (*cons_write)(char, Rominfo, Consdev),  /* write service */
           (*cons_check)(Rominfo, Consdev);        /* character check service */
   u_int32 (*cons_stat)(Rominfo, Consdev, u_int32),
           (*cons_irq)(Rominfo, Consdev),
           (*proto_upcall)(Rominfo, void*, char*);
   u_int32 cons_flags;                             /* device flags */
   u_char  cons_csave,                             /* read ahead stash */
            cons_baudrate,                         /* communication baud rate */
            cons_parsize,                          /* parity, data bits, stop bits */
            cons_flow;                             /* flow control */
   u_int32  cons_vector,                           /* interrupt vector */
            cons_priority,                         /* interrupt priority */
            poll_timeout;
   u_char   *cons_abname,                          /* abreviated name */
            *cons_name;                            /* full name and description */
   void     *cons_data;                            /* device specific data */
   void     *upcall_data;
   Consdev  cons_next;                             /* next serial device in list*/
   u_int32  cons_level;                            /* interrupt level */
   int      reserved;
};
```

The p2start entry point of the low-level I/O module must initialize this structure and link it onto a list of available devices. The conscnfg and commcnfg modules use the configured console and communication port names, respectively, to locate the proper console device records and initialize the console and communications port pointers.

# Low-Level Serial I/O Module Services

The following entry points describe the services required of each low-level serial I/O module.

**Table 1-1 Low-Level Serial I/O Module Entry Points**

| Function | Description |
|---|---|
| `cons_check()` | Check I/O Port |
| `cons_init()` | Initialize Port |
| `cons_irq()` | Polled Interrupt Service Routine for I/O Device |
| `cons_probe()` | Probe for Port |
| `cons_read()` | Read Character from I/O Port |
| `cons_stat()` | Set Status on Console I/O Device |
| `cons_term()` | De-initialize Port |
| `cons_write()` | Write Character to Output Port |

# cons_check()

Check I/O Port

## Syntax

```
u_int32 cons_check(
   Rominfo   romstr,
   Consdev   cdev);
```

## Description

`cons_check()` interrogates the port to determine if an input character is present and returns the appropriate status.

## Parameters

| | |
|---|---|
| `romstr` | Points to the `rominfo` structure |
| `cdev` | Points to the console device record for the device |

## Syntax

```
u_int32 cons_init(
   Rominfo   romstr,
   Consdev   cdev);
```

## Description

`cons_init()` initializes the port. It resets the device port, sets up for transmit and receive, and sets up baud rate, parity, bits per type, and number of stop bits.

## Parameters

| | |
|---|---|
| `romstr` | Points to the `rominfo` structure |
| `cdev` | Points to the console device record for the device |

# cons_irq()

Polled Interrupt Service Routine for I/O Device

## Syntax

```
u_int32 cons_irq(
    Rominfo   rinf,
    Consdev   cdev);
```

## Description

`cons_irq()` is an interrupt service routine installed for the device performing the following polling interrupt service on receipt of a device interrupt:

1. Disables further interrupts on the device.
2. Clears the interrupt from the device and initializes the low-level polling timer.
3. Sets the polling time-out value and loops checking the device and timer until either a character is received or the time-out occurs.
4. Sends a received character up the protocol stack by calling the uplink routine installed in the `rominfo` structure.
5. Repeats the first four steps until a timeout occurs.
6. Re-enables device interrupts and returns.

## Parameters

| | |
|---|---|
| `rinf` | Points to the `rominfo` structure |
| `cdev` | Points to the console device record for the device |

# cons_probe()

## Probe for Port

### Syntax

```
u_int32 cons_probe(
   Rominfo   romstr,
   Consdev   cdev);
```

### Description

`cons_probe()` should test to see if the hardware described by the console device record `cdev` is actually present. Generally, this could be a read of an I/O register based at the value of `cons_addr` in the console device record.

### Parameters

| | |
|---|---|
| `romstr` | Points to the `rominfo` structure |
| `cdev` | Points to the console device record for the device |

# cons_read()

## Read Character from I/O Port

### Syntax

