IRIS® ATM API Programmer's Guide

Document Number 007-2334-002

CONTRIBUTORS

Written by Irene Kuffel, Carlin Otto, and Thomas Skibo
Illustrated by Carlin Otto
Production by Gloria Ackley
Engineering contributions by Irene Kuffel and Thomas Skibo

IRIS® ATM API Programmer's Guide
Document Number 007-2334-002

Contents

Figures

Tables

About This Guide

This guide explains the design philosophy and usage for the application programming interface to IRIS® ATM. The document assumes familiarity with the UNIX® networking environment and basic programming in the C language.

Acronyms Used in This Guide

The following acronyms are used throughout this guide:

AAL	ATM Adaptation Layer
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
BLLI	Broadband Low Layer Information
CSPDU	AAL Convergence Sublayer Protocol Data Unit
ILMI	Interim Local Management Interface
PVC	Permanent Virtual Channel
QoS	Quality of Service
SVC	Switched Virtual Channel
VC	Virtual Channel
VCC	Virtual Channel Connection

Style Conventions

This guide uses the following stylistic conventions:

`screen display`
Indicates system output, such as responses to commands that you see on the screen. Code samples, screen displays, and file contents also appear in this font.

`user input`
Indicates exact text that you must enter at a command line, such as commands, options, and arguments to commands.

variable
Indicates generic, place-holding variable names. Can indicate a user input variable, where you must replace the variable with text that you select.

`<xx>`
Indicates keys on the keyboard that you press; for example, press **`<Enter>`** means press only the key labeled **Enter**.

physical label
Indicates a label for a piece of hardware (for example, a pin, a wire, a port). Can also indicate the signal on a wire or pin.

command
Designates command and utility names.

filename
Indicates filenames and filename suffixes.

[ ]
Encloses optional command arguments.

...
Denotes omitted material or indicates that the preceding optional items may appear more than once in succession.

Product Support

Silicon Graphics®, Inc., provides a comprehensive product support and maintenance program for its products. If you are in the United States of America or Canada and would like support for your Silicon Graphics-supported products, contact the Technical Assistance Center at 1-800-800-4SGI. If you are outside these areas, contact the Silicon Graphics subsidiary or authorized distributor in your country.

Chapter 1

API Specification

This document describes the Silicon Graphics® application programming interface (API) for IRIS ATM boards. This first chapter provides a general overview of the API and its use. Subsequent chapters contain detailed descriptions of each API command. The product includes a C-language coding example for an application that uses the switched virtual channel API: */usr/lib/atm/examples/sigtest.c*.

Each chapter contains the commands relevant for one of the following types of implementations:

- permanent virtual channels, Chapter 2, “IRIS ATM ioctl() Commands for Permanent VCs”
- switched virtual channels, Chapter 3, “IRIS ATM ioctl() Commands for Switched VCs”
- providing information to non-IRIS interim local management interface (ILMI) modules, Chapter 4, “IRIS ATM ioctl() Commands for Use by ILMI Modules”
- configuring and controlling the IRIS ATM hardware, Chapter 5, “IRIS ATM ioctl() Commands for Communicating With the Hardware”

Features

IRIS ATM supports the following basic features upon which the IRIS ATM API is based:

- ATM adaptation layer 5 (AAL5) protocol mapping.
- ATM Signalling (ATM Forum UNI 3.0/3.1).
- Network and address management via ILMI and its ATM management information database (MIB) for multiple ATM user-network interfaces (UNIs).
- RFC 1577 compliant ("classical IP") as well as non-compliant configurations. Ability to function as address resolution (ATM ARP) server or client for each IP subnetwork.

The IRIS ATM API supports the following ATM services:

- Permanent Virtual Channels (PVC) for point-to-point, bi-directional or uni-directional connections with constant bit rate (CBR), variable bit rate (VBR), or best-effort service. The traffic can be IP (with or without LLC/SNAP encapsulation) or non-IP.
- Switched Virtual Channels (SVC) for bi-directional point-to-point and uni-directional point-to-multipoint connections via ATM signalling with constant bit rate (CBR), variable bit rate (VBR), or best-effort service. Supports non-IP traffic only, with or without LLC/SNAP encapsulation. (IP-over-SVC traffic is handled by the IRIS ATM driver via the standard BSD socket interface.)
- Connections with symmetric or asymmetric bandwidth requirements.
- ATM quality of services (QoS) for classes Unspecified, A, B, and D.
- Strict VCI-based packet multiplexing.

Driver Architecture and Theory of Operations

The services of the IRIS ATM subsystem can be accessed using permanent virtual channels (PVCs) or switched virtual channels (SVCs), for IP or non-IP traffic. These four access scenarios are listed below, and are discussed in more detail in the paragraphs that follow:

- Non-IP traffic over PVCs
  The character device interface (IRIS ATM API) allows traffic to be sent constant bit rate, variable bit rate, or best-effort, as requested.
- IP traffic over PVCs
  The character device interface (IRIS ATM API) is used to establish PVCs (using constant bit rate, variable bit rate, or best-effort, as requested) and associate them with IP addresses. LLC/SNAP encapsulation is the default, but can be disabled. The standard BSD socket interface is used for transmit/receive once the PVC is established. IP-to-VC address resolution is handled via a lookup table.
- Non-IP traffic over SVCs
  The character device interface (IRIS ATM API) allows traffic to be sent constant bit rate, variable bit rate, or best-effort, as requested.
- IP traffic over SVCs
  The standard IRIX BSD socket interface to the IP protocol stack allows traffic to be sent best-effort over SVCs. LLC/SNAP encapsulation is done on all packets.[1]

  Note: To use the standard IP socket interface, simply configure the IRIS ATM software, as described in the *IRIS ATM Configuraton Guide*. Once the software is configured, the services of the IRIS ATM subsystem are available to upper-layer IP applications.

Access to the IRIS ATM subsystem is described below and illustrated in Figure 1-1:

- Non-IP data through PVCs
  Applications that use the character device interface for non-IP traffic access the ATM subsystem through IRIS ATM *ioctl()* commands. For each VC, this interface consists of opening a file descriptor (*open()*), using the `ATMIOC_CREATEPVC` command to create the VC, and then exchanging data (*read()*, *write()*, or *writev()*).

- IP-over-ATM traffic through PVCs
  Applications that use the character device interface for IP traffic access the ATM subsystem through IRIS ATM *ioctl()* commands. For each VC, this interface consists of opening a file descriptor (*open()*), using the `ATMIOC_CREATEPVC` command to create the VC with a tag for IP, and the `ATMIOC_SETARP` command to create an address resolution mapping. The *atmarp* PVC management program that is shipped with IRIS ATM creates PVCs in this manner. (See "PVC Management by atmarp" on page 17 for more detail.) When *atmarp* is running, customer applications can simply use the BSD socket interface, as described in the next paragraph.

  Once the PVCs are established, the BSD socket interface is used (*socket()*, *connect()*, *bind()*, *accept()*, *read()*, *write()*, or *writev()*) to exchange data. Address resolution is provided by RFC 1577 software that responds to InverseARP requests and ILMI software, as described in "Address Resolution for IP-Over-PVCs" on page 15.

- Non-IP data through SVCs
  Applications that use the character device interface for non-IP traffic access the ATM subsystem through IRIS ATM *ioctl()* commands. For each VC, this interface consists of opening a file descriptor (*open()*), using IRIS ATM *ioctl()* commands to create the VC (for example, `ATMIOC_SETUP` or `ATMIOC_REGISTER`, `ATMIOC_LISTEN`, and `ATMIOC_ACCEPT`), and then exchanging data (*read()*, *write()*, or *writev()*).

- IP-over-ATM traffic over SVCs through the BSD socket interface
  Applications that use the standard IRIX BSD socket interface for the IP suite of protocols access the services of the IRIS ATM subsystem like other IRIX network subsystems. This interface consists of standard functions (for example, *socket()*, *bind()*, *listen()*, *connect()*, *read()*, *write()*, *writev()*, and standard, non-ATM *ioctl()* calls). This interface is not described in this document. Address resolution is provided by RFC 1577 software that communicates with the subnetwork's ATM address resolution server and ILMI software, both of which are included in the IRIS ATM software.

  Note: For more information on the socket interface, see the reference (man) pages for *accept*(2), *bind*(2), *connect*(2), *fcntl*(2), *getsockname*(2), *getsockopt*(2), *ioctl*(2), *listen*(2), *read*(2), *recv*(2), *select*(2), *send*(2), *socket*(2), *socketpair*(2), *write*(2), and *writev*(2).

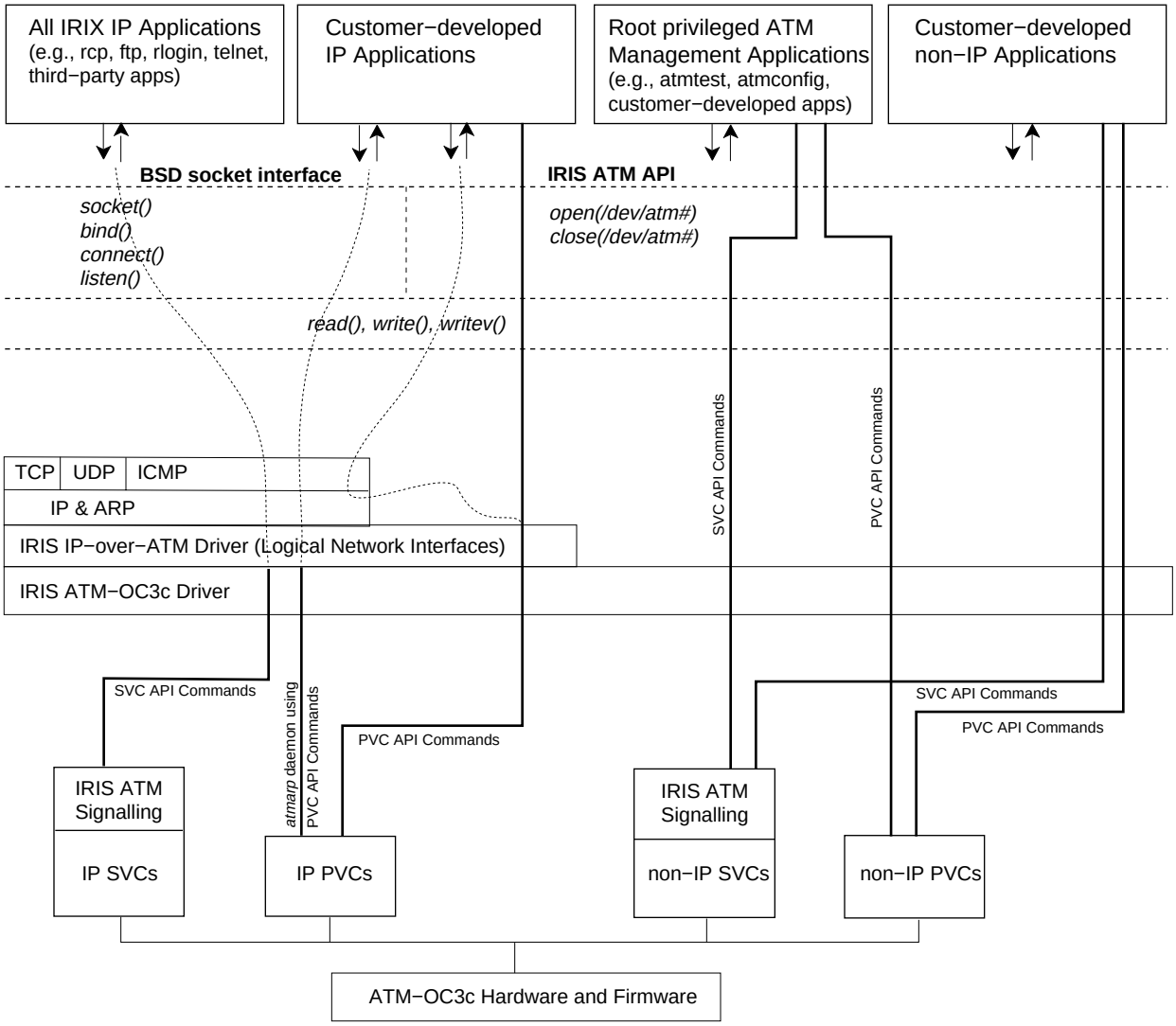


Figure 1-1 IRIS ATM Driver Architecture

Character Device Interface

The character device interface for IRIS ATM supports applications (sending IP or non-IP traffic) that require constant bit rates (CBR), variable bit rates (VBR), or best-effort service, as well as applications that manage, configure, or control the ATM subsystem. Through the character device interface, applications can use any combination of PVCs and SVCs. Standard IP applications that can tolerate best-effort service are encouraged to use the IP-over-SVC support that is built into the IRIS ATM driver via IP logical network interfaces (*atm0*, *atm1*, *atm2*, and so on) and the BSD socket interface.

The ATM subsystem clones its devices, so there is no implicit binding between a VC and a minor device (that is, an ATM port). Because of this design, each hardware device (ATM port) simultaneously supports multiple VCs. There is, however, a one-to-one binding between each file descriptor (the cloned device) and its associated VC; that is, each open file descriptor supports only one VC. These relationships are portrayed in Figure 1-2.

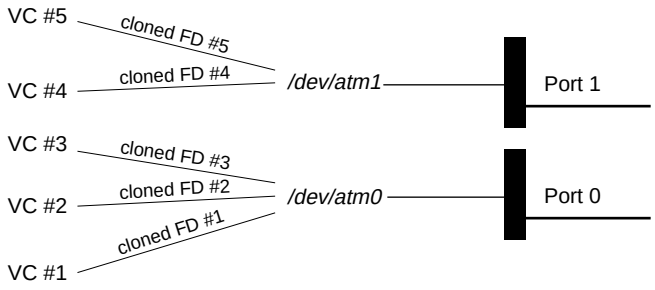


Figure 1-2 Relationship of VCs, File Descriptors, and ATM Hardware

Include Files

The following files define structures and constants that must be used with the ATM character device interface:

- "*sys/atm.h*"
- "*sys/atm_user.h*"
- "*sys/if_atm.h*" (required only for IP-over- PVCs)

open()

When *open()* is invoked on an IRIS ATM device file, the returned file descriptor is a "cloned" instantiation (minor device number) for that ATM board (or port on a multiport board). Each *open()* function establishes kernel-level connections to the selected ATM board. There can be multiple character device interfaces active for each installed ATM card. Each open device services one virtual channel (VC).

Each ATM card has a set of jumpers that sets its unit number. By standard convention, the unit number is reflected in the device file name. For example, an ATM card with jumpers indicating unit 0 has a device file name of */dev/atm0*.

The example below illustrates proper usage where the ATM-OC3c board is identified as unit 0 (*/dev/atm0*):

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int fd_atm;
if ((fd_atm = open("/dev/atm0", O_RDWR)) < 0) {
 perror("open");
 exit(-1);  }
```

Note: At this point, no VC is created; no *read()* or *write()* calls can be made. To create the desired VC, the `ATMIOC_CREATEPVC`, `ATMIOC_REGISTER`, or `ATMIOC_SETUP` *ioctl*() call must be used on the returned file descriptor. The *ioctl()* calls are described in Chapter 2 and Chapter 3.

close()

The *close()* function tears down the bound VC after all the buffered data for the VC has been transmitted. The *close()* results in closing the kernel-level link (minor device) to the ATM-OC3c board, removing the associated VC from the ATM subsystem, and freeing the board and driver resources. The example below illustrates proper usage:

```
#include <unistd.h>
if (close(fd_atm) < 0) {
perror("close"); }
```

read()

The default behavior for *read()*s on an ATM device is blocking; that is, *read()* calls return only after data has been read / made available. However, after opening an ATM file descriptor, non-blocking can be specified, using the standard *ioctl()* FIONBIO. With the default blocking mode, *read()* calls wait for data to become available.With the non-blocking mode, *read()* calls return with an EAGAIN failure whenever no data is available.

For each ATM read-access interface, it is the responsibility of the application to perform enough *read()* calls to consume the data. There is one receive queue for each VC; each queue is 50 AAL convergence sublayer protocol data units (CSPDUs) deep. If an application fails to consume incoming data fast enough and the receive queue in the kernel overflows, PDUs are dropped.

The examples below illustrate correct usage for large- and small-sized data in the current implementation.

Small-Sized Data

For data that occupies less than one page of system memory, the usage illustrated below is correct:

```
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
#include <sys/atm_user.h>
buf = (char*) malloc(size);
retvalue = read(fd_atm, buf, MAX_USER_BYTES_PDU);
```

Large-Sized Data

For data that is greater than or equal to an operating system page of memory, it is recommended that page-aligned buffers be used in order to optimize performance. This optimization is optional. If page-aligned buffers are not provided, the driver retrieves the data by copying it..

```
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
#include <sys/atm_user.h>
buf = (char*) valloc(size);
retvalue = read(fd_atm, buf, MAX_USER_BYTES_PDU);
```

write()

The default behavior for *write()*s on an ATM device is blocking. However, after opening the file descriptor, non-blocking can be specified (using the standard *ioctl()* FIONBIO). In the default blocking mode, *write()*s wait for the DMA to the board to complete before returning. In non-blocking mode, *write()*s return immediately, before the DMA is complete; however, if the previous DMA is not complete, a non-blocking *write()* fails and returns the EAGAIN error.

The list below summarizes two methods for transmitting over the ATM-OC3c subsystem with the ATM character device interface:

1. The *write()* call, using one buffer of any size and resulting in one or more AAL convergence sublayer protocol data units (CSPDUs). The ATM subsystem divides the data into fully filled CSPDUs, and when necessary, pads the final CSPDU.
2. The *writev()* call, using 1 to IOV_MAX buffers (iovecs), and resulting in one or more PDUs (that is, as many PDUs as necessary). The data is concatenated and divided into PDUs. When necessary, incomplete PDUs are padded.

The following rules apply to transmissions:

- All buffers must begin on 8-byte boundaries.
- All buffers must be pinned down.
- In the default blocking mode, calls block until the very last byte of data for the call has DMA'd to the board.
- The buffer (or *iovec*) size can end at any byte position (odd or even). For the *writev()* call, any buffer that is not a multiple of 8 causes the ATM subsystem to pad out the current CSPDU and transmit it. The data from the next *iovec*, if one is present, is placed into a new CSPDU.
- As long as buffers are multiples of 8 bytes, but not of MAX_USER_BYTES_PDU in size, there is no correlation (none, whatsoever) between the *iovec* boundaries and the CSPDU boundaries. That is, the driver does not force new CSPDUs to start on *iovec* boundaries.

  Note: If a buffer is not pinned down, an EFAULT error may occur and it is possible that garbage data will be sent.

Most audio / video applications have one very large buffer (multiple megabytes) in user virtual address space. By starting the first *write()* on an 8-byte boundary, and making every *write()* be a multiple of 8 bytes, all subsequent writes will automatically be properly aligned.

General *write()* Example

The example below demonstrates correct usage:

```
#include <unistd.h>
#include <stdlib.h>
#include <sys/lock.h>
while (needed) {
buf = (char*) memalign(8, size);/* any size */
mpin (buf, size);
retvalue = write(fd_atm, buf, size);
}
```

To Send Multiple Buffers of Data

To send a number of buffers of data, use a *writev()* call, as shown below. This method can result in many CSPDUs. For best performance, the size of each of the buffers, except the last one, should be a multiple of 8 bytes. As long as each buffer size is a multiple of 8, the ATM subsystem concatenates the data, divides it into chunks that completely fill CSPDUs, and transmits it. When the ATM subsystem gathers data that is not a multiple of 8, it places that data into the current CSPDU, pads out the CSPDU and transmits it; the next buffer, if there is one, is contained in a new CSPDU.

