15

I/O Manager

15.1 Introduction

The **input/output interface manager** provides a well-defined mechanism for accessing device drivers and a structured methodology for organizing device drivers. The directives provided by the I/O manager are:

Name	Directive Description
io_initialize	Initialize a device driver
io_open	Open a device
io_close	Close a device
io_read	Read from a device
io_write	Write to a device
io_control	Special device services

15.2 Background

15.2.1 Device Driver Table

Each application utilizing the **RTEMS** I/O manager must specify the address of a **Device Driver Table** in its **Configuration Table**. This table contains each device driver's entry points. Each device driver may contain the following entry points:

- *Initialization*
- *Open*
- *Close*
- *Read*
- *Write*
- *Control*

If the device driver does not support a particular entry point, then that entry in the **Configuration Table** should be **NO_DRIVER_ENTRY**. **RTEMS** will return **SUCCESSFUL** as the executive's and zero (0) as the device driver's return code for these device driver entry points.

15.2.2 Major and Minor Device Numbers

Each call to the I/O manager must provide a device's major and minor numbers as arguments. The major number is the index of the requested driver's entry points in the **Device Driver Table**, and is used to select a specific device driver. The exact usage of the minor number is driver specific, but is commonly used to distinguish between a number of devices controlled by the same driver.

15.2.3 Device Driver Environment

Application developers, as well as device driver developers, must be aware of the following regarding the **RTEMS** I/O Manager:

- *A device driver routine executes in the context of the invoking task. Thus if the driver blocks, the invoking task blocks.*
- *The device driver is free to change the modes of the invoking task, although the driver should restore them to their original values.*
- *Device drivers may be invoked from ISRs.*
- *Only local device drivers are accessible through the I/O manager.*
- *A device driver routine may invoke all other **RTEMS** directives, including I/O directives, on both local and global objects.*

Although the **RTEMS** I/O manager provides a framework for device drivers, it makes no assumptions regarding the construction or operation of a device driver.

15.2.4 Device Driver Interface

When an application invokes an I/O manager directive, **RTEMS** determines which device driver entry point must be invoked. The information passed by the application to **RTEMS** is then passed to the correct device driver entry point.

RTEMS will invoke each device driver entry point assuming it is compatible with the following specification:

```
procedure io_entry(
  major          : in     rtems.device_major_number;
  minor          : in     rtems.device_minor_number;
  argument_block : in     rtems.address;
  executing_task : in     rtems.id;
  status         :    out rtems.unsigned32
);
```

The format and contents of the parameter block are device driver and entry point dependent.

It is recommended that a device driver avoid generating error codes which conflict with those used by application components. A common technique used to generate driver specific error codes is to make the most significant part of the **status** indicate a driver specific code.

15.2.5 Device Driver Initialization

RTEMS automatically initializes all device drivers when multitasking is initiated via the **initialize_executive** directive. **RTEMS** initializes the device drivers by invoking each device driver initialization entry point with the following parameters:

major	the major device number for this device driver.
minor	zero.
argument_block	will point to the **Configuration Table**.
executing_task	will contain zero.

The returned **status** will be ignored by **RTEMS**. If the driver cannot successfully initialize the device, then it should invoke the **fatal_error_occurred** directive.

15.3 Operations

The I/O manager provides directives which enable the application program to utilize device drivers in a standard manner. There is a direct correlation between the **RTEMS** I/O manager directives and the underlying device driver entry points.

15.4 Directives

This section details the I/O manager's directives. A subsection is dedicated to each of this manager's directives and describes the calling sequence, related constants, usage, and status codes.