```
u_char cons_read(
   Rominfo    romstr,
   Consdev    cdev);
```

### Description

cons_read() returns a character from the device's input port. cons_read() repeatedly calls cons_check() until a character is present. cons_read() should not echo the character nor perform any special character handling (for example, XON-XOFF).

### Parameters

| | |
|---|---|
| romstr | Points to the rominfo structure |
| cdev | Points to the console device record for the device |

# cons_stat()

Set Status on Console I/O Device

## Syntax

```
u_int32 cons_stat(
   Rominfo   rinf,
   Consdev   cdev,
   u_int32   code);
```

## Description

`cons_stat()` changes the operational mode of the I/O module.

## Parameters

| | |
|---|---|
| `rinf` | Points to the `rominfo` structure |
| `cdev` | Points to the console device record for the device |
| `code` | Is the low-level `setstat` code indicating operational mode change |

## Supported Setstat Codes

The supported `setstat` codes are defined in `MWOS/SRC/DEFS/ROM/rom.h`. A description follows:

`CONS_SETSTAT_POLINT_OFF/CONS_SETSTAT_ROMBUG_ON`

Show interrupts are disabled for the device, changing the operational mode to strict polling mode.

`CONS_SETSTAT_ROMBUG_OFF`

Shows interrupts are enabled for the device changing the operational mode to interrupt driven mode.

`CONS_SETSTAT_POLINT_ON`

Shows interrupts are enabled for device only if a low-level timer is available, changing the operational mode to polled interrupt.

# cons_term()

## De-initialize Port

### Syntax

```
u_int32 cons_term(
   Rominfo   romstr,
   Consdev   cdev);
```

### Description

`cons_term()` should shut the port down by disabling transmit and receive.

### Parameters

| | |
|---|---|
| `romstr` | Points to the `rominfo` structure |
| `cdev` | Points to the console device record for the device |

# cons_write()

Write Character to Output Port

## Syntax

```
u_int32 cons_write (
   char        c,
   Rominfo     romstr,
   Consdev     cdev);
```

## Description

`cons_write()` writes a character to the output port with no special character processing.

The previous entry points are sufficient to support resident debugging using RomBug. For the driver to support remote debugging over SLIP, the following entry points must also be defined.

## Parameters

| | |
|---|---|
| `c` | Is the character to be written |
| `romstr` | Points to the `rominfo` structure |
| `cdev` | Points to the console device record for the device |

# Initializing the Low-Level Serial Device Drivers

The initialization entry point for the low-level system modules is supplied in a relocatable (`.r`) file in the distribution. This entry point branches to the C function `p2start()` you need to provide for each of your low level I/O modules. The initialization routine should perform these tasks:

- Allocate/initialize the console device structure for the device.
- Make the entry points for its services available through the `consdev` structure.
- Initialize configuration data for the I/O device.
- Install its `consdev` structure on the list of I/O devices in the console record.

An example `p2start()` routine for a low level I/O module follows. (The console device structure is allocated in the modules static data area.)

```
consdev    cons_r;          /* allocate console device structure */

error_code p2start(
Rominfo rinf,               /* bootstrap services record structure pointer */
u_char *glbls)              /* bootstrap global data pointer */
{
  Cons_svcs console = rinf->cons;
                            /* get the console services record pointer*/
  Consdev   cdev;           /* local console device structure pointer */


          /* verify a console services module has been initialized */

  if (console == NULL)
    return (EOS_NOTRDY);    /*cannot install w/o the console services record*/