15.4.1 IO_INITIALIZE - Initialize a device driver

CALLING SEQUENCE:

```
procedure io_initialize (
   major        : in     rtems.device_major_number;
   minor        : in     rtems.device_minor_number;
   argument     : in     rtems.address;
   return_value :    out rtems.unsigned32;
   result       :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	successfully initialized
INVALID_NUMBER	invalid major device number

DESCRIPTION:

This directive calls the device driver initialization routine specified in the **Device Driver Table** for this **major** number. This directive is automatically invoked for each device driver when multitasking is initiated via the **initialize_executive** directive.

A device driver initialization module is responsible for initializing all hardware and data structures associated with a device. If necessary, it can allocate memory to be used during other operations.

NOTES:

This directive may or may not cause the calling task to be preempted. This is dependent on the device driver being initialized.

15.4.2 IO_OPEN - Open a device

CALLING SEQUENCE:

```
procedure io_open (
   major        : in     rtems.device_major_number;
   minor        : in     rtems.device_minor_number;
   argument     : in     rtems.address;
   return_value :    out rtems.unsigned32;
   result       :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	successfully initialized
INVALID_NUMBER	invalid major device number

DESCRIPTION:

This directive calls the device driver open routine specified in the **Device Driver Table** for this **major** number. The open entry point is commonly used by device drivers to provide exclusive access to a device.

NOTES:

This directive may or may not cause the calling task to be preempted. This is dependent on the device driver being invoked.

15.4.3 IO_CLOSE - Close a device

CALLING SEQUENCE:

```
procedure io_close (
   major        : in     rtems.device_major_number;
   minor        : in     rtems.device_minor_number;
   argument     : in     rtems.address;
   return_value :    out rtems.unsigned32;
   result       :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	successfully initialized
INVALID_NUMBER	invalid major device number

DESCRIPTION:

This directive calls the device driver close routine specified in the **Device Driver Table** for this **major** number. The close entry point is commonly used by device drivers to relinquish exclusive access to a device.

NOTES:

This directive may or may not cause the calling task to be preempted. This is dependent on the device driver being invoked.

15.4.4 IO_READ - Read from a device

CALLING SEQUENCE:

```
procedure io_read (
   major        : in     rtems.device_major_number;
   minor        : in     rtems.device_minor_number;
   argument     : in     rtems.address;
   return_value :    out rtems.unsigned32;
   result       :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	successfully initialized
INVALID_NUMBER	invalid major device number

DESCRIPTION:

This directive calls the device driver read routine specified in the **Device Driver Table** for this **major** number. Read operations typically require a buffer address as part of the argument parameter block. The contents of this buffer will be replaced with data from the device.

NOTES:

This directive may or may not cause the calling task to be preempted. This is dependent on the device driver being invoked.

15.4.5 IO_WRITE - Write to a device

CALLING SEQUENCE:

```
procedure io_write (
   major        : in     rtems.device_major_number;
   minor        : in     rtems.device_minor_number;
   argument     : in     rtems.address;
   return_value :    out rtems.unsigned32;
   result       :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	successfully initialized
INVALID_NUMBER	invalid major device number

DESCRIPTION:

This directive calls the device driver write routine specified in the **Device Driver Table** for this **major** number. Write operations typically require a buffer address as part of the argument parameter block. The contents of this buffer will be sent to the device.

NOTES:

This directive may or may not cause the calling task to be preempted. This is dependent on the device driver being invoked.

15.4.6 IO_CONTROL - Special device services

CALLING SEQUENCE:

```
procedure io_control (
   major        : in     rtems.device_major_number;
   minor        : in     rtems.device_minor_number;
   argument     : in     rtems.address;
   return_value :    out rtems.unsigned32;
   result       :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	successfully initialized
INVALID_NUMBER	invalid major device number

DESCRIPTION:

This directive calls the device driver I/O control routine specified in the **Device Driver Table** for this **major** number. The exact functionality of the driver entry called by this directive is driver dependent. It should not be assumed that the control entries of two device drivers are compatible. For example, an RS-232 driver I/O control operation may change the baud rate of a serial line, while an I/O control operation for a floppy disk driver may cause a seek operation.

NOTES:

This directive may or may not cause the calling task to be preempted. This is dependent on the device driver being invoked.