          /* initialize device structure for our device */

  cdev = &cons_r;           /* point to our console device structure */
  cdev->struct_id = CONSDEVID;          /* id and version tags */
  cdev->struct_ver = CDV_VER_MAX;
          /* export our service routine entry points */
  cdev->cons_probe = &io16450_probe;
  cdev->cons_init = &io16450_init;
  cdev->cons_term = &io16450_term;
  cdev->cons_read = &io16450_read;
  cdev->cons_write = &io16450_write;
  cdev->cons_check = &io16450_check;
```

```
          /*  The following services are not required for the initial port */
  /*
  cdev->cons_stat = &io16450_stat;
  cdev->cons_irq = &io16450_irq;
  */

          /* initialize the device configuration data */

  cdev->cons_addr = (void *)COMM2ADDR;       /* base address of I/O port */
  cdev->cons_baudrate = CONS_BAUDRATE_9600;  /* communication baud rate  */
  cdev->cons_vector = COMMVECTOR;            /* interrupt vector */
  cdev->cons_priority = COMMPRIORITY;        /* interrupt priority */
  cdev->poll_timeout = 2000;                 /* polling routine timout value */
  cdev->cons_abname = (u_char *)COMM2ABNAME; /* abreviated device name */
  cdev->cons_name = (u_char *)COMM2NAME;     /* device name */

        /* install the device structure on the list of available I/O modules */

  cdev->cons_next = console->rom_conslist;
  console->rom_conslist = cdev;

  return (SUCCESS);
}
```

The definitions used to initialize the device configuration data should be placed in the port-specific `systype.h` header file, leaving the I/O module source code portable across platforms.

If the same I/O module is to be used with both the console and communications ports, then an additional console device structure, say, `comm_r` should be allocated and initialized with the proper data for the communications port. Both console device records should then be added to the list of available devices.

## Note

The console and communications port configuration modules (`conscnfg` and `commcnfg`), using the configuration data module (`conscnfg`), determine which console device record is selected as console and communications port.

## Building the Low-Level Serial Device Drivers

The makefile for you I/O module should be created in a properly named subdirectory of your ports ROM directory (for example, `<target>/ROM/<device>`). Use the makefiles from the example ports as a guide.

To add your low level serial I/O module to the system, edit the makefile, `<target>/ROM/makefile`, and add your device directory name to the list of targets used to define the `TRGTS` macro. Add your directory names before the name `BOOTROM`, making sure `BOOTROM` is the last directory name used in the `TRGTS` macro definition.

By doing this, you ensure your low level I/O module is rebuilt along with the bootstrap code and the rest of the low-level system modules when the boot image is made.

# Low-Level Timer Module

You need to provide a low-level timer module to support the low-level driver modules for remote debugging. The distribution contains sources for example timers in the `MWOS/SRC/ROM/TIMERS` directory.

The following entry points are required in the low-level timer module.

Table 1-2 Low-Level Timer Module Entry Points

| Function | Description |
|---|---|
| `timer_deinit()` | Remove Timer Initialization |
| `timer_get()` | Get Time Remaining |
| `timer_init()` | Initialize Timer |
| `timer_set()` | Set Timer Flag |

# timer_deinit()

## Remove Timer Initialization

### Syntax

```
void timer_deinit(Rominfo rinf);
```

### Description

`timer_deinit()` clears the timer data structures and hardware to free the timer modules.

### Parameters

| | |
|---|---|
| `rinf` | Points to the `rominfo` structure |

# timer_get()

## Get Time Remaining

### Syntax

```
u_int32 timer_get(Rominfo rinf);
```

### Description

`timer_get()` returns the amount of time remaining until the time-out occurs. If the time-out value has been reached, `timer_get()` returns `0`.

### Parameters

| | |
|---|---|
| `rinf` | Points to the `rominfo` structure |

# timer_init()

Initialize Timer

## Syntax

```
error code timer_init(Rominfo rinf);
```

## Description

`timer_init()` initializes data structures and hardware targeted by timer modules.

## Parameters

| | |
|---|---|
| `rinf` | Points to the `rominfo` structure |

# timer_set()

## Set Timer Flag

### Syntax

```
void timer_set(
   Rominfo    rinf,
   u_init32   timeout);
```

### Description

`timer_set()` uses the specified time-out value to initialize a `time-out` flag checked by subsequent calls to `timer_get()`.

### Parameters

`rinf`
Points to the `rominfo` structure

`timeout`
Is the counter indicating the amount of time to wait

# Chapter 2: p2lib Functions

Three libraries are shipped as part of this distribution:

- `p2privat.l`
- `romsys.l`
- `p2lib.l`

The `p2privte.l` and `romsys.l` libraries are only used by the bootstrap code (`romcore`). The `p2lib.l` library contains functions you can use to customize your own low-level system modules.

RadiSys.

MICROWARE SOFTWARE

# Functions

The `p2lib.l` functions and descriptions are shown in **Table 2-1**.

**Table 2-1 p2lib.l Functions**

| Function | Description |
|---|---|
| `getrinf()` | Get the Rominfo Structure Pointer |
| `hwprobe()` | Check a System Hardware Address |
| `inttoascii()` | Convert an Integer to ASCII |
| `outhex()` | Display One Hexidecimal Digit |
| `out1hex()` | Display a Hexidecimal Byte |
| `out2hex()` | Display a Hexidecimal Word |
| `out4hex()` | Display a Hexidecimal Longword |
| `rom_udiv()` | Unsigned Integer Division |
| `setexcpt()` | Install Exception Handler |
| `swap_globals()` | Exchange Current Globals Pointer |

# getrinf()

## Get the Rominfo Structure Pointer

### Syntax

```
error_code getrinf(Rominfo *rinf_p)
```

### Description

`getrinf()` finds and returns the pointer to the `rominfo` structure from the system globals.

### Parameters

| | |
|---|---|
| `rinf_p` | Is the address where `getrinf()` stores the pointer to the `rominfo` structure |

# hwprobe()

## Check a System Hardware Address

### Syntax

```
error_code hwprobe(
   void      *addr,
   u_int32   ptype,
   Rominfo   rinf);
```

### Description

`hwprobe()` sets up the appropriate handlers to catch bus trap errors, then probes the system memory at the specified address, attempting to read either a byte, word, or long. In the event of a bus fault, an error is returned. `SUCCESS` is returned if the read is successful.

### Parameters

`*addr` Is the specific memory address you want probed

`ptype` Is the probe type, either byte, word, or long

`rinf` Points to the `rominfo` structure

# inttoascii()

Convert an Integer to ASCII

## Syntax

```
char *inttoascii(
   u_int32    value,
   char       *bufptr);
```

## Description

inttoascii() converts its input value to its base 10 ASCII representation stored in bufptr. The caller must ensure bufptr points to a sufficient storage space for the ASCII representation. inttoascii() returns bufptr.

## Parameters

| | |
|---|---|
| value | Is the integer value converted |
| bufptr | Points to the location where the ASCII value is stored |

# outhex()

## Display One Hexidecimal Digit

### Syntax

```
void outhex(
   u_char     n,
   Rominfo    rinf);
```

### Description

`outhex()` displays one hexidecimal digit on the system console. The lower 4 bits of the character `n` are displayed using the `putchar()` service of the system console device.

### Parameters

| | |
|---|---|
| `n` | Is the character for which the hex value is to be displayed |
| `rinf` | Points to the `rominfo` structure |

# out1hex()

Display a Hexidecimal Byte

## Syntax

```
void out1hex(
   u_char    byte,
   Rominfo   rinf);
```

## Description

`out1hex()` displays the hexidecimal representation of a byte on the system console device.

## Parameters

| | |
|---|---|
| `byte` | Is the byte for which the hex value is to be displayed |
| `rinf` | Points to the `rominfo` structure |

# out2hex()

## Display a Hexidecimal Word

### Syntax

```
void out2hex(
   u_short   word,
   Rominfo   rinf);
```

### Description

`out2hex()` displays the hexidecimal representation of a word on the system console device.

### Parameters

| | |
|---|---|
| `word` | Is the word for which the hex value is to be displayed |
| `rinf` | Points to the `rominfo` structure |

# out4hex()

Display a Hexidecimal Longword

## Syntax

```
void out4hex(
   u_long    longword,
   Rominfo   rinf);
```

## Description

`out4hex()` displays the hexidecimal representation of a longword on the system console device.

## Parameters

| | |
|---|---|
| `longword` | Is the longword for which the hex value is to be displayed |
| `rinf` | Points to the `rominfo` structure |

# rom_udiv()

Unsigned Integer Division

## Syntax

```
unsigned rom_udiv(
   unsigned  dividend,
   unsigned  divisor);
```

## Description

`rom_udiv()` provides an integer division routine that does not rely on the presence of a built-in hardware division instruction.

## Parameters

`dividend` Is the number to be divided

`divisor` Is the number by which the dividend is to be divided

# setexcpt()

Install Exception Handler

## Syntax

```
u_int32 setexcpt(
   u_int32   vector,
   u_int32   irqsvc,
   Rominfo   rinf);
```

## Description

`setexcpt()` installs an exception handler on the system exception vector table for the specified exception. This is usually used with the `setjump()` and `longjump()` C functions to provide a bus fault recovery mechanism prior to polling hardware.

## Parameters

| | |
|---|---|
| `vector` | Is the number of the exception for which the handler should be installed |
| `irqsvc` | Points to the exception handling code you want installed |
| `rinf` | Points to the `rominfo` structure |

# swap_globals()

Exchange Current Globals Pointer

## Syntax

```
u_char *swap_globals(u_char *new_globals);
```

## Description

`swap_globals()` replaces the caller's global data pointer with a new value and returns the old value.

## Parameters

| | |
|---|---|
| `new_globals` | Is the value to be assigned to the global data pointer |

# Chapter 3: Console I/O Services

The console module provides a high level I/O interface to the entry points of the low-level serial device driver configured as the system console. These services are made available through the console services field of the `rominfo` structure. Assuming the variable `rinf` points to the `rominfo` structure, `rinf->cons` can be used to reference the console services record.

# Functions

The header file MWOS/SRC/DEFS/ROM/rom.h contains the structure definitions for the rominfo structure and the console services record, cons_svcs.

Table 3-1 lists the services are available through the console services record.

**Table 3-1 Console I/O Services**

| Function | Description |
|---|---|
| rom_getc() | Read the First Character |
| rom_getchar() | Read First Character Not XON or XOFF |
| rom_gets() | Read a Null-terminated String |
| rom_putc() | Output One Character |
| rom_putchar() | Output a Character and a Line Feed for Carriage Returns |
| rom_puterr() | Convert Error Code to a Null-terminated String |
| rom_puts() | Write a Null-terminated String |

# rom_getc()

## Read the First Character

### Syntax

```
char rom_getc(
   Rominfo   rinf,
   Consdev   cdev);
```

### Description

`rom_getc()` calls the low-level read routine of the specified console device record to read a single input character from the associated serial device.

`rom_getc()` returns the character read.

### Parameters

| | |
|---|---|
| `rinf` | Points to the `rominfo` structure |
| `cdev` | Points to the console device record for the serial device to be used |

### Example

```
char ch;
ch = rinf->cons->rom_getc(rinf, cdev);
```

# rom_getchar()

Read First Character Not XON or XOFF

## Syntax

```
char rom_getchar(Rominfo rinf);
```

## Description

`rom_getchar()` calls the low-level read routine of the console device record configured for use as the system console. `rom_getchar()` reads characters from the console until the first character other than XON or XOFF is read.

If echoing is enable for the console, `rom_getchar()` calls `putchar()` to echo this character. The character is then returned by `rom_getchar()`.

## Parameters

| | |
|---|---|
| `rinf` | Points to the `rominfo` structure |

## Example

```
ch = rinf->cons->rom_getchar(rinf);
```

## rom_gets()

### Read a Null-terminated String

### Syntax

```
char *rom_gets(
   char       *buff,
   u_int32    count,
   Rominfo    rinf);
```

### Description

rom_gets() calls the low-level read routine of the console device record configured for use as the system console. rom_gets() reads a null-terminated string from the console into the buffer designated by the pointer buff. The rudimentary line editing feature of <backspace> is supported by rom_gets().

rom_gets() returns to the caller when it receives a carriage return character (0x0d), or when count many characters have been read. A pointer to the beginning of the buffer is passed back to the caller.

### Parameters

| | |
|---|---|
| buff | Points to the input buffer into which the string is read |
| count | Is the integer used as the size of the input buffer including the null termination |
| rinf | Points to the rominfo structure |

### Example

```
str = rinf->cons->rom_gets(buffer, count, rinf);
```

# rom_putc()

Output One Character

## Syntax

```
void rom_putc(
   char       c,
   Rominfo    rinf,
   Consdev    cdev);
```

## Description

`rom_putc()` calls the low-level write routine of the specified console device record to output a single character to the associated serial device.

## Parameters

| | |
|---|---|
| `c` | Is the character to output |
| `rinf` | Points to the `rominfo` structure |
| `cdev` | Points to the console device record for the serial device to be used |

## Example

```
rinf->cons->rom_putc(ch, rinf, cdev);
```

## rom_putchar()

### Output a Character and a Line Feed for Carriage Returns

### Syntax

```
void rom_putchar(
   char        c,
   Rominfo     rinf);
```

### Description

`rom_putchar()` calls the low-level write routine of the console device record configured for use as the system console. `rom_putchar()` writes the specified character to the console. If the character is a carriage return character (`0x0d`) `rom_putchar()` also writes a line feed character (`0x0a`) to the console.

### Parameters

| | |
|---|---|
| `c` | Is the character to output |
| `rinf` | Points to the `rominfo` structure |

### Example

```
rinf->cons->rom_putchar(ch, rinf);
```

# rom_puterr()

Convert Error Code to a Null-terminated String

## Syntax

```
void rom_puterr(
   error_code   stat,
   Rominfo    rinf);
```

## Description

`rom_puterr()` converts the specified error code to a null terminated ascii string representation of the form `XXX:YYY` and outputs this string to the system console using the `rom_putc()` service.

## Parameters

| | |
|---|---|
| `stat` | Is the value of the error code to be displayed |
| `rinf` | Points to the rominfo structure |

## Example

```
rinf->cons->rom_getchar(status, rinf);
```

# rom_puts()

## Write a Null-terminated String

### Syntax

```
void rom_puts(
   char       *buff,
   Rominfo    rinf);
```

### Description

`rom_puts()` calls the low-level write routine of the console device record configured for use as the system console. `rom_puts()` writes a null terminated string to the console device.

### Parameters

| | |
|---|---|
| `buff` | Points to the first character of the string to output |
| `rinf` | Points to the `rominfo` structure |

### Example

```
rinf->cons->rom_puts(buffer, rinf);
```

# Index

# Product Discrepancy Report

To: Microware Customer Support

FAX: 515-224-1352

From:________________________________________

Company:_____________________________________

Phone:_______________________________________

Fax:____________________Email:_________________

Product Name:

Description of Problem:

_____________________________________________

_____________________________________________

_____________________________________________

_____________________________________________

_____________________________________________

_____________________________________________

_____________________________________________

_____________________________________________

_____________________________________________

_____________________________________________

Host Platform___________________________________

Target Platform__________________________________

RadiSys.

MICROWARE SOFTWARE