```
struct iovec iov[IOV_MAX];
for (vec=0; vec<vec_count, vec++) {
iov.iov[vec].iov_base = (caddr_t) memalign( 8, size );
iov.iov[vec].iov_len = size;
mpin( iov.iov[vec].iov_base, size );
}
retvalue = writev( fd_atm, iov, vec_count );
```

To Gather Data Into One Packet

A number of buffers can be gathered into a single CSPDU with the *writev()* call. The size (length) of each buffer, except the last one, must be a multiple of 8 bytes, and the total data for all the buffers must be less than or equal to MAX_USER_BYTES_PDU.

```
struct iovec iov[IOV_MAX];

for (vec=0; vec < (vec_count), vec++) {
/* size = multiple of 8*/
iov.iov[vec].iov_base = (caddr_t) memalign( 8, size );
iov.iov[vec].iov_len = size;
mpin( iov.iov[vec].iov_base, size );
}

/* total size ≤ MAX_USER_BYTES_PDU */
retvalue = writev( fd_atm, iov, vec_count );
```

To Send One Buffer of Data

To send a single buffer, use the *write()* call. The ATM subsystem divides the data into chunks that completely fill CSPDUs, and transmits the CSPDUs. If the final chunk of data does not completely fill a CSPDU, the ATM subsystem pads it and transmits it. Amounts of data smaller than MAX_USER_BYTES_PDU can be written, and the ATM subsystem does all appropriate padding; however, throughput is adversely affected.

```
char *buf = memalign(8, size);
mpin (buf, size)
retvalue = write(fd_atm, buf, size);
```

IRIS ATM API Command Format

All the IRIS ATM API commands are available through the IRIS character device interface in the following format:

`ioctl(`*fd_atm*`, `*COMMAND*`, `*arg*`);`

Managing and Configuring the ATM-0C3c Subsystem

Before an application can use the IRIS ATM API to utilize the services of an ATM subsystem, one or more control (management) programs must take care of the tasks listed in Table 1-1. The IRIS ATM driver performs these tasks at startup, thus making available a default configuration of the subsystem. For environments using this default configuration, no additional control program is necessary. For environments requiring a non-default configuration, a customer-developed control program must reconfigure the subsystem after the IRIS ATM driver has completed its tasks.

Table 1-1 indicates which ATM *ioctl()* command is used to carry out each task. It is not important if one or many programs are created to perform these tasks; however, the following restrictions apply:

- For any single ATM-OC3c board, each specific task listed in the “Task” column should be performed by only one control program. Chaos can occur if a number of programs are doing the same task to the same board.
- Each task can be performed by a separate control program, or a single program can do all of them.
- The tasks must be performed in the order shown in the “Task” column.
- A program doing the tasks described in the table may (or may not) also do user-data transfers.

- Each task assumes an open file descriptor (cloned minor device) to the board it is configuring. The file descriptor can be closed whenever the program has finished its task(s).

Table 1-1 Configuration Tasks That Must Be Done for Each ATM-OC3c Board

Task (in order)	Calls	Comment	More Info
Configure operational modes	ATMIOC_GETCONF	Retrieve the current configuration.	page 124
	ATMIOC_SETCONF	If changes are needed, set new configuration parameters.	page 147
Configure one or more rate queues, if not correct[a]	ATMIOC_SETRATEQ ATMIOC_SETRATEQ ATMIOC_SETRATEQ ATMIOC_SETRATEQ	rate queue ## rate queue ## rate queue ## rate queue ##	page 153
Monitor status (optional)	ATMIOC_GETSTAT	Retrieve board statistics.	page 135
	ATMIOC_GETIOSTAT	Retrieve driver-internal statistics	page 127

a. See “Characteristics of the ATM-OC3c Hardware” for a description of how IRIS ATM configures and manages transmission rates.

Each application that wants to transfer data through the ATM subsystem must wait until the control program(s) has completed its tasks, then it must obtain a file descriptor and create a VC before reading or writing data. When the data transfer is finished, the application simply closes its file descriptor. The ATM subsystem tears down the VC, cleans up, and releases resources.

Note: When IP applications are going to use the ATM subsystem, there are additional management requirements, as described in the section “IP Support for PVCs.”

IP Support for PVCs

This section describes IRIS ATM support for IP-over-ATM using permanent virtual channels (PVCs).

Address Resolution for IP-Over-PVCs

IRIS ATM address resolution for IP-over-PVC traffic can be thought of as divided into two parts: IP-to-ATM address resolution and IP-to-VC address resolution, as described below:

- IP-to-ATM address resolution consists of obtaining (registering) an ATM address from the adjacent switch or self-assigning this address, and responding to InverseARP requests in order to verify or provide the IP address that is mapped to the ATM address. The first process is handled automatically by ILMI software modules on both the adjacent switch and the local system, and InverseARP is handled automatically by RFC 1577 software on both the local system and the other endpoint.

  Note: On PVCs, IRIS ATM address resolution software responds to received InverseARP requests when LLC/SNAP encapsulation is enabled; however, it does not generate InverseARP requests.

- IP-to-VC address resolution consists of mapping an IP address to a PVC that is identified by a local “hardware” address made from a VPI/VCI value and an ATM port identification number. All the mappings are stored in the kernel-resident ATM address resolution (AR) table. The *atmarp* utility (or equivalently the `ATMIOC_SETARP` command) loads PVC address resolution information into the AR table. The `ATMIOC_GETARPTAB` command retrieves the contents of the table.

The VC address is defined by the `atm_laddr_t` structure, illustrated in Figure 1-3. The `atm_laddr_t` structure fits conveniently into the standard hardware address, `arp_ha` structure, of an `arpreq`.

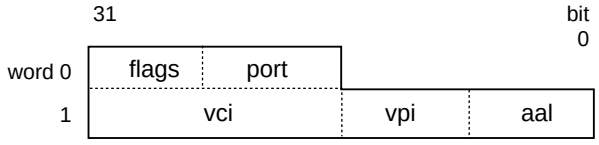


Figure 1-3 ATM Address Resolution Table Entry: the *atm_laddr_t* Structure

The ATM-specific *ioctl()* calls that are available for address resolution are listed below and described in Chapter 2.

- `ATMIOC_SETARP` (add an entry to the AR table)
- `ATMIOC_GETARP` (retrieve one entry from the table)
- `ATMIOC_DELARP` (delete an entry from the AR table)
- `ATMIOC_GETARPTAB` (retrieve the entire table)

Address resolution and internal routing of IP packets is handled in the following manner: the `ATMIOC_CREATEPVC` command with the IP flag set to ON and the `ATMIOC_SETARP` command create the links between the IP interface (if_net) and the PVC that allow incoming and outgoing IP packets to be routed correctly.

LLC/SNAP Encapsulation for PVCs

Each PVC can be configured to perform or not to perform subnetwork access protocol encapsulation (802.2 LLC / SNAP) for packets on VCs associated with an IP logical network interface. When LLC / SNAP encapsulation is enabled for a VC, the LLC and SNAP headers are attached to every packet on that VC, thus allowing ATM subsystems to differentiate among upper layer protocol stacks (for example, IP and ARP). When LLC / SNAP is enabled on a VC, the IRIS ATM subsystem responds to InverseARP requests. When LLC / SNAP encapsulation is disabled, IP packets on that VC are not encapsulated and InverseARP requests are not answered. The default behavior is to do LLC / SNAP encapsulation.

Configuration of LLC/SNAP encapsulation for each PVC can be done by either of the following methods:

- edit the IP-to-PVC address resolution table and let the VCs be opened and configured by the IRIS ATM *atmarp* utility
- set the configuration for each PVC when it is created with the `ATMIOC_CREATEPVC` command

IRIS ATM Subsystem Management for IP-Over-PVCs

Before any IP applications can utilize IP-over-PVC services, one or more control (management) programs must take care of the tasks listed in Table 1-2. For most implementations, the default control provided by the IRIS ATM utility *atmarp* (which is invoked during startup) is sufficient.

PVC Management by *atmarp*

During system startup, the */etc/init.d/network.atm* script starts the *atmarp* PVC management application if the */var/atm/pvc.conf* file exists. This user-configurable file maps IP addresses to local ports and VPI/VCI addresses. For each entry in the table, *atmarp* opens a file descriptor for the indicated port, and makes an `ATMIOC_CREATEPVC` and an `ATMIOC_SETARP` *ioctl()* call in order to establish a best-effort PVC and associate it with an IP address. The *atmarp* utility then goes to sleep, leaving the VCs open and ready for use. (If the file descriptors were to be closed, the PVCs would be torn down.) At this point, an IP application that opens a socket to any of the IP addresses in the table transmits/receives over the associated PVC. If *atmarp* is interrupted with a SIGHUP signal (for example, *killall -HUP atmarp*) it wakes up, reloads the lookup table from the *pvc.conf* file, makes any changes necessary by closing file descriptors (for deleted entries) or establishing new PVCs (for new entries), then goes back to sleep.

PVC Management by a Customer-Developed Application

For implementations that do not wish to use *atmarp* to manage their PVCs, the following guidelines should be adhered to when designing the management application. It is not important if one or many programs are created to perform these tasks; however the following restrictions apply:

- The tasks must be performed in the order shown in the "Task" column of Table 1-2.
- Before doing any of the tasks listed in Table 1-2, the tasks in Table 1-1 must be performed, either by another control program or by the same program doing the tasks listed in Table 1-2.
- The management program doing these tasks may (or may not) read / write over these VCs.
- The management program must keep the file descriptor open for the entire duration of the PVC's use.

Table 1-2 Configuration Tasks That Must Be Done for Each ATM Network Interface Servicing IP if *atmarp* Is Not Running

Task (in order)	Calls	Comment	More Info
Open as many file descriptors for the board as there will be PVCs.	fd1=open("/dev/atm0") fd2=open("/dev/atm0") fd3=open("/dev/atm0") fd4=open("/dev/atm0") etc.	The control program must keep each file descriptor open as long as the associated PVC is being used.	page 7
Create one virtual channel for each file descriptor.	ATMIOC_CREATEPVC ATMIOC_CREATEPVC etc.	Each *ioctl()* call creates one virtual channel with a cellrate that is as close as possible to the requested rate. Tag each VC for IP.	page 31
Manage ATM address resolution.	ATMIOC_SETARP ATMIOC_SETARP etc.	Create an IP-to-VC mapping in the ATM subsystem's address resolution table for each IP endpoint. Each SETARP *ioctl()* call creates one entry.	page 48
Tear down a PVC.	close(*fd#*)		page 8
Monitor the AR table (optional).	ATMIOC_GETARPTAB		page 42

When the control program closes a file descriptor, the ATM subsystem automatically tears down the associated VC, cleans up the address resolution table, and releases the associated resources.

Each IP application that wants to transfer data through the ATM subsystem simply does what all IP applications do (*socket*(), *bind*(), *connect*(), *accept*(), and so on) before reading or writing data. When the data transfer is finished, the application closes its socket. The ATM subsystem does not tear down the VC; only closing the file descriptor tears down the VC.

Characteristics of the ATM-OC3c Hardware

The IRIS ATM-OC3c for CHALLENGE® and Onyx™ board manages transmission rates with rate queues and divisors. The board has eight rate queues organized as two banks: a0-a3 and b0-b3. Each queue can support one peak rate and 63 different sustainable rates. The “a” bank consists of four high-priority queues that are designed for constant bit rate traffic (CBR and VBR channels). The other bank contains four low-priority queues and are only used for best-effort traffic.

High-priority queues are serviced before low-priority ones. As long as there is data awaiting transfer on any high-priority queue, low-priority data is not transmitted. This means that, for applications with a constant flow of data, only queues a0-a3 will ever operate.

During startup, the IRIS ATM driver configures each rate queue, as explained below:

1. Queues that are mentioned in the */var/atm/atmhw.conf* file are configured to a fixed rate, as specified in the file. The IRIS ATM driver never changes the rates for these queues; this ensures that site-specified rates are always available, even when the queues are not actively being used. Appendix A lists the supported rates, which range from 0 to 135,991,460 bits per second.
2. Queues that are not mentioned (or are commented out) in the file are left unconfigured. The driver configures these during operation.

During operation, as VCs are created, the driver associates each newly created VC with the queue whose transmission rate best matches the peak rate requested for that VC. For each `ATMIOC_CREATEPVC` or `ATMIOC_SETUP` command, the driver looks for a queue whose transmission rate best matches the rate requested in the API call, following the guidelines explained below:

1. For VCs carrying best-effort traffic, the driver uses the low-priority queue whose rate is closest to, but slower than, the requested peak rate.
2. For VCs carrying CBR and VBR traffic, the driver uses the high-priority queue whose configured rate exactly matches the requested peak rate. If the requested rate does not exist, the driver searches for a high-priority queue with the following characteristics and reconfigures it to the requested peak rate:
   - a queue that does not currently have a VC associated with it
   - a queue that was not configured from the *atmhw.conf* file during startup

   Note: There can be dozens of CBR and VBR virtual channels active on a board, but the peak rate for each one must be one of the four rates that are configured on the high-priority queues.

To set the sustainable transmission rate for a particular VC, one of the board's configured rates is divided by a divisor (ranging between 1 and 64). The IRIS ATM driver sets all divisors. Peak rates for CBR, VBR, or best-effort traffic use divisors of 1. Sustainable (average) rates for VBR traffic use divisors from 2 through 64 (inclusive).

To summarize, the IRIS ATM-OC3c board simultaneously makes available for selection up to 8 different peak rates and up to 504 (8x63) sustainable rates. Not all of these available selections can be actively used simultaneously, since this exceeds the board's bandwidth.

Table 1-3 summarizes the default settings configured for the IRIS ATM-OC3c board's rates.

Table 1-3 Default Transmission Rates on ATM-OC3c Queues

Rate Number Id	Queue String Id	Default Cellrate (in ATM cells per second)	Default Bit Rate (in user payload bits per second)	Priority / Use
0	a0	unconfigured	none	High / CBR, VBR[a]
1	a1	unconfigured	none	High / CBR, VBR
2	a2	unconfigured	none	High / CBR, VBR
3	a3	unconfigured	none	High / CBR, VBR
4	b0	26041	10000000	Low / BE
5	b1	78125	30000000	Low / BE
6	b2	178571	68000000	Low / BE
7	b3	357142	135991460	Low / BE

a. CBR = constant bit rate; VBR = variable bit rate; BE = best-effort

A board is oversubscribed when the sum of all the open VCs multiplied by their average transmission rates is greater than the board's total payload bandwidth.[1] The IRIS ATM software contains a number of features that prevent performance degradation due to oversubscription. Whenever there is even one VC open for a CBR traffic contract, the IRIS ATM software refuses to create new VCs once the board's total payload bandwidth is allocated to open VCs (including best-effort)[2]. If all the VCs on a board are best-effort (regardless of which queues they are using), the IRIS ATM software allows the board to become oversubscribed and handles the transmission in the best manner possible.

---

1 When a VC does not specify a sustainable rate, the average rate that is used for this calculation is the peak rate.

2 Total OC3 bandwidth is 155.52 megabits per second; however, of this, only 135,991,460 is available for user data, and is referred to as the payload bandwidth.

Note: The default TCP / IP configuration uses the maximum bandwidth for any connection. Therefore, a single TCP / IP connection can oversubscribe the port it uses and prevent CBR traffic. To prevent this, there are two options: (1) reduce the default TCP / IP bandwidth (for example, by editing the */var/atm/ifatm.conf* file) or (2) use *ifconfig* to disable the TCP / IP logical network interfaces.

User-Level Commands

The IRIS ATM software includes utilities in the */usr/etc* directory (*atmarp*, *atmconfig*, *ifatmconfig*, *atmstat*, and *atmtest*) and the */usr/lib/atm/bin* directory (*sigtest*). Each utility is briefly described below. Complete details are provided in the online reference (man) pages.

atmarp

The *atmarp* utility provides command-level support for displaying and reloading the IP-to-ATM address resolution table. Also, it operates as an IP-to-PVC address resolution daemon, managing the mappings between VCs, ATM hardware, and ATM logical network interfaces.

Note: The */etc/init.d/network.atm* IP startup script invokes this utility during each system startup or each invocation of the script. The command loads the contents of the */var/atm/pvc.conf* IP-to-VC address mapping file into the kernel-resident address resolution table, maintains the file, and responds to address resolution requests.

atmconfig

The *atmconfig* utility provides command-level support for on-the-fly configuring and controlling of the ATM hardware:

- configuring the state of ATM boards: UP / DOWN
- configuring transmission rates on rate queues
- configuring the size and the number of board transmit and receive buffers: both large and small
- burning firmware into FLASH EEPROM
- resetting and reinitializing the board

ifatmconfig

The *ifatmconfig* utility provides command-level support for setting RFC 1577 Logical IP Subnetwork (LIS) parameters, such as the ATM address resolution server, the time out for inactive VCs, the maximum cellrate to use for the VCs, and the ATM physical port to use for each LIS. Each ATM LIS appears as a logical network interface that can be given an IP address and enabled / disabled with *ifconfig* just like other conventional network devices.

Note: The IRIS ATM startup script (*/etc/init.d/atm*) invokes this utility during each system startup or each invocation of the script, telling it to read the */var/atm/ifatm.conf* LIS configuration file for settings of these parameters.

atmstat

The *atmstat* utility provides command-level support for monitoring the status and operational statistics of ATM interfaces and ATM-OC3c boards.

atmtest

The *atmtest* utility provides command-level support for testing data transmission over the ATM subsystem when it is physically looped back (that is, an ATM-OC3c output is connected to an ATM-OC3c input). Command line options allow you to control parameters such as the length of the randomly generated data and the speed at which it is sent.

sigtest

The *sigtest* utility provides command-level support for testing data transmission and reception for switched virtual channels. The program allows you to create the following types of connections:

- A point-to-point loopback connection through the switch: a transmitting VCC to the switch that feeds into a receiving VCC from the switch. The transmitter and receiver are two instances of *sigtest* running on the same system.
- A point-to-point connection between two different systems that are both running *sigtest*.
- A point-to-multipoint connection in which the members of the party (the receivers) can include any combination of the following: one receiving *sigtest* session on the same system that is setting up the call and one receiving *sigtest* session on each remote system.

IRIS ATM *ioctl()* Commands for Permanent VCs

This chapter summarizes the IRIS ATM application interface calls that support permanent virtual channels (PVCs). These commands are described alphabetically in the subsections that follow, and are summarized in Table 2-1.

Note: The IRIS ATM *atmarp* utility handles IP-to-VC address resolution for PVCs that carry IP traffic. When *atmarp* is running, the commands in Table 2-1 under the heading "Address Resolution for IP-over-ATM When *atmarp* is Not Running" do not need to be used. These commands are provided for management implementations that do not wish to utilize the *atmarp* utility. See the *atmarp* reference (man) page for further details.

Table 2-1 Summary of ATM PVC *ioctl()* Calls

Type of Operation	Command (or function)	Brd State	Description	More Info
Getting a link to the ATM-subsystem	*open()*	all	Opens a file descriptor for a cloned device. Must be held open as long as the bound VC is active.	page 7
Tearing down a VC	*close()*	all	Closing the file descriptor causes the VC to be torn down and all resources released.	page 8
Managing transmission rates on the OC3c board				
	`ATMIOC_SETRATEQ`	up / dn	Sets rate for one of the 8 rate queues.	page 153
	`ATMIOC_GETRATEQ`	up	Reads rate for the indicated rate queue.	page 132
Managing PVCs				
	`ATMIOC_CREATEPVC`	up	Binds one pair of virtual path / virtual channel identifiers to a cloned file descriptor.	page 31

Table 2-1 (continued) Summary of ATM PVC *ioctl()* Calls

Type of Operation	Command (or function)	Brd State	Description	More Info
	ATMIOC_GETVCTAB	up	Retrieves entire virtual channel table.	page 45
Address resolution for IP-over-ATM				
	ATMIOC_GETARPTAB	up / dn	Retrieves the entire IP-to-ATM address resolution table.	page 42
	ATMIOC_GETARP	up / dn	Retrieves one entry from the ATM address resolution table.	page 40
Address resolution for IP-over-ATM when *atmarp* is not running				
	ATMIOC_SETARP	up / dn	Sets a static entry in IP-to-ATM address resolution table. AR table maps IP addresses to `atm_laddr_t` structures.	page 48
	ATMIOC_DELARP	up / dn	Deletes one entry from IP-to-ATM AR table.	page 38
Managing data				
	write()	up	Pinned down, 8-byte aligned buffer of any size. If necessary, ATM subsystem divides data into different packets for transmission.	page 9
	writev()	up	Gathers data from a number of buffers for transmission as one or more packets.	page 9
	read()	up	Retrieves incoming data.	page 8

Include Files for PVCs

The following files must be included in any program using the ATM-specific *ioctl()* calls:

- "*sys/atm.h*"
- "*sys/atm_user.h*"
- "*sys/if_atm.h*" (only for applications doing IP-over-ATM)

Frequently Used Structures

Some structures are used as arguments for many of the ATM-specific *ioctl()* calls. For reference, these frequently used structures are described below.

The atm_laddr_t Structure

The `atm_laddr_t` structure is the ATM subsystem's local "hardware address" used for IP-to-VC address resolution (that is, the IRIS ATM "ARP" for PVCs) commands. For IP-over-PVCs, the structure is used within the standard `arpreq` structure. Table 2-2 and the following paragraphs describe the `atm_laddr_t` structure and its usage.

Table 2-2 IRIS ATM Local "Hardware" Address: *atm_laddr_t*

Field of atm_laddr_t	Recommended Value	Comments
port	0 - 11	Board's unit number. The unit number can be determined with the */sbin/hinv* command. The value must be less than ATM_MAXBD.
flags	none	Used internally by IRIS ATM software.
aal	AALTYPE_5	Currently, only AAL5 is supported.
vpi	0 - 255 (decimal)	Virtual path identifier.
vci	0 - 65535 (decimal)	Virtual channel identifier. The VPI/VCI combination must be currently unused (available) both locally and on the switch.

From the *if_arp.h* file:

```
struct arpreq {
   struct sockaddr arp_pa; /* protocol address */
   struct sockaddr arp_ha; /* hardware address */
                           /* for ATM = atm_laddr_t*/
   int arp_flags;
};
```

From the *socket.h* file:

```
struct sockaddr {
   u_short sa_family; /* address family */
   char sa_data[14]; /* up to 14 bytes of direct address */
};
```

From the *atm_user.h* file:

```
typedef struct atm_laddr {
   u_char port; /* local port number; brd's unit nmbr*/
   u_char flags; /* flags - local use only */
   u_char aal; /* aal type - local use only */
   u_char vpi; /* remote VPI */
   u_short vci; /* remote VCI */
} atm_laddr_t;
```

From the *atm_b2h.h* file (included in the *atm_user.h* file), values for the *aal* field of `atm_laddr_t`:

```
#define AALTYPE_34 0
#define AALTYPE_5 1
#define AALTYPE_CBR 6
#define AALTYPE_RAW 7
```

PVC Code Sample

This section provides a simple code example showing creation, use and tear down of one PVC.

```
/* open a file descriptor */
fd = open( “/dev/atm0”, rw );
   if ( fd < 0 )
   perror( “couldn’t open device” ),exit(1);

/* define the VC’s parameters */
vpi = <your value>
vci = <your value>
xmitMaxCSDU = <your value>
recvMaxCSDU = <your value>
cellrate_type = <your value>
cellrate_peak_rate = <your bits-per-second/384>
cellrate_sustainable_rate = <your bits-per-second/384>
cellrate_maxburst_size = <your value>


/* prepare the argument for ATMIOC_CREATEPVC with VC’s */
/* parameters */
atm_createpvc_t  pvcreq;
bzero( &pvcreq, sizeof(pvcreq) );

pvcreq.vpi = vpi;
pvcreq.vci = vci;
pvcreq.xmitMaxCSDU = xmitMaxCSDU;
pvcreq.recvMaxCSDU = recvMaxCSDU;
pvcreq.xmitcellrate.cellrate_type = cellrate_type;


/* then one of these two sets,  */
/* depending on which type was used */
/* this for CRT_PEAK_AGG or CRT_BEST_EFFORT */
pvcreq.xmitcellrate.rate.pcr_01.pcr01 = cellrate_peak_rate;


/* or this set for CRT_PSB_AGG */
pvcreq.xmitcellrate.rate.psb_01.pcr01 = cellrate_peak_rate;
pvcreq.xmitcellrate.rate.psb_01.scr01 = cellrate_sustainable.rate;
pvcreq.xmitcellrate.rate.psb_01.mbs01 = cellrate_maxburst_size;
```

```
/* create the VC */
if ( ioctl( fd, ATMIOC_CREATEPVC, &pvcreq ) < 0 )
   perror( “couldn’t ATMIOC_CREATEPVC” ),exit(

/* the VC can now be written and read
write(fd, obuf, length); #follow the guidelines in Chapter 1
read(fd, ibuf, length ); #follow the guidelines in Chapter 1

/* to tear down the VC */
error = close( fd, rw );
   if ( error != 0 )
      perror( “couldn’t close device” ),exit(1);
```

PVC Commands

This section describes each ATM PVC *ioctl()* command in detail. The commands are organized alphabetically.

ATMIOC_CREATEPVC

The `ATMIOC_CREATEPVC` *ioctl()* command creates a permanent virtual channel. A successful call binds an open (cloned) file descriptor to one (a read-only or write-only) or two (a read and a write) virtual channel connections (VCCs), creates entries in the appropriate VC tables, and allocates board resources. Each VCC is identified by a VC address: virtual path identifier (VPI) and virtual channel identifier (VCI). The call creates a single VCC when the open file descriptor is read-only or write-only; it creates two VCCs (one forward and one back, using the same VC address for each) when the file descriptor is read and write. Only one `ATMIOC_CREATEPVC` can be called for each open (cloned) file descriptor. Only one PVC is allowed for each VPI/VCI pair. The software prevents creation of a second VCC to the same VPI/VCI pair.

Creating a PVC for a readable file descriptor causes the ATM subsystem to send all incoming PDUs (received on the incoming VCC) up to the application. Received PDUs are buffered in the kernel in per-VC queues. Cells received for a VPI/VCI address that has not been created are discarded by the ATM subsystem.

The board must be in the UP state.

Note: To tear down the VC, simply close the file descriptor. The IRIS ATM subsystem tears down the VC, releases resources, and cleans up.

Usage

Use the following format:

`ioctl (`*`fd_atm`*`, ATMIOC_CREATEPVC, &`*`createpvc`*`);`

where *createpvc* is an `atm_createpvc_t` structure.

Argument Values

The pointer to *createpvc* identifies an instance of an `atm_createpvc_t` structure that is set up as shown in Table 2-3.

Table 2-3 Recommended Values for ATMIOC_CREATEPVC's Argument

Field of atm_createpvc_t	Recommended Value	Comments
vpi	0 - 0xFF	Virtual path identifier. Value must match the one used by the switch for this VC and, if servicing IP traffic, the one used in any local IP-to-VC address mapping file.
vci	0 - 0xFFFF	Virtual channel identifier. Value must match the one used by the switch for this VC.[a]
xmitMaxCSDU	up to 0x2FF8	Maximum size for user-level packets (PDUs). Value cannot be 0 or larger than MAX_CS_PDU, and must be divisible by 8.
recvMaxCSDU	up to 0x2FF8	Maximum size for user-level packets (PDUs). Value cannot be 0 or larger than MAX_CS_PDU, and must be divisible by 8.
flags	as desired	0 = no flags; default functionality, or one or more of the following flags: ATMPVCFL_IP = the VC is servicing an IP logical network interface. If this flag is set, the command ATMIOC_SETARP must be used to bind this VPI/VCI to an IP address. ATMPVCFL_NOSNAP = do not attach 802.2 LLC/SNAP encapsulation on the packets on this VC.
xmitcellrate	cellrate_t *Upon return =out value*	Set up as described in Table 2-4. Out value: actual value for the VC.

a. VPI/VCI values 0/0-32 are reserved by the ATM standards for use by ATM signalling and ILMI modules.

The `cellrate_t` structure defines the traffic parameters for the PVC. The supported values are described in Table 2-4 where CR stands for cellrate expressed in cells per second. The specified peak cellrate must match one of the rates on the board's transmission rate queues. See "Characteristics of the ATM-OC3c Hardware" in Chapter 1 for a description of the transmission rate queues and how they are configured.

Table 2-4 Supported Values for Traffic Parameters of ATMIOC_CREATEPVC

Fields of cellrate_t Structure	Possible Values	Description
cellrate_type:	CRT_NULL	Zero bandwidth.
	CRT_PEAK_AGG	Aggregate peak CR for CLP0+1. CBR traffic.
	CRT_PSB_AGG	Aggregate peak CR, sustainable CR, and burst size for CLP 0+1. VBR traffic.
	CRT_BEST_EFFORT	Peak CR for CLP0+1 with best-effort indication.
	CRT_PEAK	Not supported in this release. Peak CRs[a] for CLP0 and CLP0+1.
	CRT_PEAK_TAG	Not supported in this release. Same as above with tagging requested.
	CRT_PSB	Not supported in this release. Peak CR for CLP0+1, sustainable CR for CLP0, burst size for CLP0.
	CRT_PSB_TAG	Not supported in this release. Same as above with tagging requested.
rate:		
for type CRT_PEAK_AGG	struct pcr_01:	
	pcr01	Peak CR for CLP 0+1. If all high-priority rate queues are in use, this value must match one of the configured rates.

Table 2-4 (continued) Supported Values for Traffic Parameters of

Fields of cellrate_t Structure	Possible Values	Description
for type CRT_PSB_AGG	struct psb_01:	
	pcr01	Peak CR for CLP 0+1. If all high-priority queues are in use, this must match one of the configured rates.
	scr01	Sustainable CR for CLP 0+1. PCR divided by SCR must be equal to or less than 64.
	mbs01	Max burst size for CLP 0+1 in cells per burst. Valid values are multiples of 32 between 1 and 2048, inclusive. Zero is invalid.
for type CRT_BEST_EFFORT	struct pcr_01:	
	pcr01	Peak CR for CLP 0+1. IRIS ATM subsystem assigns VC to a low-priority rate queue that is equal to or slower than the rate specified; if necessary, driver divides one of the configured rates to create a slower rate. If specified rate is slower than the slowest configured low-priority rate queue divided by 64, then the rate cannot be supported.
for types CRT_PEAK CRT_PEAK_TAG CRT_PSB CRT_PSB_TAG	not applicable	Not supported in this release.

a. CR or cr = cellrate expressed in cells per second. For example, a CR of 100 means that 4800 bytes of user data (100 cells * 48 bytes of payload for each ATM cell) are transmitted each second.

Success or Failure

If successful, ATMIOC_CREATEPVC returns zero. The out values should be read.

On failure, the *ioctl()* returns -1 with an error stored in errno. See the "Errors" heading for descriptions of individual errors.

Out Values

When the VC is successfully created, the actual values that were used to create the VC are written to the call's argument. The *xmitcellrate* value should be read and verified since it may be different from the requested value.

When the ATMIOC_CREATEPVC fails, the values in the argument do not change and are not meaningful.

Relevant Structures

Below is the atm_createpvc_t structure, as defined in the *sys/atm_user.h* file:

```
typedef struct {
    u_short vpi;
    u_short vci;
    u_short xmitMaxCSDU, recvMaxCSDU;
    u_char flags;
    cellrate_t xmitcellrate;
} atm_createpvc_t;
```

```
typedef struct {
    char cellrate_type;
    union {

   /* for cellrate_type = CRT_PEAK, CRT_PEAK_TAG */
      struct {
      int pcr0;
      int pcr01;
      } pcr_0_01;
```

```
    /* for cellrate_type = CRT_PEAK_AGG, CRT_BEST_EFFORT */
       struct {
       int pcr01;
       } pcr_01;

    /* for cellrate_type = CRT_PSB, CRT_PSB_TAG */
       struct {
       int pcr01;
       int scr0;
       int mbs0;
       } psb_0_01;

    /* for cellrate_type = CRT_PSB_AGG */
       struct {
       int pcr01;
       int scr01;
       int mbs01;
       } psb_01;

    } rate;
} cellrate_t;
```

Errors

Possible errors include:

EADDRINUSE	The VCI value is already in use by another VC.
EFAULT	An error occurred as the driver was copying in the command's *createpvc* argument.

EINVAL	The specified type of cellrate is not supported. Or, the specified cellrate is invalid for the type of cellrate. (For example, for a best-effort type, the slowest configured low-priority rate is still too fast, or for peak aggregate, all the high-priority queues are in use or are configured at a fixed value and none of their rates matches the value specified for `pcr01`). Or, the specified maximum CSDU size is larger than MAX_CS_PDU (that is, 12kilobytes - 8bytes). Or, there is no open file descriptor.
ENODEV	The board is not UP.
ENOMEM	The board was unable to allocate enough on-board memory to complete this task.
ENOSPC	The maximum number of supported open VCs (MAX_FWD_VCS or MAX_RVS_VCS) are already created. Or, the board is out of buffers for the PDU size specified in the argument. Or, the board is out of resources (all the bandwidth is currently occupied by other open VCs).

ATMIOC_DELARP

The `ATMIOC_DELARP` *ioctl()* command deletes one static PVC entry from the IP-to-ATM address resolution table.

Usage

Use the following format:

```
ioctl (fd_atm, ATMIOC_DELARP, &arp);
```

where *arp* is an instance of `arpreq`.

Argument Values

The pointer to *arp* identifies an instance of an `arpreq` structure that indicates which entry in the ATM address resolution table is to be removed. The `arpreq` structure must be set up as described in Table 2-5.

Table 2-5 Recommended Values for ATMIOC_DELARP's Argument

Field of arpreq_t	Recommended Value	Comments
arp_pa	IP address	In *sa_family* field, set the protocol family to AF_INET, and, in *sa_data* field, provide the IP address of remote system.
arp_ha	none	This field is ignored.
arp_flags	none	

Success or Failure

If successful, `ATMIOC_DELARP` returns zero.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Relevant Structures

The `arpreq` and `atm_laddr_t` structures are described for reference in “Frequently Used Structures” on page 27.

Errors

Possible errors include:

EAFNOSUPPORT	The address family specified in the protocol portion of the `arpreq` structure is not AF_INET.
EFAULT	When attempting to copy the data, an error occurred.
EINVAL	An invalid entry occurred during processing of the address resolution. It may be that the requested address was not found in the AR table.
ENODEV	The board was not in the UP or DOWN state.

ATMIOC_GETARP

The `ATMIOC_GETARP` *ioctl()* command retrieves the mapping for one static PVC entry from the IP-to-ATM address resolution table.

Usage

Use the following format:

```
ioctl (fd_atm, ATMIOC_GETARP, &arp);
```

where *arp* is an `arpreq` structure.

Argument Values

The pointer to *arp* identifies an instance of a standard `arpreq` structure defining the protocol address half of the IP-to-ATM address resolution entry to be retrieved.

The `arpreq` structure should be set up as shown in Table 2-6.

Table 2-6 Recommended Values for ATMIOC_GETARP's Argument

Field of arpreq_t	Recommended Value	Comments
arp_pa	AF_INET and IP address	In *sa_family* field, set the protocol family to AF_INET, and, in *sa_data* field, provide the IP address of remote system.
arp_ha	none	
	Upon return =out value	Out value: retrieved `atm_laddr_t` structure. See Table 2-2 for description.
arp_flags	none	

Success or Failure

If successful, `ATMIOC_GETARP` returns zero. The out values should be read.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Out Values

The retrieved PVC "hardware" address is written as an `atm_laddr_t` structure within the *arp_ha* field of the argument.

Relevant Structures

The `arpreq` and `atm_laddr_t` structures are described for reference in "Frequently Used Structures" on page 27.

Errors

Possible errors include:

EAFNOSUPPORT	The address family specified in *arp_pa* is not supported.
EFAULT	When attempting to copy the data, an error occurred.
ENODEV	The board was not in the UP or DOWN state.
ENXIO	The *arp_pa* specified in the argument was not found in the ATM address resolution table.

ATMIOC_GETARPTAB

The `ATMIOC_GETARPTAB` *ioctl()* command retrieves the entire contents of the IP-to-ATM address resolution table. The retrieved entries include all PVCs that, at creation, were tagged with the ATMPVCFL_IP flag (even those that do not have an IP address assigned).

Usage

Use the following format:

```
ioctl (fd_atm, ATMIOC_GETARPTAB, &sioc);
```

where *sioc* is an `atmsioc_t` structure.

Argument Values

The pointer to *sioc* identifies an instance of an `atmsioc_t` structure, set up as shown in Table 2-7. Within *sioc*, the **ptr* field must be a pointer to an array of `atm_arptab_t` structures.

Table 2-7 Recommended Values for ATMIOC_GETARPTAB's Argument

Field of atmsioc_t	Recommended Value	Comments
*ptr	pointer to `atm_arptab[ ]`	Start address where retrieved ATM address resolution table is written.
	Upon return =out value	Out value: array of `atm_arptab_t` structures
len	= sizeof(atm_arptab[ATMARP_TABLESZ∗2])	Maximum possible size of table.
	Upon return =out value;	Out value: length of retrieved table.

Success or Failure

If successful, `ATMIOC_GETARPTAB` returns zero. The out values should be read.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Out Values

The *len* field in the argument (*sioc*) is updated to contain the actual length of the retrieved data. The retrieved table is written to the `atm_arptab[]`. Each table entry is one `atm_arptab_t` structure, described in Table 2-8.

Table 2-8 Values Retrieved by ATMIOC_GETARPTAB

Field in atm_arptab_t	Type	Description
iaddr	struct in_addr *Upon return =out value*	Out value: IP address
atmaddr	struct atm_address_t *Upon return =out value*	Out value: ATM address, if one exists.
laddr	struct atm_laddr_t *Upon return =out value*	Out value: local "hardware" address: VPI, VCI, PT. See "The atm_laddr_t Structure" on page 27.
flags	u_char *Upon return =out value*	Out value: entries from Table 2-9.

Table 2-9 Flags Retrieved by ATMIOC_GETARPTAB

Flag	Description
COMPL	The ATM address for this IP address has been obtained.
CONN	The connection has been established for the VC.
NAK	The ATMARP server has responded that it does not recognize this endpoint.

Table 2-9 (continued) Flags Retrieved by ATMIOC_GETARPTAB

Flag	Description
NOSNAP	The VC is not using LLC/SNAP encapsulation.
PEND	The connection has not yet been established; it is pending setup completion.
PVC	The VC is a permanent virtual channel, not a switched one.
VALIDATE	The IP address is in the process of being validated with InverseARP.

Relevant Structures

The `atmsioc_t` is described below, for reference. The `atm_arptab_t` structure is described in Table 2-8. The `atm_laddr_t` structure is described on page 27.

The `atmsioc_t`, as defined in the *sys/atm_user.h* file:

```
typedef struct atmsioc {
   void *ptr;/* where data is located */
   u_int len;/* size of structure at *ptr */
} atmsioc_t;
```

The `atm_arptab_t` structure, as defined in the *if_atm.h* file:

```
typedef struct atm_arptab {
   struct in_addr iaddr;
   atm_address_t atmaddr;
   atm_laddr_t laddr;
   u_char flags;
} atm_arptab_t;
```

Errors

Possible errors include:

EFAULT	When attempting to copy the data, an error occurred.
ENODEV	The board was not in the UP or DOWN state.

ATMIOC_GETVCTAB

The ATMIOC_GETVCTAB *ioctl()* command retrieves the entire virtual channel table (both transmit and receive VCs). The board must be in the UP state.

Usage

Use the following format:

```
ioctl (fd_atm, ATMIOC_GETVCTAB, &sioc);
```

where *sioc* is an atmsioc_t structure.

Argument Values

The pointer to *sioc* identifies an instance of an atmsioc_t structure. The *sioc* should be set up as summarized in Table 2-10.

Table 2-10 Recommended Values for ATMIOC_GETVCTAB's Argument

Field of atmsioc_t	Recommended Value	Comments
*ptr	=pointer to vct[]	Pointer to location for retrieved information.
	Upon return =out value	Out value: an array of atm_vcte_t structures.
len	=sizeof(vct[MAX_FWD_VCS+MAX_RVS_VCS]);	Maximum possible size of the table.
	Upon return =out value	Out value: length of retrieved table.

Success or Failure

If successful, ATMIOC_GETVCTAB returns zero. The out values should be read.

On failure, the *ioctl()* returns -1 with an error stored in errno. See the "Errors" heading for descriptions of individual errors.

Out Values

The *len* field in the argument (*sioc*) is updated to contain the actual length of the retrieved data, as described in Table 2-10. The retrieved data is written to the array of `atm_vcte_t` structures. Each table entry is one structure, as described in Table 2-11.

Table 2-11 Values Retrieved by ATMIOC_GETVCTAB

Field of atm_vcte_t	Type	Description
cell_hdr	u_int	VPI=bits 27:20; VCI=bits 19:4; PT=bits 3:0
max_cs_pdu_size	u_int	Maximum PDU size on this VC.
burst_size	u_short	Maximum burst allowed. A burst is the maximum number of back-to-back cells transmitted at peak cellrate (CQ). 32 modulo bucket depth.
rate_queue_number	u_char	Rate queue ID. The configured rate on this queue is the peak cellrate for this VC.
avg_rate_divisor	u_char	The peak cellrate is divided by this value to give the average or sustainable cellrate for the VC (TIQ).
read_write	u_char	VCC-type: VCTE_RW = read+write; VCTE_RO = read-only; VCTE_WO = write-only.
aal_type	u_char	AAL-Type: AAL3/4, AAL5, Raw, CBR.
flags	u_char	Flags: VCTE_IP = VC carries IP traffic; VCTE_NOTRAILERS = no AAL5 trailers or CRCs are used; VCTE_NOSNAP = packets are not encapsulated with 802.2 LLC/SNAP.
ifunit_in	u_char	Logical network interface number (if_net) that is the endpoint. Only for VCs servicing IP traffic.
vcte	u_int	Local index (number), which was provided by the driver at the time the VC was created.

Relevant Structures

The `atmsioc_t` structure, as defined in the *sys/atm_user.h* file and the `atm_vcte_t` structure, as defined in the *sys/atm_b2h.h* file (which is included in the *sys/atm_user.h* file), are shown below for reference.

```
typedef struct atmsioc {
   void *ptr;
   u_int len;
} atmsioc_t;


typedef struct atm_vcte {
   u_int cell_hdr;
   u_int max_cs_pdu_size;
   u_short burst_size;
   u_char rate_queue_number;
   u_char avg_rate_divisor;
   u_char read_write;
   u_char aal_type;
   u_char flags;
   u_char ifunit_in;
   u_int vcte;
} atm_vcte_t;
```

Errors

Possible errors include:

EFAULT	An error occurred when the driver was copying the data.
EINVAL	The *len* specified in the argument is too small to contain the information being retrieved.
ENODEV	The board was not in the UP state.

ATMIOC_SETARP

The ATMIOC_SETARP *ioctl()* command puts one static mapping for a PVC into the IP-to-ATM address resolution table. This command is required for any VC that had the ATMPVCFL_IP flag set when the VC was created (with ATMIOC_CREATEPVC). The VC must already have been created with the ATMIOC_CREATEPVC call.

Usage

Use the following format:

```
ioctl (fd_atm, ATMIOC_SETARP, &arp);
```

where the file descriptor used for *fd_atm* is relatively unimportant (either the file descriptor from the ATMIOC_CREATEPVC or an IP socket descriptor can be used), and *arp* is a `struct arpreq`.

Argument Values

The argument is a pointer to an `arpreq` structure, set up as explained in Table 2-12.

Table 2-12 Recommended Values for ATMIOC_SETARP's Argument

Field of arpreq_t	Recommended Value	Comments
arp_pa	AF_INET and IP address	Within *sa_data* field, set the protocol family to AF_INET and provide the IP address of remote system.
arp_ha	`atm_laddr_t` structure	The local "hardware" address for the PVC. See Table 2-3 for complete details.
arp_flags	none	

Success or Failure

If successful, `ATMIOC_SETARP` returns zero.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Relevant Structures

The `arpreq` and `atm_laddr_t` structures are described in "Frequently Used Structures" on page 27.

Errors

Possible errors include:

EADDRINUSE	The address resolution table is already full. The current entry request was not added.
EAFNOSUPPORT	One of the *sa_family* fields within the `arpreq` indicated an address family that is not supported. Only AF_UNSPEC is supported for the `arp_ha` information, and only AF_INET is supported for the `arp_pa` area.
EFAULT	An error occurred as the driver was trying to copy the command's argument.
EINVAL	The *port* indicated in the `atm_laddr_t` is invalid, or the *vpi*/*vci* pair indicated in the `atm_laddr_t` already exists in the table, or the specified VC is not flagged for IP use.
ENODEV	The board was not in the UP or DOWN state.

Chapter 3

IRIS ATM *ioctl()* Commands for Switched VCs

This chapter summarizes the IRIS ATM Signalling application interface calls that support switched virtual channels (SVCs). The product includes an example of an application coded in C, */usr/lib/atm/examples/sigtest.c*, that uses this SVC API.

The services of the ATM subsystem are accessed through the IRIX character device interface *ioctl()* calls that specify ATM Signalling requests (commands). These calls are described alphabetically in the subsections that follow and are summarized in Table 3-1.

Table 3-1 Summary of SVC *ioctl()* Calls

Type of Operation	Command (or function)	Brd State	Description	More Info
Getting a link to the ATM-subsystem	*open()*	all	Opens a file descriptor for a cloned device. Must be held open as long as the SVC or the SVC request-queue is active.	page 7
Tearing down a VC	*close()*	all	Closes the file descriptor and causes the VC to be torn down and all resources released, including graceful rejection of any setup requests in the input queue.	page 8
Activating SVCs as the called party				
	`ATMIOC_REGISTER`	up / dn	Creates a request queue for incoming setup requests. Setup requests that match the specified traffic contract are accepted.	page 85
	`ATMIOC_LISTEN`	up / dn	Retrieves one setup request from the SVC's request queue.	page 77
	`ATMIOC_ACCEPT`	up / dn	Accepts a setup request. This results in a new SVC.	page 69
	`ATMIOC_REJECT`	up / dn	Refuses to accept a setup request.	page 89

Table 3-1 (continued) Summary of SVC *ioctl()* Calls

Type of Operation	Command (or function)	Brd State	Description	More Info
Activating SVCs as the calling party				
	`ATMIOC_SETUP`	up / dn	Requests a point-to-point SVC.	page 91
	`ATMIOC_MPSETUP`	up / dn	Requests a point-to-multipoint SVC and adds the first party.	page 80
Maintaining a multipoint SVC				
	`ATMIOC_ADDPARTY`	up / dn	Adds one more destination address to a point-to-multipoint SVC.	page 72
	`ATMIOC_DROPPARTY`	up / dn	Drops one destination address from a point-to-multipoint SVC.	page 75
Retrieving VC Information				
	`ATMIOC_GETVCTAB`	up	Retrieve information about all the open VCs.	page 45
Managing data				
	write()	up	Pinned down, 8-byte aligned buffer of any size. If necessary, ATM subsystem divides data into different packets for transmission.	page 9
	writev()	up	Gathers data from a number of buffers for transmission as one or more packets.	page 9
	read()	up	Retrieves incoming data.	page 8

Include Files for SVCs

The following files must be included in any program using the ATM-specific *ioctl()* calls:

- *"sys/atm.h"*
- *"sys/atm_user.h"*
- *"sys/if_atm.h"* (only for applications doing IP-over-ATM)

Overview

The IRIS ATM Signalling software makes it possible for applications to dynamically set up and tear down switched virtual channels (SVCs) in accordance with the ATM User-Network Interface (ATM UNI) standard. The software consists of the following components that work together to transparently provide support for SVCs:

- driver for the IRIS ATM network controller hardware
- signalling daemon (*atmsigd*) that implements the ATM User-Network Interface "signalling" standard for setting up and tearing down SVCs
- interim local management interface daemon (*atmilmid*) that implements the ATM User-Network Interface "local management" standard for exchange of status, configuration, and control information, including obtaining ATM addressing information from an adjacent switch

The IRIS ATM driver is the access point for applications using IRIS ATM services, as illustrated in Figure 3-1. Applications use the IRIS ATM application programming interface (API) to place their requests for creating and tearing down SVCs. The driver communicates these requests to the *atmsigd* and *atmilmid* modules, as appropriate. The *atmsigd* and *atmilmid* modules process requests in compliance with the ATM protocols as specified in the *ATM User-Network Interface Specification*.

The *atmsigd* module interfaces with other modules that handle the ATM signalling protocols and communication with the adjacent ATM switch. The ATM Signalling protocol stack consists of three protocols: Q.2931, QSAAL, and AAL5. The software can be configured so that multiple UNIs are created, each with possibly a different configuration.

The *atmilmid* module uses the simple network management protocol (SNMP, RFC 1157) to maintain a management information database (MIB) for each physical ATM connection and to communicate with adjacent ILMI programs. The objects within this MIB are those that are defined in the ILMI section of the ATM User-Network Interface standard. See Chapter 4 for the API calls that retrieve ILMI information.

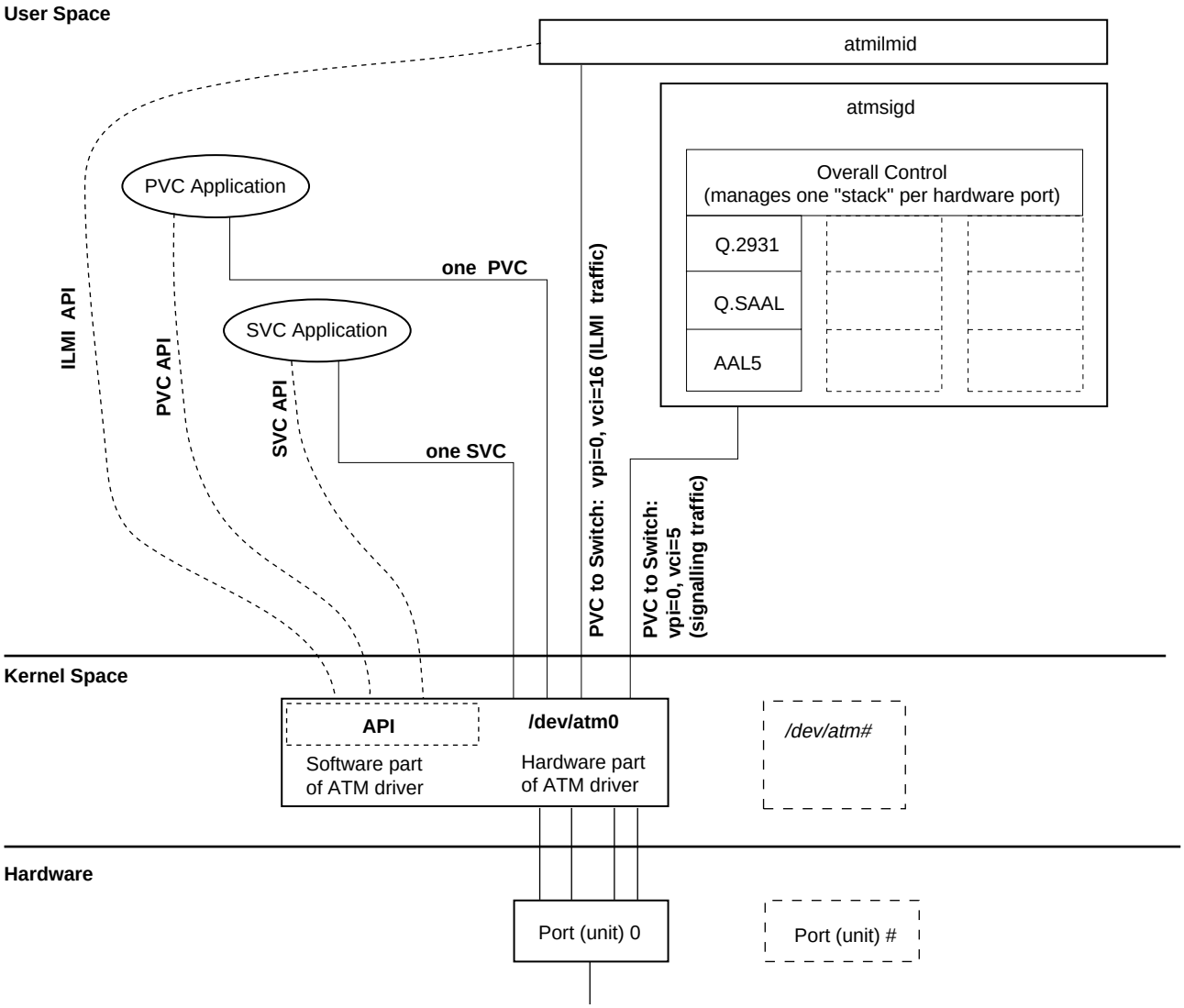


Figure 3-1 Overview of IRIS ATM Software Modules

Note: SVCs are created using `ATMIOC_SETUP` or `ATMIOC_REGISTER`, `ATMIOC_LISTEN`, and `ATMIOC_ACCEPT`. PVCs are created using `ATMIOC_CREATEPVC`.

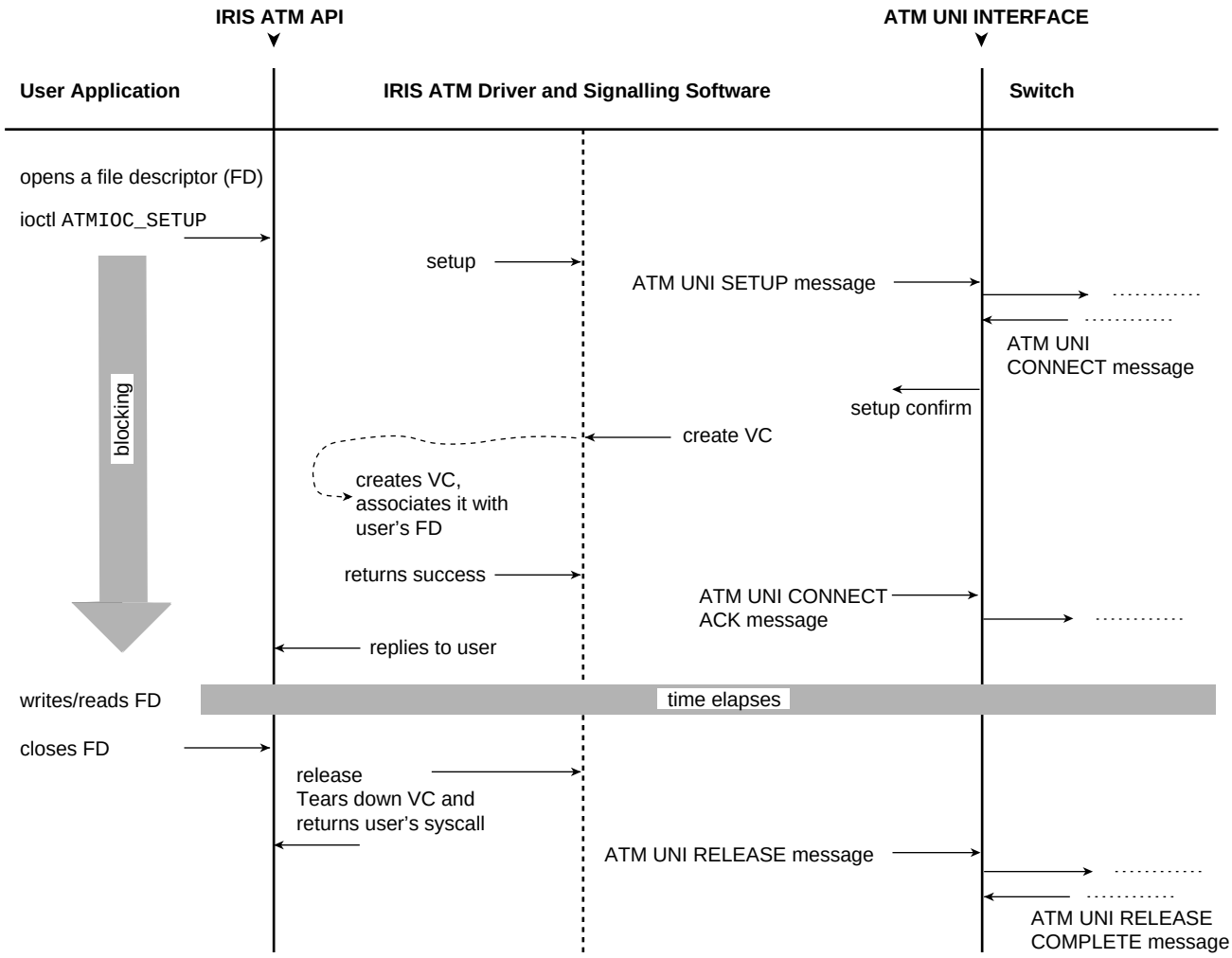


Figure 3-2 Successful Call Setup by Calling User

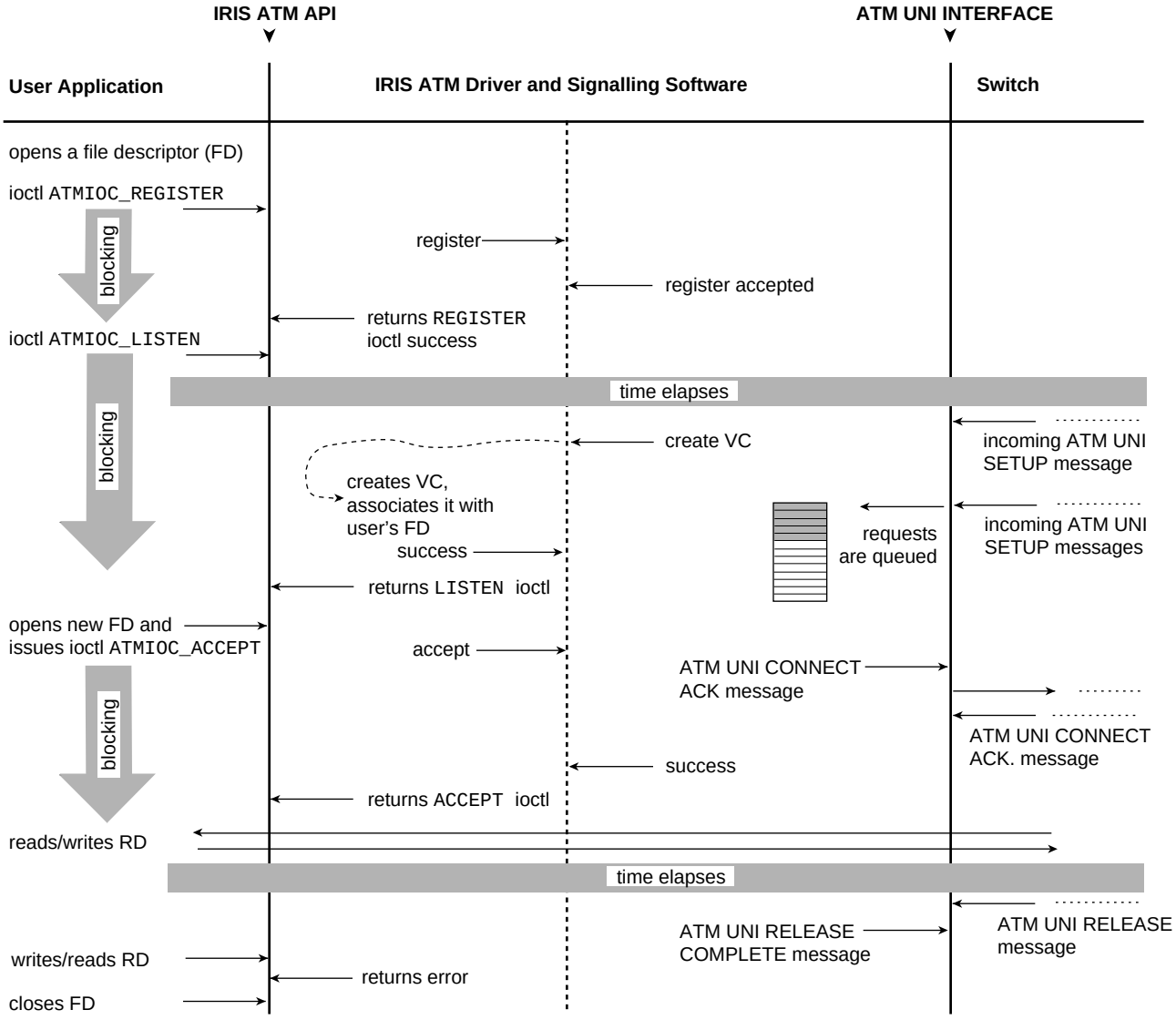


Figure 3-3 Successful Call Setup by Called User

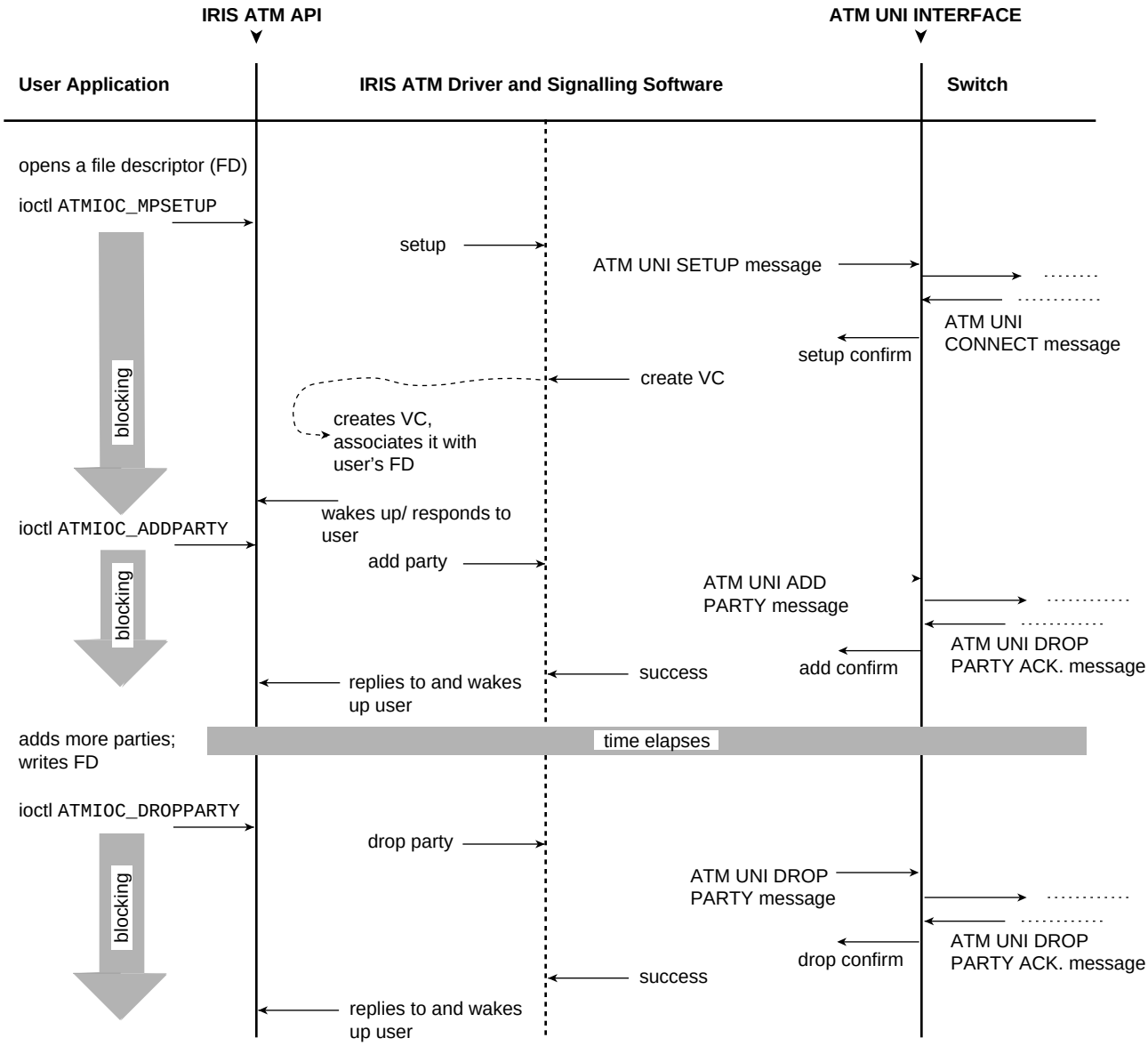


Figure 3-4 Successful Call Setup for Multicast SVC

Frequently Used Structures

The data structures described in this section are used as arguments for many of the ATM Signalling *ioctl()* calls.

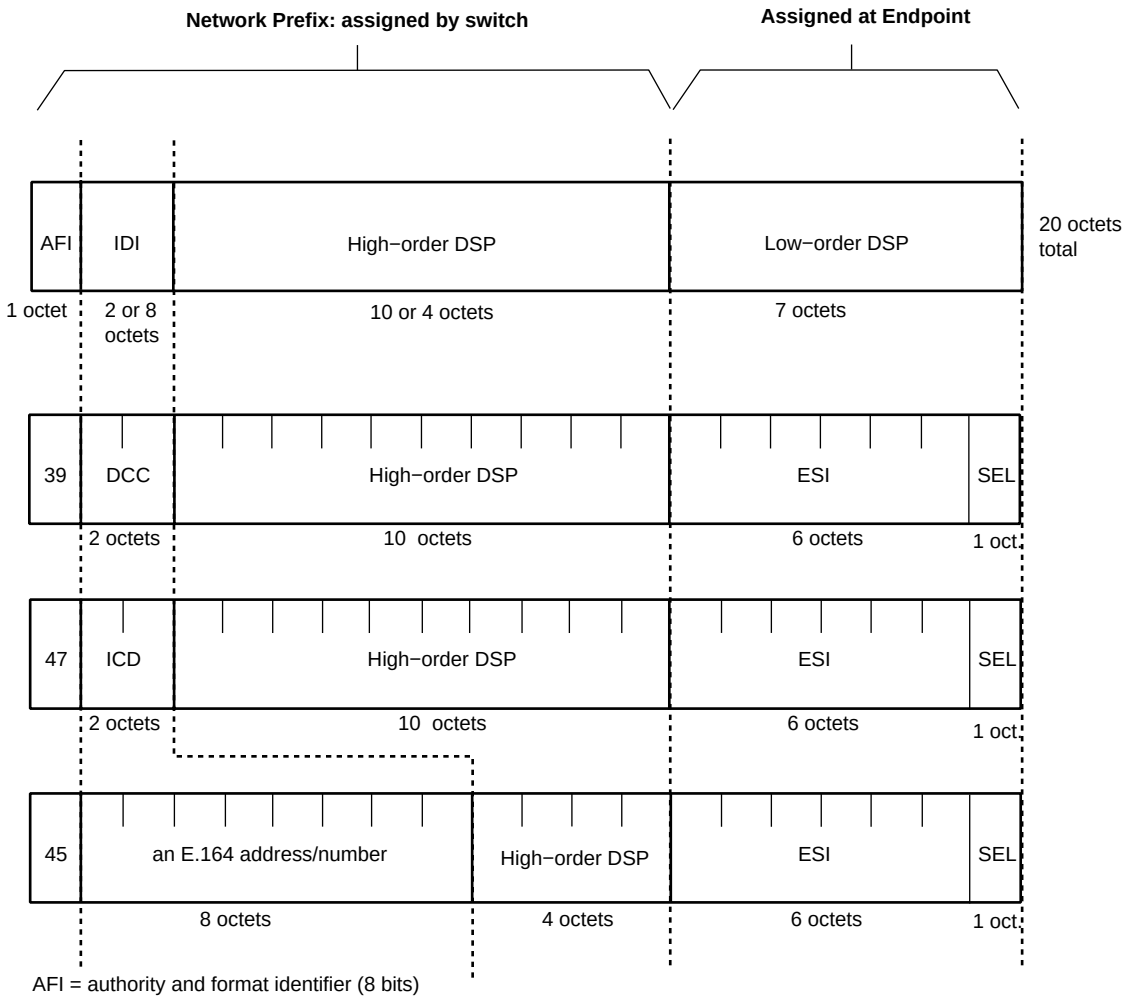
The atm_address_t Structure

The `atm_address_t` structure contains an ATM subsystem's network layer address, used for identifying users (the two endpoints) of a VC. Separate addresses are used for the called and the calling ATM subsystems. All fields of this address, except the ESI and SEL fields of the ATM NSAP, are assigned by an endpoint's switch.

Table 3-2 describes the `atm_address_t` structure. The first byte (*addrType* field) of the structure indicates the type of address: null, ATM NSAP, or native-E.164. The remaining field, *addr*, contains either a 20-byte ATM NSAP address (array of characters) or a variable-length E.164 address structure.

Table 3-2 The *atm_address_t* Structure

Field	Type	Values
addrType	char	NULLADDR_TYPE: no address is specified. NSAP_TYPE E164_TYPE
addr	union	One of the structures below:
nsap	array of char	atm_nsap_t[20]: an array of 20 numerals. Table 3-3 and Figure 3-5 provide more details.
e164	struct	atm_e164_t: variable length structure (as described in next 2 rows).
len	char	Number of valid digits in *addr[]* array.
addr[15]	array of char	Up to 15 digits encoded in IA5 characters. Appendix B describes the IA5 character set.



AFI = authority and format identifier (8 bits)
IDI = initial domain identifier (16 or 64 bits)
DSP = domain specific part (136 or 88 bits)

DCC = data country code (16 bits)
ICD = international code designator (16 bits)
ESI = end system identifier; can be a MAC address (48 bits)
IRIS ATM registers port's MAC addresss for this field.
SEL = end system selector; defined by local system, not by ATM standard (8 bits)
IRIS ATM software makes this field match the logical network interface number, so *atm1* uses SEL=0x01 and *atm47* uses SEL=0x2F.

Figure 3-5 ATM NSAP Format

Table 3-3 Contents for Fields of ATM NSAP

AFI Value[a]		IDI Content[1] (data size, field length)	DSP Length	Total Length of NSAP When in This Format
AFI_DCC	39	An ISO DCC value, which is a data country code from ISO 3166 (3-digit code, represented by 2 octets in which the unused least-significant 4 bits are set to ones).	17 octets	20 octets
AFI_E164	45	An E.164 address / number (up to 15 digits, represented by 8 octets in which the least significant four bits are ones, and any unused most-significant bits are set to zeros)	11 octets	20 octets
AFI_ICD	47	An ISO ICD value, which is an international code designator from ISO 6523 (4-digit code, represented by 2 octets)	17 octets	20 octets

a. Encoded in binary-coded decimal (BCD) format, where each four bits encodes one decimal numeral. For example, 0001 0010 (binary) represents 12 decimal. Binary values 0xA to 0xF are invalid for BCD encoding.

From the *sys/atm_user.h* file:

```
typedef struct atm_address {
#define NULLADDR_TYPE 0 /* No address specified */
#define NSAP_TYPE 0x02
#define E164_TYPE 0x11
    char addrType; /* one of the above types */
    union {
       nsap_address_t nsap;
       e164_address_t e164;
    } addr;
} atm_address_t;
```

```
#define AFI_DCC 0x39
#define AFI_ICD 0x47
#define AFI_E164 0x45

typedef char nsap_address_t[20];

typedef struct e164_address {
    unsigned char len;
    char addr[15];
} e164_address_t;
```

The cellrate_t Structure

The `cellrate_t` structure is used to specify an SVC's transmission rate and other traffic contract parameters. The user selects one of the cellrate types listed in Table 3-4, and specifies that selection in the first byte of the `cellrate_t` structure, described in Table 3-5. The format for the remaining portions of the `cellrate_t` structure depends on the content of the *cellrate_type* field. The various formats are described in Table 3-5. The specified peak cellrate must match one of the rates on the board's transmission rate queues. See "Characteristics of the ATM-OC3c Hardware" in Chapter 1 for a description of the transmission rate queues and how they are configured.

Table 3-4 Values for Cellrate Type

Value for *cellrate_type* Field	Description
CRT_NULL	Zero bandwidth.
CRT_PEAK_AGG	Aggregate peak cellrate for CLP0+1.
CRT_PSB_AGG	Aggregate peak cellrate, sustainable cellrate, and max burst size for CLP 0+1.
CRT_BEST_EFFORT	Peak cellrate for CLP0+1 with best-effort indication.
CRT_PEAK	Not supported in this release. Peak cellrates for CLP0 and CLP0+1.
CRT_PEAK_TAG	Not supported in this release. Same as above with tagging requested.

Table 3-4 (continued) Values for Cellrate Type

Value for *cellrate_type* Field	Description
CRT_PSB	Not supported in this release. Peak cellrate for CLP0+1, sustainable cellrate for CLP0, maximum burst size for CLP0.
CRT_PSB_TAG	Not supported in this release. Same as above with tagging requested.

Table 3-5 The *cellrate_t* Structure

Field	Type	Values
cellrate_type	char	From Table 3-4
rate	union	One of the formats (structures) below:
pcr_01	struct	Use with CRT_PEAK_AGG and CRT_BEST_EFFORT
pcr01	int	Peak cellrate for CLP 0+1, in cells per second. IRIS ATM subsystem assigns VC to a low-priority rate queue that is equal to or slower than the rate specified; if necessary, driver divides one of the configured rates to create a slower rate. If the specified rate is slower than the slowest configured low-priority rate queue divided by 64, then the rate cannot be supported.
psb_01	struct	Use with CRT_PSB_AGG.
pcr01	int	Peak cellrate for CLP 0+1, in cells per second. If all high-priority queues are used, this must match one of the configured rates.
scr01	int	Sustainable cellrate for CLP 0+1, in cells per second. Sustainable CR for CLP 0+1. PCR divided by SCR must be equal to or less than 64.
mbs01	int	Maximum burst size for CLP 0+1, in cells per burst. Valid values are multiples of 32 between 1 and 2048, inclusive. Zero is invalid.

From the *sys/atm_user.h* file:

```
typedef struct {
    char cellrate_type; /* a value from Table 3-4 */

    union {
       /* for cellrate_type = CRT_PEAK, CRT_PEAK_TAG */
       struct {
          int pcr0;
          int pcr01;
       } pcr_0_01;

       /* for cellrate_type = CRT_PEAK_AGG, CRT_BEST_EFFORT */
       struct {
          int pcr01;
       } pcr_01;

       /* for cellrate_type = CRT_PSB, CRT_PSB_TAG */
       struct {
          int pcr01;
          int scr0;
          int mbs0;
        } psb_0_01;

       /* for cellrate_type = CRT_PSB_AGG */
       struct {
          int pcr01;
          int scr01;
          int mbs01;
       } psb_01;

    } rate;
} cellrate_t;
```

The reject_reason_t Structure

Many of the *ioctl()* SVC commands provide causal information returned from the ATM network when a signalling message fails or is rejected. The structure used for this information is `reject_reason_t`, summarized in Table 3-6.

Table 3-6 The *reject_reason_t* Structure

Field	Type	Values
location	char	Identifies where along the VCC the failure or rejection occurred. Table 3-7 lists the values for this field.
cause	char	Describes the reason for the failure. Appendix C lists the values for this field.
diags[4]	array of char	Reserved for future use. Does not contain valid data.

Table 3-7 Values for Location Field In *reject_reason_t*

Text	Value for *location* Field
User	0x00
Private network serving the local user	0x01
Public network serving the local user	0x02
Transit network	0x03
Public network serving the remote user	0x04
Private network serving the remote user	0x05
International network	0x07
Network beyond interworking point	0x0A

From the *sys/atm_user.h* file:

```
typedef struct {
    char cause;/* value from Table C-1 or Table C-2 */
    char location;/* value from Table 3-7 */
    char diags[4];/* reserved for future use */
} reject_reason_t;
```

The QOS Variables

The one-byte quality of service variables (*fwdQOS* and *bwdQOS*) are used in a number of ATM Signalling commands to specify the forward and backward ATM service classes. Table 3-8 summarizes the valid values.

Table 3-8 Values for *QOS* Variables

Text	Value for QOS Variable	Description
QOS_CLASS_0	0	Use with best-effort traffic.
QOS_CLASS_1	1	Use with constant bit rate (CBR).
QOS_CLASS_2	2	Use with variable bit rate (VBR).
QOS_CLASS_3	3	Use for connection-oriented data.
QOS_CLASS_4	4	Use for connectionless data.

The BLLI Variable

The *blli* variable is used in a number of ATM Signalling commands to specify or communicate the ATM UNI broadband low layer information (BLLI) for a VCC. Calling parties can specify one to three BLLI options in their setup requests; after the request succeeds the single negotiated BLLI option is returned in the first element of the array. Called parties register for one option. Each BLLI value can be registered (with `ATMIOC_REGISTER`) by only one process at a time. (This does not mean one VC, since by forking, the registered process can support multiple VCs, as explained in the section describing the `ATMIOC_ACCEPT` command.) Table 3-9 summarizes the supported BLLI values.

When the BLLI_ANY value is specified in an ATMIOC_REGISTER call, any incoming BLLI value from the other party is accepted (including null BLLI). Use of all other values requires that the other party's specified BLLI selection match exactly; if there is no match, the IRIS ATM software rejects the connection request and does not place it on reception queue.

Table 3-9 Values for *BLLI* Variable

Text	Value for blli Variable	Description
BLLI_NULL	0	Null low layers. When used with ATMIOC_SETUP, always results in a negotiated BLLI of null. When used with ATMIOC_REGISTER, matches only to an incoming null BLLI.
BLLI_ANY	1	Any BLLI. Not valid for ATMIOC_SETUP. With ATMIOC_REGISTER, matches any BLLI, including null, on incoming setup requests.
BLLI_LLC2	2	Level 2 LLI = LLC. Whenever IP-over-ATM is enabled, this BLLI is registered (occupied) by the IP stack (the input queues for logical IP network interfaces), so other processes cannot receive on it. Additional ATMIOC_REGISTERs fail.
BLLI_LE_C	3	LAN Emulation control
BLLI_LE_ENET	4	LAN Emulation 802.3 data
BLLI_LE_ENET_MC	5	LAN Emulation 802.3 multicast
BLLI_LE_TR	6	LAN Emulation 802.5 data
BLLI_LE_TR_MC	7	LAN Emulation 802.5 multicast

The bearerClass Variable

The one-byte bearerClass variable is used in a number of ATM Signalling commands to specify the broadband bearer (also called transport or network) capability. Table 3-10 summarizes the valid values. See ATM UNI 3.1, Appendix F, for usage guidelines.

Table 3-10 Values for *bearerClass* Variables

Text	Value for bearerClass Variable	Description
BCOB_A	1	For use with non-ATM endpoints. Intermediate network nodes may map the data to another format.
BCOB_C	2	For use with non-ATM endpoints. Intermediate network nodes may map the data to another format.
BCOB_X_UNSPEC	3	Use for best-effort ATM traffic.
BCOB_X_CBR	4	Use for constant bit rate (CBR) ATM traffic.
BCOB_X_VBR	5	Use for variable bit rate (VBR) ATM traffic.

The MaxCSDU Variables

CSDU is a shortened version of CPCS-SDU, which stands for common-part convergence sublayer service data unit. The two-byte MaxCSDU integer value specifies the maximum size for the data units (packets) at the convergence sublayer of the AAL layer. This variable is subject to negotiation during connection setup, so the MaxCSDU sizes that are actually used are not necessarily those requested with the SETUP, MPSETUP, or REGISTER command.

Valid values range from 8 to 0x2FF8, and must be divisible by 8.

Separate MaxCSDU sizes are specified for the forward and the back channels of a VC. The *fwdMaxCSDU* size specifies a maximum packet size for the forward channel (that is, the channel on which the calling party transmits and the called party receives). The *bwdMaxCSDU* size specifies a maximum packet size for the back channel (that is, the channel on which the calling party receives and the called party transmits).

Note: "Forward" and "back" are always labeled from the calling party's viewpoint.

SVC Code Sample

An extensive sample of ATM-over-SVC code is provided in the file */usr/lib/atm/examples/sigtest.c*.

SVC Commands

This section describes each ATM SVC *ioctl()* command in detail. The commands are organized alphabetically.

Note: In these descriptions, *forward* refers to the channel carrying data from the calling party to the called party, while *backward* refers to the channel carrying data (in the opposite direction) from the called party to the calling party.

ATMIOC_ACCEPT

The ATMIOC_ACCEPT *ioctl()* command accepts a connection setup request that has already been retrieved by an ATMIOC_LISTEN. The file descriptor used in this call must be a **new** file descriptor for the same device used in the ATMIOC_REGISTER call. The application must block until the ATM software replies, which it does when an ATM UNI CONNECT ACKNOWLEDGE message returns from the calling party. The request is not removed from the queue until the call setup has completed (either by creating the SVC or by acknowledging a rejection). While waiting for the CONNECT ACKNOWLEDGE, the program that made the *ioctl()* call is put to sleep.

Invoking this *ioctl()* causes the ATM Signalling software to generate an ATM UNI CONNECT message. (An ATMIOC_LISTEN *ioctl()* must have completed successfully before the ATMIOC_ACCEPT can be invoked.) If the application wants to open multiple SVCs simultaneously for the associated traffic contract, it forks the new file descriptor (*new_fd_atm*) as soon as the ATMIOC_ACCEPT returns. At that point, the application can retrieve (do an ATMIOC_LISTEN) and accept (ATMIOC_ACCEPT) the next item on the queue. The application can receive (*read()*) data from all its open SVCs.

When the application wants to close a receiving SVC (accept no more requests), it simply closes the file descriptor. If one or more child processes have been forked, and they are still running, they must be killed or must also close their file descriptors. When the original file descriptor is closed, the ATM Signalling software generates an ATM UNI RELEASE message to the calling party.

Usage

Use the following format:

```
open (new_fd_atm, O_RDWR);
ioctl (new_fd_atm, ATMIOC_ACCEPT, &accept);
<wait for return, proceed as described in the next paragraph>
```

where *new_fd_atm* is a new read-write file descriptor for the same ATM device used in the ATMIOC_REGISTER call, and *accept* is an atm_accept_t structure.

Once the ATMIOC_ACCEPT returns, one of the following actions must be taken:

- If it is desirable to continue accepting other calls on this SVC (specifically its BLLI value), the process should fork, then the parent process should close its copy of the *new_fd_atm* that was used in the ATMIOC_ACCEPT. The parent process goes back to blocking on the ATMIOC_LISTEN call and processing new connection requests as they appear on the SVC's queue. The child process should close its copy of the ATMIOC_LISTEN's file descriptor and use the open connection until it is finished, at which time it simply closes its file descriptor.
- If this is the only call for this SVC, the process should close the file descriptor from the ATMIOC_LISTEN so that no more incoming calls are enqueued. This releases the BLLI value associated with that SVC for registration by another process. The process can then proceed to *read()* and *write()* the *new_fd_atm*.

Argument Values

The atm_accept_t structure should be prepared as described in Table 3-11.

Table 3-11 Recommended Values for ATMIOC_ACCEPT's Argument

Field in atm_accept_t	Type	Values
userHandle	int	The out value from the ATMIOC_LISTEN.
callHandle	int	The out value from the ATMIOC_LISTEN.

Success or Failure

If successful, ATMIOC_ACCEPT returns zero.

On failure, the *ioctl()* returns -1 with an error stored in errno. See the "Errors" heading for descriptions of individual errors.

Relevant Structures

From the *sys/atm_user.h* file:

```
typedef struct {
    int userHandle;
    int callHandle;
} atm_accept_t;
```

Errors

Possible errors include:

EINTR	While waiting for the accept call to complete from over the network, the *ioctl()* was interrupted unexpectedly.
EINVAL	The file descriptor was already bound (for example, with `ATMIOC_CREATEPVC`, `ATMIOC_SETUP`, `ATMIOC_MPSETUP`, or `ATMIOC_ACCEPT`). Or, the *userHandle* or *callHandle* was invalid or belonged to a different application. Or, the supplied *userHandle* did not identify a registered queue. Or, the ATM software discovered that the queue was empty.
ENOTCONN	The connection request is no longer valid. It has timed out or, has been released by the calling party.
EFAULT	An error occurred when the ATM software attempted to read the call's argument.
ENOSPC	The driver was not able to allocate a *userHandle* to the new file descriptor for the SVC.
ENODEV	The board was not in the UP or DOWN state. Or, the port was not operational.

ATMIOC_ADDPARTY

The `ATMIOC_ADDPARTY` *ioctl()* is invoked by a calling party to cause the ATM Signalling software to add another party to an already existing point-to-multipoint connection. The ATM Signalling software issues an ATM UNI ADDPARTY message. No backward channel is created for this SVC.

Usage

Use the following format:

```
ioctl (mp_fd_atm, ATMIOC_ADDPARTY, &addparty);
```

where *mp_fd_atm* is the same file descriptor used in the `ATMIOC_MPSETUP` call and *addparty* is an `atm_addparty_t` structure.

Argument Values

The `atm_addparty_t` structure should be prepared as described in Table 3-12.

Table 3-12 Recommended Values for ATMIOC_ADDPARTY's Argument

Field in atm_addparty_t	Type	Values
addparams	struct	An `addpartyparams_t` structure as described below:
calledNumber	struct	See "The atm_address_t Structure" on page 58.

Table 3-12 (continued) Recommended Values for ATMIOC_ADDPARTY's

Field in atm_addparty_t	Type	Values
	int	A locally unique tag, supplied by the program making this call. The handle is for identifying each party on an existing multipoint connection or connection request. User is responsible for ensuring that all its active tags are unique within its own "world." This value is not used in any meaningful way by the ATM Signalling software.
reject	struct *Upon failure =out value*	See "The reject_reason_t Structure" on page 64. Out value: if the add request fails to create an SVC, this structure contains the reason. A zero indicates that the failure occurred in the driver (before contacting the ATM Signalling daemon). A non-zero value indicates that the failure or rejection occurred at the called endpoint or at an intermediate system. The *cause* field identifies the cause for the failure as described in Appendix C.

Success or Failure

If successful, `ATMIOC_ADDPARTY` returns zero.

When a failure occurs within the driver, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors. When the error occurs within the driver, the *reject* field is zero. When a failure is due to a negative response from the network, the *ioctl()* wakes the sleeping program and returns -1 with an EIO error stored in `errno`. The *reject* out value should be read.

Out Values

When the *ioctl()* fails to create a VCC for the party, the out value in the reject field of the argument contains one of the causes described in Appendix C. A *reject* field of zero indicates that the *ioctl()* failed within the driver (not due to a negative response from the network).

Relevant Structures

The `atm_address_t` and `reject_reason_t` structures are described in "Frequently Used Structures" on page 58.

From the *sys/atm_user.h* file:

```
typedef struct {
    addpartyparams_t addparams;
    reject_reason_t reject;
} atm_addparty_t;

typedef struct {
    atm_address_t calledNumber;
    int partyHandle;
} addpartyparams_t;
```

Errors

Possible errors include:

EFAULT	An error occurred when the ATM software attempted to read the call's argument.
EINVAL	The SVC associated with the file descriptor is not connected or is not a multipoint connection (for example, the `ATMIOC_MPSETUP` has not been called or did not succeed).
EIO	The add party call was rejected by the network (an intermediate system) or by the called party. The reasons have been written into the *reject* field of the argument (which is a `reject_reason_t` structure). See "The reject_reason_t Structure" on page 64 and Appendix C.
ENODEV	The board was not in the UP or DOWN state. Or, the port was not operational.

ATMIOC_DROPPARTY

The ATMIOC_DROPPARTY *ioctl()* is invoked by a calling party to cause the ATM Signalling software to drop a called party from an existing point-to-multipoint connection. This *ioctl()* causes the ATM Signalling software to issue an ATM UNI DROPPARTY message.

Usage

Use the following format:

```
ioctl (mp_fd_atm, ATMIOC_DROPPARTY, &dropparty);
```

where *mp_fd_atm* is the same file descriptor used in the ATMIOC_MPSETUP or ATMIOC_ADDPARTY call that was used to add the party, and *dropparty* is an atm_dropparty_t structure.

Argument Values

The atm_dropparty_t structure should be prepared as described in Table 3-13.

Table 3-13 Recommended Values for ATMIOC_DROPPARTY's Argument

Field in atm_dropparty_t	Type	Values
partyHandle	int	The tag that was supplied by the program when it added this party to the SVC.

Success or Failure

If successful, `ATMIOC_DROPPARTY` returns zero.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Relevant Structures

From the *sys/atm_user.h* file:

```
typedef struct {
    int partyHandle;
} atm_dropparty_t;
```

Errors

Possible errors include:

EFAULT	An error occurred when the ATM software attempted to read the call's argument.
EINVAL	The SVC associated with the file descriptor is not connected or is not a multipoint connection (for example, the `ATMIOC_MPSETUP` has not been called or did not succeed).
ENODEV	The board was not in the UP or DOWN state. Or, the port was not operational.

ATMIOC_LISTEN

The ATMIOC_LISTEN *ioctl()* command retrieves connection setup requests from the input queue created by the ATMIOC_REGISTER call. The program calling this *ioctl()* must block until the ATM software replies, which it does whenever there is a request on the queue. If there are currently no requests waiting, the caller of the *ioctl()* is put to sleep and awakened when a request becomes available.

Each invocation of this *ioctl()* retrieves the topmost (longest awaiting) item on the queue. Each retrieval provides identification tags (handles) and the negotiated traffic contract for the SVC, which may be different from the parameters specified in the ATMIOC_REGISTER call. The request is not actually removed from the queue until the request has been completely processed by an ATMIOC_ACCEPT or ATMCIO_REJECT.

Note: An ATMIOC_REGISTER *ioctl()* must have completed successfully before ATMIOC_LISTEN can be invoked.

Usage

Use the following format:

```
ioctl (reg_fd_atm, ATMIOC_LISTEN, &listen);
```

where *reg_fd_atm* is the file descriptor used in the ATMIOC_REGISTER call, and *listen* is an atm_listen_t structure.

Argument Values

The argument is a pointer to an empty atm_listen_t structure (described in Table 3-14.

Success or Failure

If successful, ATMIOC_LISTEN returns zero. The out values should be read.

On failure, the *ioctl()* returns -1 with an error stored in errno. See the “Errors” heading for descriptions of individual errors.

Out Values

When the `ATMIOC_LISTEN` *ioctl()* completes successfully, each field of the call's argument contains information about one connection setup request from the input queue for the SVC associated with the file descriptor. The retrieved information describes the traffic contract for the connection, as described in Table 3-14.

Table 3-14 Values Retrieved by ATMIOC_LISTEN

Field in atm_listen_t	Type	Values
userHandle	int	Unique value provided by the ATM Signalling software to identify the application that invoked the `ATMIOC_LISTEN`. The value must be used in future *ioctl()* calls for this SVC.
callHandle	int	Unique value provided by the ATM Signalling software to identify this connection (SVC). The value must be used in future *ioctl()* calls for this SVC.
fwdMaxCSDU	u_short	The negotiated *fwdMaxCSDU* for the SVC. Value is always equal to or smaller than the value specified in the `ATMIOC_REGISTER`. See "The MaxCSDU Variables" on page 67.
bwdMaxCSDU	u_short	The negotiated *bwdMaxCSDU* for the SVC. Value is always equal to or smaller than the value specified in the `ATMIOC_REGISTER`. See "The MaxCSDU Variables" on page 67
blli	char	The *blli* value for the SVC. See "The BLLI Variable" on page 65.
caller	struct	The ATM address of the calling party as taken from the setup request. See "The atm_address_t Structure" on page 58.
xmitcellrate	struct	The cellrate for the SVC. See "The cellrate_t Structure" on page 61.

Relevant Structures

The `atm_listen_t` structure is described in Table 3-14. The `atm_address_t` and `cellrate_t` structures, and the `MaxCSDU` and `blli` variables are described in “Frequently Used Structures” on page 58.

From the *sys/atm_user.h* file:

```
typedef struct {
    int userHandle;
    int callHandle;
    u_short fwdMaxCSDU;
    u_short bwdMaxCSDU;
    char blli;
    atm_address_t caller;
    cellrate_t xmitcellrate;
} atm_listen_t;
```

Errors

Possible errors include:

EFAULT	An error occurred when the ATM software was accessing the call’s argument.
EINTR	While waiting for a request to appear on the queue, the call was interrupted unexpectedly.
ENODEV	The board was not in the UP or DOWN state. Or, the port was not operational.

ATMIOC_MPSETUP

The ATMIOC_MPSETUP *ioctl()* is invoked by a calling party to cause the ATM Signalling software to initiate an ATM UNI SETUP request message for the first party on a point-to-multipoint channel. No backward channel is created, so the device must be write-only. The application must block until the ATM driver replies, which it does when the SVC is either ready to use or has been refused. The driver puts the calling process to sleep until the call is complete or has been rejected.

When the remote endpoint accepts the connection request, the driver wakes the caller up and returns the negotiated traffic contract, which can be different (smaller) than what was specified in the call. Once open, the SVC is accessed by *write()*s to the specified file descriptor. The file descriptor opened for the ATM device (*fd_atm*) should be writable only.

To add additional parties and drop individual parties on this SVC, use the ATMIOC_ADDPARTY and ATMIOC_DROPPARTY commands.

To tear down (clear) this SVC, the application uses the ATMIOC_DROPPARTY command for each party that has been added to the SVC. This causes the ATM Signalling software to generate an ATM UNI DROPPARTY message for each party, until only one party remains, at which point a RELEASE message is generated. After the final party is dropped, the application can close the file descriptor.

Note: If the application closes the file descriptor without calling ATMIOC_DROPPARTY for each party, the software still gracefully releases and tears down the SVC.

Usage

Use the following format:

```
ioctl (fd_atm, ATMIOC_MPSETUP, &mpsetup);
```

where *fd_atm* is a write-only file descriptor for the desired ATM hardware, and *mpsetup* is an atm_mpsetup_t structure.

Argument Values

The `atm_mpsetup_t` structure should be prepared as described in Table 3-15.

Table 3-15 Recommended Values for ATMIOC_MPSETUP's Argument

Field in atm_mpsetup_t	Type	Values
mpcallparams		
calledNumber	struct	See "The atm_address_t Structure" on page 58.
callingNumber	struct	See "The atm_address_t Structure" on page 58.
fwdCSDU	u_short *Upon return =out value*	See "The MaxCSDU Variables" on page 67 Out value: when the *ioctl()* returns successfully, this field contains the negotiated value, which may be smaller than the original value.
fwdCellRate	struct	See "The cellrate_t Structure" on page 61.
fwdQOS	char	See "The QOS Variables" on page 65.
blliCount	char	0-3. Number of BLLI values in *blli[]* field. When this count is set to zero, the software specifies BLLI_NULL (which is the same as setting *blliCount*=1 and *blli[0]*=BLLI_NULL).
blli[3]	char *Upon return =out value*	See "The BLLI Variable" on page 65 Out value: *blli[0]* indicates the BLLI selected for this VCC which may be any of the original selections.
bearerClass	char	See "The bearerClass Variable" on page 67.
sscsType	char	Zero. Reserved for future use.
bhli	char	Zero. Reserved for future use.

Table 3-15 (continued) Recommended Values for ATMIOC_MPSETUP's

Field in atm_mpsetup_t	Type	Values
partyHandle	int	A locally unique tag supplied by the program making this call. The handle is for identifying each party on an existing multipoint connection or connection request. User is responsible for ensuring that all its active tags are unique within its own "world." This value is not used in any meaningful way by the ATM Signalling software.
reject	struct *Upon failure =out value*	See "The reject_reason_t Structure" on page 64. Out value: if the setup request fails to create an SVC, this structure contains the reason. A zero indicates that the failure occurred in the driver (before contacting the ATM Signalling daemon). A non-zero value indicates that the failure or rejection occurred at the called endpoint or at an intermediate system. The *cause* field identifies the cause for the failure as described in Appendix C.

Success or Failure

If successful, `ATMIOC_MPSETUP` returns zero. The out values should be read.

When a failure occurs within the driver (before it has placed the request onto the network), the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors. Under this condition, the *reject* field is zero. When a failure is due to a negative response from the network, the *ioctl()* wakes the sleeping program and returns -1 with an EIO error stored in `errno`. The *reject* out value contains information about the network's reason for the failure, so it should be read.

Out Values

When the *ioctl()* is successful, the calling party should check the values in the *fwdMaxCSDU* and *blli[0]* fields of the call's argument to discover the negotiated parameters. If the new values are acceptable, the calling party can start using the SVC. If the traffic contract is unacceptable (which really should not ever occur since the negotiated values are always lower), the application should close the file descriptor to close the connection. This action causes the IRIS ATM signalling subsystem to generate a RELEASE.

When the *ioctl()* fails to create an SVC, the out value in the reject field of the argument contains one of the causes described in Appendix C. A *reject* field of zero indicates that the *ioctl()* failed within the driver (not due to a negative response from the network).

Relevant Structures

The `atm_address_t`, `cellrate_t`, and `reject_reason_t` structures, and the `MaxCSDU`, `QOS`, `bearerClass`, and `blli` variables are described in "Frequently Used Structures" on page 58.

From the *sys/atm_user.h* file:

```
typedef struct {
    mpcallparams_t callparams;
    reject_reason_t reject;
} atm_mpsetup_t;

typedef struct {
    atm_address_t calledNumber;
    atm_address_t callingNumber;
    u_short fwdMaxCSDU;
    cellrate_t fwdCellRate;
    char fwdQOS;
    char blliCount;
    char blli[3];
    char bearerClass;
    char sscsType; /* reserved*/
    char bhli;    /* reserved*/
    int partyHandle;
} mpcallparams_t;
```

Errors

Possible errors include:

EFAULT	An error occurred when the ATM software attempted to read the call's argument.
EINTR	While waiting for a response from the switch, the driver was interrupted. The setup request cannot be completed. Try again.
EINVAL	The file descriptor was already bound (for example, with `ATMIOC_CREATEPVC`, `ATMIOC_SETUP`, `ATMIOC_MPSETUP`, or `ATMIOC_ACCEPT`). Or the access mode (read / write) was incorrect.
EIO	The setup call was rejected by the network (an intermediate system) or by the called party. The reasons have been written into the *reject* field of the argument (which is a `reject_reason_t` structure). See "The reject_reason_t Structure" on page 64 and Appendix C.
ENODEV	The board was not in the UP or DOWN state. Or, the port was not operational.
ENOSPC	The driver was not able to allocate a *userHandle* to the SVC.

ATMIOC_REGISTER

The `ATMIOC_REGISTER` *ioctl()* is invoked by an application to inform the IRIS ATM Signalling software of its presence and readiness as a called party. The file descriptor must be open for read-write access. The application must block until the ATM driver replies, which it does when the SVC is either ready to use or has been refused. The driver puts the calling process to sleep until the software has completed the SVC registration. When the ATM subsystem replies to this *ioctl()*, the application should immediately call `ATMIOC_LISTEN` to retrieve the first queued connection request.

Each `ATMIOC_REGISTER` call defines a traffic contract. For each registered traffic contract, the ATM subsystem maintains a queue of incoming connection (SVC) setup requests. The ATM Signalling software compares the registered traffic contracts to incoming connection setup request parameters. When the incoming values are higher than the registered values, the software negotiates down to the traffic contract. When the incoming values are equal to or smaller than the traffic contract, the software accepts the setup request and places it on the queue. This *ioctl()* fails if the specified traffic contract is currently registered.

When this *ioctl()* returns successfully, the ATM Signalling software has created a queue of the length specified by the application and has started queuing incoming connection (ATM UNI SETUP) requests. As long as the file descriptor remains open, the ATM Signalling software continues to queue requests.

When the application no longer wants to accept connection requests for this traffic contract, it simply closes the file descriptor. The ATM Signalling software generates ATM UNI RELEASE messages for the unretrieved requests remaining in the queue, and stops accepting requests for the associated traffic contract. Once the file descriptor is closed, the application cannot retrieve any more of the queued connection requests.

Usage

Use the following format:

```
ioctl (fd_atm, ATMIOC_REGISTER, &register);
```

where *fd_atm* is a read-write file descriptor for the desired ATM hardware, and *register* is an `atm_register_t` structure.

Argument Values

The `atm_register_t` structure should be prepared as described in Table 3-16.

Table 3-16 Recommended Values for ATMIOC_REGISTER's Argument

Field in atm_register_t	Type	Values
fwdMaxCSDU	u_short	Upper limit for size of CPCS-SDUs on calling party's forward channel. This value is compared to the requested value on incoming setup requests. A request is queued when the incoming value is equal to or smaller than this value. See "The MaxCSDU Variables" on page 67.
bwdMaxCSDU	u_short	Upper limit for size of CPCS-SDUs on calling party's backward channel. This value is compared to the requested value on incoming setup requests. A request is queued when the incoming value is equal to or smaller than this value. See "The MaxCSDU Variables" on page 67.
listenQlength	short	Maximum number of incoming setup requests that can be queued for this traffic contract.
blli	char	BLLI that is acceptable for these SVCs. When BLLI_ANY is specified, all incoming BLLI values are acceptable. See "The BLLI Variable" on page 65.

Table 3-16 (continued) Recommended Values for ATMIOC_REGISTER's

Field in atm_register_t	Type	Values
sscsType	char	Zero. Reserved for future use.
cause	int *Upon failure =out value*	Out value: if the register request fails to create an VCC, this field contains the reason. A zero indicates that the failure occurred in the driver (before contacting the ATM Signalling daemon). A non-zero value indicates that the failure or rejection occurred at the network. The value identifies the cause for the failure as described in Appendix C.

Success or Failure

If successful, `ATMIOC_REGISTER` returns zero.

When a failure occurs within the driver, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors. When the error occurs within the driver, the *cause* field is zero. When a failure is due to a negative response from the network, the *ioctl()* wakes the sleeping program and returns -1 with an EIO error stored in `errno`. The *cause* out value should be read.

Out Values

When the *ioctl()* fails to create a VCC for the party, the out value in the *cause* field of the argument contains one of the causes described in Appendix C. A *cause* field of zero indicates that the *ioctl()* failed within the driver (not due to a negative response from the network).

Relevant Structures

The `MaxCSDU` and `blli` variables are described in “Frequently Used Structures” on page 58.

From the *sys/atm_user.h* file:

```
typedef struct {
    u_short fwdMaxCSDU;
    u_short bwdMaxCSDU;
    short listenQlength; /* Nmbr of outstndng reqs to queue
*/
    char blli;
    char sscsType; /* reserved for future use */
    int cause; /* if ioctl fails with EIO, cause contains */
               /* a value from Appendix C*/
} atm_register_t;
```

Errors

Possible errors include:

EFAULT	An error occurred when the ATM software attempted to read the call’s argument.
EINVAL	The file descriptor was already bound (for example, with `ATMIOC_CREATEPVC`, `ATMIOC_SETUP`, `ATMIOC_MPSETUP`, or `ATMIOC_ACCEPT`). Or, the access mode (read / write) was incorrect. Or, the *listenQlength* value was invalid.
EIO	The registration request was rejected, and the reason has been written into the *cause* field of the argument. See Appendix C for a complete list of the possible values (causes).
ENOSPC	The driver was not able to allocate a *userHandle* to the SVC.
ENODEV	The board was not in the UP or DOWN state. Or, the port was not operational.

ATMIOC_REJECT

The ATMIOC_REJECT *ioctl()* refuses a connection setup request (that has already been retrieved by an ATMIOC_LISTEN) and to indicate the reason for the rejection. (An ATMIOC_LISTEN *ioctl()* must have completed successfully before ATMIOC_REJECT can be invoked.) ATMIOC_REJECT is invoked on the same file descriptor as the ATMIOC_LISTEN call. This *ioctl()* causes the ATM Signalling software to issue an ATM UNI RELEASE message.

The explanation for the rejection is given in the call's argument and is any of the ATM UNI cause codes, summarized in Appendix C.

The program calling this *ioctl()* can retrieve the next request from the queue immediately.

Note: This *ioctl()* cannot be used to release an existing SVC or to stop queuing SVC requests onto a registered queue. To stop accepting SVC setup requests, an application must close the file descriptor associated with the ATMIOC_REGISTER. To tear down (clear) an active SVC, the calling application closes the file descriptor associated with ATMIOC_SETUP.

Usage

Use the following format:

```
ioctl (listen_fd_atm, ATMIOC_REJECT, &reject);
```

where *listen_fd_atm* is the same file descriptor used in the ATMIOC_LISTEN call, and *reject* is an atm_reject_t structure.

Argument Values

The `atm_reject_t` structure should be prepared as described in Table 3-17.

Table 3-17 Recommended Values for ATMIOC_REJECT's Argument

Field in atm_reject_t	Type	Values
callHandle	int	This value must be the out value from the `ATMIOC_LISTEN` for this SVC.
cause	int	The reason the application is rejecting the setup request. Can be any of the ATM UNI causes listed in Table C-3.

Success or Failure

If successful, `ATMIOC_REJECT` returns zero.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Relevant Structures

From the *sys/atm_user.h* file:

```
typedef struct {
    int callHandle;
    int cause;
} atm_reject_t;
```

Errors

Possible errors include:

EFAULT	An error occurred when the ATM software attempted to read the call's argument.
EINVAL	The supplied *callHandle* did not identify a registered queue. Or, the ATM software discovered that the queue was empty.
ENODEV	The board was not in the UP or DOWN state. Or, the port was not operational.

ATMIOC_SETUP

The `ATMIOC_SETUP` *ioctl()* is invoked by a calling party to set up a point-to-point SVC with traffic contract parameters specified in the call's argument. The application must block until the ATM driver replies, which it does when the SVC is either ready to use or has been refused. The driver puts the calling process to sleep until the call is complete or has been rejected.

This *ioctl()* causes the ATM Signalling software to initiate an ATM UNI SETUP request message for creation of both a forward and a backward channel. When the remote endpoint accepts the connection request, the driver wakes the caller up and returns the negotiated traffic contract, which can be different (smaller) than what was specified in the call. Once open, the SVC is accessed by *read()*s from and *write()*s to the specified file descriptor. The file descriptor opened for the ATM device (*fd_atm*) should be readable and writable.[1]

To tear down (clear) this SVC, the application simply closes the file descriptor. This causes the ATM Signalling software to generate an ATM UNI RELEASE message.

Usage

Use the following format:

```
ioctl (fd_atm, ATMIOC_SETUP, &setup);
```

where *fd_atm* is a read-write file descriptor for the desired ATM hardware and *setup* is an `atm_setup_t` structure.

[1] It is not possible to create a unidirectional SVC.

Argument Values

The `atm_setup_t` structure should be prepared as described in Table 3-18.

Table 3-18 Recommended Values for ATMIOC_SETUP's Argument

Field in atm_setup_t	Recommended Value	Values
callparams	struct	
calledNumber	struct	See "The atm_address_t Structure" on page 58.
callingNumber	struct	See "The atm_address_t Structure" on page 58.
fwdMaxCSDU	u_short *Upon return =out value*	See "The MaxCSDU Variables" on page 67. Out value: when the *ioctl()* returns successfully, this field contains the negotiated value, which may be smaller than the original value.
bwdMaxCSDU	u_short *Upon return =out value*	See "The MaxCSDU Variables" on page 67. Out value: when the *ioctl()* returns successfully, this field contains the negotiated value, which may be smaller than the original value.
fwdCellRate	struct	See "The cellrate_t Structure" on page 61.
bwdCellRate	struct	See "The cellrate_t Structure" on page 61.
fwdQOS	char	See "The QOS Variables" on page 65.
bwdQOS	char	See "The QOS Variables" on page 65.
blliCount	char	0-3. Number of BLLI values in *blli[]* field. When this count is set to zero, the software specifies BLLI_NULL (which is the same as setting *blliCount*=1 and *blli[0]*=BLLI_NULL).

Table 3-18 (continued) Recommended Values for ATMIOC_SETUP's

Field in atm_setup_t	Recommended Value	Values
blli[3]	array of char *Upon return =out value*	See "The BLLI Variable" on page 65. Out value: when the *ioctl()* returns successfully, the first element (*blli[0]*) contains the negotiated value, which may be any one of the original values.
bearerClass	char	See "The bearerClass Variable" on page 67.
sscsType	char	Zero. Reserved for future use.
reject	struct *Upon failure = out value.*	See "The reject_reason_t Structure" on page 64. Out value: if the setup request fails to create an SVC, this structure contains the reason. A zero indicates that the failure occurred in the driver (before contacting the ATM Signalling daemon). A non-zero value indicates that the failure or rejection occurred at the called endpoint or at an intermediate system. The *cause* field identifies the cause for the failure as described in Appendix C.

Success or Failure

If successful, `ATMIOC_SETUP` returns zero. The out values should be read.

When a failure occurs within the driver (before it has placed the request onto the network), the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors. Under this condition, the *reject* field is zero. When a failure is due to a negative response from the network, the *ioctl()* wakes the sleeping program and returns -1 with an EIO error stored in `errno`. The *reject* out value contains information about the network's reason for the failure, so it should be read.

Out Values

The calling party should check the values in the *xxxMaxCSDU* and *blli[0]* fields of the call's argument to discover the negotiated parameters. If the new values are acceptable, the calling party can start using the SVC. If the traffic contract is unacceptable (which really should not ever occur since the negotiated values are always lower), the application should close the file descriptor to close the connection. This action causes the IRIS ATM signalling subsystem to generate a RELEASE.

When the *ioctl()* fails to create an SVC, the out value in the reject field of the argument contains one of the causes described in Appendix C. A *reject* field of zero indicates that the *ioctl()* failed within the driver (not due to a negative response from the network).

Relevant Structures

The `atm_address_t`, `cellrate_t`, and `reject_reason_t` structures, and the `MaxCSDU`, `QOS`, `bearerClass`, and `blli` variables are described in "Frequently Used Structures" on page 58.

From the *sys/atm_user.h* file:

```
typedef struct {
    ppcallparams_t callparams;
    reject_reason_t reject;
} atm_setup_t;

typedef struct {
    atm_address_t calledNumber;
    atm_address_t callingNumber;
    u_short fwdMaxCSDU, bwdMaxCSDU;
    cellrate_t fwdCellRate,bwdCellRate;
    char fwdQOS, bwdQOS;
    char blliCount;
    char blli[3];
    char bearerClass;
    char sscsType; /* reserved for future use */
    char bhli;     /* reserved for future use */
} ppcallparams_t;
```

Errors

Possible errors include:

EFAULT	An error occurred when the ATM software attempted to read the call's argument.
EINTR	While waiting for a response from the switch, the driver was interrupted. The setup request cannot be completed. Try again.
EINVAL	The file descriptor was already bound (for example, with `ATMIOC_CREATEPVC`, `ATMIOC_SETUP`, `ATMIOC_MPSETUP`, or `ATMIOC_ACCEPT`). Or, the access mode (read / write) was incorrect.
EIO	The setup call was rejected by the network (an intermediate system) or by the called party. The reasons have been written into the *reject* field of the argument (which is a `reject_reason_t` structure). See "The reject_reason_t Structure" on page 64 and Appendix C.
ENODEV	The board was not in the UP or DOWN state. Or, the port was not operational.
ENOSPC	The driver was not able to allocate a *userHandle* to the SVC.

Chapter 4

IRIS ATM *ioctl()* Commands for Use by ILMI Modules

This chapter summarizes the IRIS ATM application interface calls provided for use by interim local management interface (ILMI) modules. The calls allow an ILMI module to communicate with the IRIS ATM subsystem in retrieving and configuring UNI and MIB information. In most situations, these calls do not need to be used by customer-developed applications since the IRIS ATM ILMI software (*atmilmid*) does the tasks described in this chapter. However, these commands are provided for customers who want to use their own ILMI software for ATM network management.

Table 4-1 Summary of ILMI *ioctl()* Calls

Command	Brd State	Description	More Info
ATMIOC_GETMIBSTATS	up / dn	Retrieves data from an ATM subsystem for the ATM UNI MIB.	page 102
ATMIOC_GETPORTINFO	up / dn	Retrieves status and hardware specification information about the device.	page 104
ATMIOC_GETATMLAYERINFO	up / dn	Retrieves configuration information about the ATM layer of the device.	page 99
ATMIOC_GETVCCTABLEINFO	up / dn	Retrieves information about all the VCCs currently open on the device.	page 107
ATMIOC_GETATMADDR	up / dn	Retrieves the device's ATM address.	page 112
ATMIOC_SETATMADDR	up / dn	Sets (configures) the ATM address for the device.	page 116

Include Files for ILMI Programs

The following files must be included in any program using these ATM-specific *ioctl()* calls:

- *"sys/atm.h"*
- *"sys/atm_user.h"*

ILMI Commands

This section describes each ATM ILMI *ioctl()* command in detail. The commands are organized alphabetically.

ATMIOC_GETATMLAYERINFO

The `ATMIOC_GETATMLAYERINFO` *ioctl()* command is invoked by an ILMI application to retrieve information about the ATM layer for inclusion in an ATM management information database (MIB).

Usage

Use the following format:

`ioctl (`*fd_atm*`, ATMIOC_GETATMLAYERINFO, `*&layerinfo*`);`

where *layerinfo* is an `atm_layerinfo_t` structure.

Argument Values

The argument is a pointer to an empty `atm_layerinfo_t` structure.

Success or Failure

If successful, `ATMIOC_GETATMLAYERINFO` returns zero. The out values should be read.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Out Values

The retrieved values are copied to the structure pointed to by the call's argument, described in Table 4-2.

Table 4-2 Values Retrieved by ATMIOC_GETATMLAYERINFO

Field in atm_layerinfo_t	Type	Values
maxVPCs	int	0 to 0xFF (inclusive)
maxVCCs	int	0 to 0xFFFFFF (inclusive)
configuredVPCs	int	0 to 0xFF (inclusive)
configuredVCCs	int	0 to 0xFFFFFF (inclusive)
maxVPIbits	int	0 to 0x8 (inclusive)
maxVCIbits	int	0 to 0x20 (inclusive)
uniType	int	The type of UNI maintained for the port: 1 = PUBLIC_UNI 2 = PRIVATE_UNI 2

Relevant Structures

The `atm_layerinfo_t` structure is described Table 4-2 and included below as it is defined in the *sys/atm_user.h* file:

```
typedef struct {
    int maxVPCs;
    int maxVCCs;
    int configuredVPCs;
    int configuredVCCs;
    int maxVPIbits;
    int maxVCIbits;
    int uniType;
} atm_layerinfo_t;
```

Errors

Possible errors include:

EFAULT	An error occurred when the ATM software attempted to write the call's argument.
ENODEV	The board was not in the UP or DOWN state.

ATMIOC_GETMIBSTATS

The `ATMIOC_GETMIBSTATS` *ioctl()* command is invoked by an ILMI application to retrieve information about overall performance on the UNI for inclusion in an ATM management information database (MIB).

Usage

Use the following format:

```
ioctl (fd_atm, ATMIOC_GETMIBSTATS, &mibstats);
```

where *mibstats* is an `atm_getmibstats_t` structure.

Argument Values

The argument is a pointer to an empty `atm_getmibstats_t` structure.

Success or Failure

If successful, `ATMIOC_GETMIBSTATS` returns zero. The out values should be read.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Out Values

The retrieved values are copied to the structure pointed to by the call's argument, described in Table 4-3.

Table 4-3 Values Retrieved by ATMIOC_GETMIBSTATS

Field in atm_getmibstats_t	Type	Description
receivedCells	int	Total number of ATM cells received.
droppedReceivedCells	int	Total number of ATM incoming cells that were dropped due to errors or unknown VPI/VCI addresses.
cellsTransmitted	int	Total number of ATM cells transmitted.

Relevant Structures

The `atm_getmibstats_t` structure is described in Table 4-3 and included below as defined in the *sys/atm_user.h* file:

```
typedef struct {
    int receivedCells;
    int droppedReceivedCells;
    int cellsTransmitted;
} atm_getmibstats_t;
```

Errors

Possible errors include:

EFAULT — An error occurred when the ATM software attempted to write the call's argument.

ENODEV — The board was not in the UP or DOWN state.

ATMIOC_GETPORTINFO

The `ATMIOC_GETPORTINFO` *ioctl()* command is invoked by an ILMI application to retrieve information about the hardware for inclusion in an ATM management information database (MIB).

Usage

Use the following format:

```
ioctl (fd_atm, ATMIOC_GETPORTINFO, &portinfo);
```

where *portinfo* is an `atm_portinfo_t` structure.

Argument Values

The argument is a pointer to an empty `atm_portinfo_t` structure.

Success or Failure

If successful, `ATMIOC_GETPORTINFO` returns zero. The out values should be read.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Out Values

The retrieved values are copied to the structure pointed to by the call's argument, described in Table 4-4.

Table 4-4 Values Retrieved by ATMIOC_GETPORTINFO

Field in atm_portinfo_t	Type	Values
portOperStatus	int	The status of the port: 1 = OPSTATUS_OTHER 2 = OPSTATUS_INSERVICE 3 = OPSTATUS_OUTOFSERVICE 4 = OPSTATUS_LOOPBACK
portXmitType	int	The physical layer protocol: 1 = XMITTYPE_UNKNOWN 2 = XMITTYPE_SONETSTS3C 3 = XMITTYPE_DS3 4 = XMITTYPE_4B5B 5 = XMITTYPE_8B10B
portMediaType	int	The type of transport medium used on the port: 1 = MEDIATYPE_UNKNOWN 2 = MEDIATYPE_COAX 3 = MEDIATYPE_SINGLEMODE 4 = MEDIATYPE_MULTIMODE 5 = MEDIATYPE_SHIELDEDTP 6 = MEDIATYPE_UNSHIELDEDTP

Relevant Structures

The `atm_portinfo_t` structure is described in Table 4-4 and included below as it is defined in the *sys/atm_user.h* file:

```
typedef struct {
    int portOperStatus;
    int portXmitType;
    int portMediaType;
} atm_portinfo_t;
```

Errors

Possible errors include:

EFAULT	An error occurred when the ATM software attempted to write the call's argument.
ENODEV	The board was not in the UP or DOWN state.

ATMIOC_GETVCCTABLEINFO

The `ATMIOC_GETVCCTABLEINFO` *ioctl()* command is invoked by an ILMI module to retrieve information about each open virtual channel (VC). The retrieved listing includes permanent and switched VCs.

Usage

Use the following format:

`ioctl (`*fd_atm*`, ATMIOC_GETVCCTABLEINFO, &`*sioc*`);`

where *sioc* is an `atmsioc_t` structure.

Argument Values

The pointer to *sioc* identifies an instance of an `atmsioc_t` structure. The *sioc* should be set up as summarized in Table 4-5.

Table 4-5 Recommended Values for ATMIOC_GETVCCTABLEINFO's Argument

Field of atmsioc_t	Recommended Value	Comments
*ptr	=pointer to `atm_vcce_t[]`	Pointer to location for retrieved information.
	Upon return =out value	Out value: an array of `atm_vcce_t` structures
len	=sizeof(`atm_vcce_t[MAX_FWD_VCS+ MAX_RVS_VCS]`)	Maximum possible size of the table.
	Upon return =out value	Out value: length of retrieved table

Success or Failure

If successful, `ATMIOC_GETVCCTABLEINFO` returns zero. The out values should be read.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Out Values

The *len* field in the argument (*sioc*) is updated to contain the actual length of the retrieved data, as described in Table 4-5. The retrieved data are written at the location indicated by the *sioc* pointer as an array of `atm_vcce_t` structures. Each table entry is one structure, as described in Table 4-6.

Table 4-6 Values Retrieved by ATMIOC_GETVCCTABLEINFO

Field in atm_vcce_t	Type	Values
vpi	int	The VC's virtual path identifier.
vci	int	The VC's virtual channel identifier.
xmit_cellrate	struct cellrate_t	The VC's transmit cellrate. See Table 4-7.
recv_cellrate	struct cellrate_t	The VC's receive cellrate. See Table 4-7.
xmitQOS	int	The quality of service on the VC's transmit channel.
recvQOS	int	The quality of service on the VC's receive channel.

Table 4-7 Cellrate Values

Field	Type	Values
cellrate_type	char	From Table 3-4
rate	union	One of the structures below:
pcr_0_01	struct	
pcr0	int	Peak cellrate for CLP 0, in cells per second
pcr01	int	Peak cellrate for CLP 0+1, in cells per second
pcr_01	struct	
pcr01	int	Peak cellrate for CLP 0+1, in cells per second
psb_0_01	struct	
pcr01	int	Peak cellrate for CLP 0+1, in cells per second
scr0	int	Sustainable cellrate for CLP 0, in cells per second
mbs0	int	Max Burst Size for CLP 0, in cells per burst
psb_01	struct	
pcr01	int	Peak cellrate for CLP 0+1, in cells per second
scr01	int	Sustainable cellrate for CLP 0+1, in cells per second
mbs01	int	Max Burst Size for CLP 0+1, in cells per burst

Relevant Structures

The `atm_vcce_t` structure is described Table 4-6 and included below as defined in the *sys/atm_user.h* file. The `cellrate_t` structure is described in Table 4-7 and is also included below as it is defined in the *sys/atm_user.h* file.

```
typedef struct {
    int vpi;
    int vci;
    cellrate_t xmit_cellrate;
    cellrate_t recv_cellrate;
    int xmitQOS;
    int recvQOS;
} atm_vcce_t;
```

```
typedef struct {
    char cellrate_type;
    union {
       /* for cellrate_type = CRT_PEAK, CRT_PEAK_TAG */
       struct {
          int pcr0;
          int pcr01;
       } pcr_0_01;

    /* for cellrate_type = CRT_PEAK_AGG, CRT_BEST_EFFORT */
       struct {
          int pcr01;
       } pcr_01;

       /* for cellrate_type = CRT_PSB, CRT_PSB_TAG */
       struct {
          int pcr01;
          int scr0;
          int mbs0;
       } psb_0_01;

       /* for cellrate_type = CRT_PSB_AGG */
       struct {
          int pcr01;
          int scr01;
          int mbs01;
       } psb_01;

    } rate;
} cellrate_t;
```

Errors

Possible errors include:

EFAULT	An error occurred when the ATM software attempted to write the call's argument.
EINVAL	The argument's length is too small to accommodate the table. No data has been copied out.
ENODEV	The board was not in the UP or DOWN state.

ATMIOC_GETATMADDR

The `ATMIOC_GETATMADDR` *ioctl()* command is invoked by an ILMI module to retrieve the ATM address that is currently being used on the device (port).

Usage

Use the following format:

`ioctl (`*fd_atm*`, ATMIOC_GETATMADDR, &`*address*`);`

where *address* is an `atm_address_t` structure.

Argument Values

The argument is a pointer to an empty `atm_address_t` structure (described in Table 4-8).

Success or Failure

If successful, `ATMIOC_GETATMADDR` returns zero. The out values should be read.

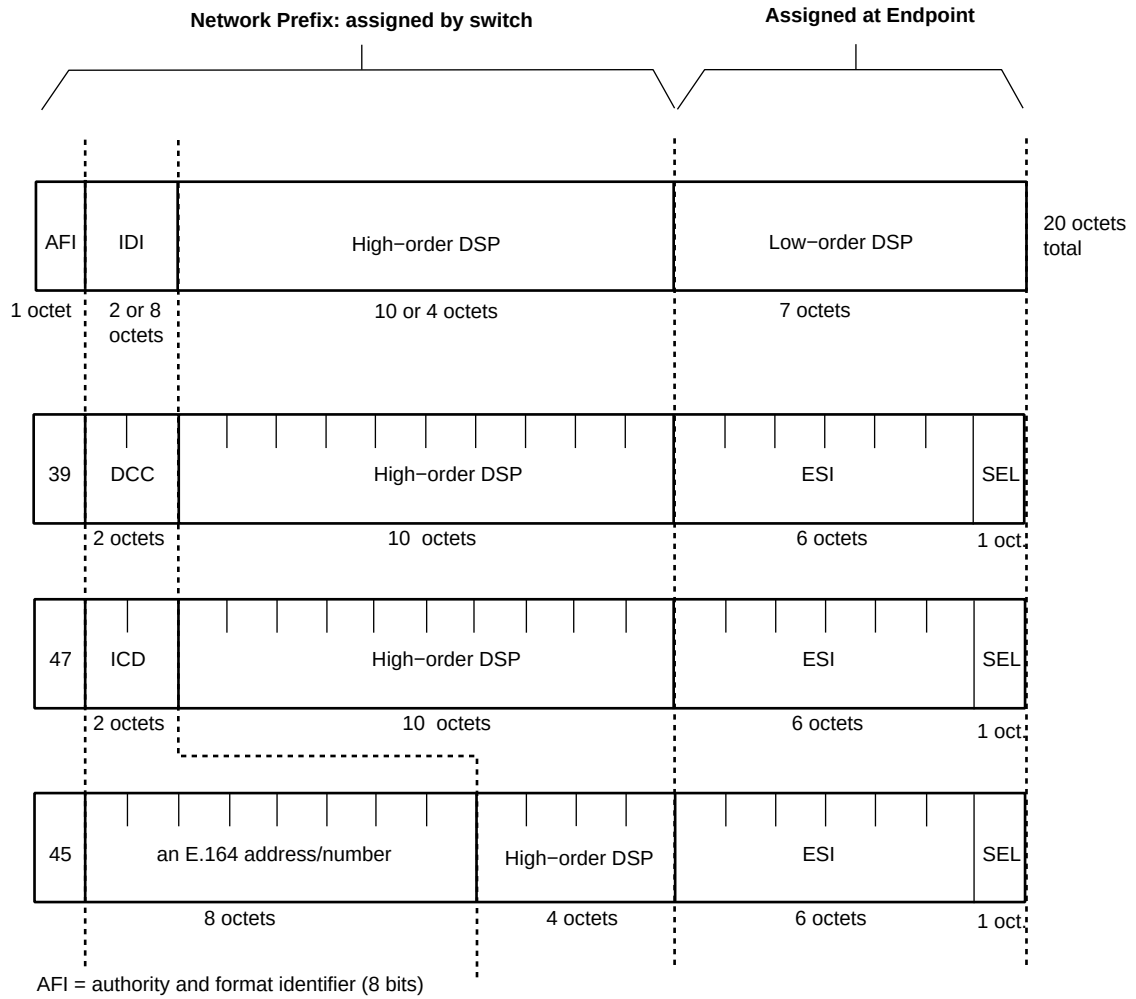
On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Out Values

The retrieved ATM address, described in Table 4-8 and Figure 4-1, is copied into the call's argument. The address can be either the ATM NSAP or native-E.164 format.

Table 4-8 Values Retrieved by ATMIOC_GETATMADDR

Field in atm_address_t	Type	Values
addrType	char	The type of ATM address: 0 = NULLADDR_TYPE 0x02 = NSAP_TYPE 0x11 = E164_TYPE
addr	union	
nsap	char nsap_address_t[20]	See Figure 4-1.
e164	e164_address_t	Up to 15 bytes. See definition in "Relevant Structures."



AFI = authority and format identifier (8 bits)
IDI = initial domain identifier (16 or 64 bits)
DSP = domain specific part (136 or 88 bits)

DCC = data country code (16 bits)
ICD = international code designator (16 bits)
ESI = end system identifier; can be a MAC address (48 bits)
IRIS ATM registers port's MAC addresss for this field.
SEL = end system selector; defined by local system, not by ATM standard (8 bits)
IRIS ATM software makes this field match the logical network interface number, so *atm1* uses SEL=0x01 and *atm47* uses SEL=0x2F.

Figure 4-1 ATM Address: NSAP Format

Relevant Structures

The `atm_address_t` structure is described below, as it is defined in the *sys/atm_user.h* file:

```
typedef struct atm_address {
    char addrType;
    union {
       nsap_address_t nsap;
       e164_address_t e164;
 } addr;
} atm_address_t;

typedef char nsap_address_t[20];

typedef struct e164_address {
    unsigned char len;
    char addr[15];
} e164_address_t;
```

Errors

Possible errors include:

EFAULT	An error occurred when the ATM software attempted to write the call's argument.
ENODEV	The board was not in the UP or DOWN state. Or, the port was not operational.

ATMIOC_SETATMADDR

The `ATMIOC_SETATMADDR` *ioctl()* command is invoked by an ILMI module to set the ATM address for the port. The program making this call must have superuser (root) access privileges.

Usage

Use the following format:

`ioctl (`*fd_atm*`, ATMIOC_SETATMADDR, &`*address*`);`

where *address* is an `atm_address_t` structure.

Argument Values

The `atm_address_t` structure should be prepared as described in Table 4-9.

Table 4-9 Recommended Values for ATMIOC_SETATMADDR's Argument

Field in atm_address_t	Type	Values
addrType	char	The type of ATM address: 0 = NULLADDR_TYPE 0x02 = NSAP_TYPE 0x11 = E164_TYPE
addr	union	
nsap	char nsap_address_t[20]	20 bytes as illustrated in Figure 4-1.
e164	struct e164_address_t	
len	u_char	Number of digits in `addr` array.
addr	char addr[15]	Up to 15 digits.

Success or Failure

If successful, `ATMIOC_SETATMADDR` returns zero.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Relevant Structures

The `atm_address_t` structure is described below, as it is defined in the *sys/atm_user.h* file:

```
typedef struct atm_address {
    char addrType;
    union {
       nsap_address_t nsap;
       e164_address_t e164;
 } addr;
} atm_address_t;

typedef char nsap_address_t[20];

typedef struct e164_address {
    unsigned char len;
    char addr[15];
} e164_address_t;
```

Errors

Possible errors include:

EFAULT	An error occurred when the ATM software attempted to read the call's argument.
ENODEV	The board was not in the UP or DOWN state. Or, the port was not operational.
EPERM	The program does not have superuser (root) access privileges.

Chapter 5

IRIS ATM *ioctl()* Commands for Communicating With the Hardware

This chapter summarizes the IRIS ATM application interface calls that communicate with IRIS ATM boards.

Table 5-1 Summary of ATM-OC3c *ioctl()* Calls

Type of Operation	Command (or function)	Usage	Brd State	Description	More Info
Retrieving board status and information					
	`ATMIOC_GETSTAT`		all	Retrieves current status information from ATM-OC3c board.	page 135
	`ATMIOC_GETIOSTAT`		up / dn	Retrieves driver-internal statistics.	page 127
	`ATMIOC_GETCONF`		up / dn	Reads configuration information from ATM-OC3c board.	page 124
	`ATMIOC_GETOPT`	root only	up / dn	Retrieves settings for board's operating modes / options.	page 131
	`ATMIOC_GETRATEQ`		up	Retrieves setting for one of the board's eight transmission rates.	page 132
	`ATMIOC_GETMACADDR`		up / dn	Retrieves the medium access control (MAC) address from ATM-OC3c board.	page 130
Configuring ATM-OC3c board					
	`ATMIOC_SETCONF`	root only	up / dn	Configures ATM-OC3c board.	page 147

Table 5-1 (continued) Summary of ATM-OC3c *ioctl()* Calls

Type of Operation	Command (or function)	Usage	Brd State	Description	More Info
	ATMIOC_SETOPT	root only	up / dn	Sets (configures) the board's operating modes / options: loopback and clock recovery.	page 150
	ATMIOC_SETRATEQ	root only	up	Sets the transmission rate on one of the eight queues	page 153
Controlling the ATM-OC3c board					
	ATMIOC_CONTROL	root only	all	Transitions board to different state: UP: to UP state INIT: to DOWN state RESET: to pre-init state	page 121

Include Files for Hardware Calls

The following file must be included in any program using the ATM-specific *ioctl()* calls for controlling the hardware:

- *"sys/atm_user.h"*

Hardware Commands

This section describes each ATM hardware control *ioctl()* command in detail. The commands are organized alphabetically.

ATMIOC_CONTROL

The `ATMIOC_CONTROL` *ioctl()* command changes the state of the ATM-OC3 board. This command is available only to the superuser.

Once powered on, the ATM-OC3 board has three states, as described below:

- Pre-initialized:
  The board is ready to be initialized. This state exists after each reset of the board. The only commands available in this state are `ATMIOC_CONTROL` with the INIT argument and `ATMIOC_GETSTAT`.
- Down:
  The board is initialized, alive, and ready to respond to the driver; however, the board is not receiving or transmitting over its network connection. In this state the board's memory can be configured and written.
- Up:
  The board is receiving and transmitting over its network connection.

Usage

Use the following format:

`ioctl (`*`fd_atm`*`, ATMIOC_CONTROL, `*`int`*`);`

where *int* is one of the values from Table 5-2.

Argument Values

The *int* argument's values are described in Table 5-2.

Table 5-2 Values for ATMIOC_CONTROL's Argument

int	Required State of Board	Description
ATM_CONTROL_RESET	Any	Allowed under all conditions. Shuts down board, throws away all in-progress data and host-to-board commands, and puts board into pre-initialized state. Wakes up processes that are awaiting completion of host-to-board commands and returns ENODEV.
ATM_CONTROL_INIT	Pre-init	Initializes board and brings it to DOWN state. Not allowed when there are open file descriptors for the device.
ATM_CONTROL_UP	Down	Brings board to UP state.

Success or Failure

If successful, `ATMIOC_CONTROL` returns zero.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Errors

Possible errors include:

EBUSY	When trying to INIT (initialize, bring to DOWN state) the board, the driver found that there are file descriptors open for this device. These must be closed before initializing the board.
EINVAL	When trying to INIT (initialize) or bring the board to the UP state, the driver found that the board was not in the required state.
EIO	When trying to INIT (initialize) the board, the driver could not successfully bring the board into the DOWN state.
EPERM	The calling application does not have superuser access privileges.
ETIME	When trying to bring the board to the UP state, the driver's call to the board timed out.

ATMIOC_GETCONF

The `ATMIOC_GETCONF` *ioctl()* command retrieves the ATM-OC3c board's current configuration.

Usage

Use the following format:

`ioctl (`*fd_atm*`, ATMIOC_GETCONF, &`*conf*`);`

where *conf* is an `atm_conf_t` structure.

Argument Values

The argument is a pointer to an `atm_conf_t` structure, described in Table 5-3.

Success or Failure

If successful, `ATMIOC_GETCONF` returns zero. The out values should be read.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Out Values

The retrieved configuration values are written into the argument as described in Table 5-3.

Table 5-3 Values Retrieved by ATMIOC_GETCONF

Field	Default Value	Comments
sign	ATM_MAGIC	ATM-OC3c board's signature.
vers	varies	ATM-OC3c board's / FLASH EPROMs version
flags	0x0608	Hardware and firmware capabilities. See Table 5-4.

Table 5-3 (continued) Values Retrieved by ATMIOC_GETCONF

Field	Default Value	Comments
xtype	2	Transmission type: 1 =XT_UNKNOWN 2 =XT_STS3C, SONET STS-3c PHY at 155.52 Mbps 3 =XT_DS3=3, DS3 PHY at 44.736 Mbps 4 =XT_4B5B=4, 4B/5B encoding PHY at 100 Mbps 5 =XT_8B10B, 8B/10B encoding PHY at 155.52 Mbps
mtype	4	Media type: 1 =MT_UNKNOWN 2 =MT_COAX, coax cable 3 =MT_SMF, single mode fiber 4 =MT_MMF, multi-mode fiber 5 =MT_STP, shielded twisted pair 6 =MT_UTP, unshielded twisted pair
maxvpibits	8	Maximum number of bits that can be used for a VPI. Range of possible values is 0 to 8.
maxvcibits	16	Maximum number of bits that can be used by a VCI. Range of possible values is 0 to 16.
hi_pri_qs	4	Number of transmission rate queues that are treated as high-priority queues.
lo_pri_qs	4	Number of transmission rate queues that are treated as low-priority queues.
xmt_large_size	12K	Size (in bytes) of large-sized transmit buffers.
xmt_large_bufs	78	Number of large-sized transmit buffers.
xmt_small_size	2K	Size (in bytes) of small-sized transmit buffers.
xmt_small_bufs	78	Number of small-sized transmit buffers.
rcv_large_size	12K	Size (in bytes) of large-sized receive buffers.
rcv_large_bufs	69	Number of large-sized receive buffers.
rcv_small_size	0	Size (in bytes) of small-sized receive buffers.
rcv_small_bufs	0	Number of small-sized receive buffers. This size buffer is only used for AAL3/4.
reserved	0	

Relevant Structures

Table 5-3 and Table 5-4 describe the `atm_conf_t` structure, as defined in the *atm_b2h.h* file (which is automatically included in the *atm_user.h* file).

Table 5-4 Capability Flags for *atm_conf_t*

Flag	Mask	Description
ATM_CAP_AAL_1	0x0001	AAL1 supported
ATM_CAP_AAL_2	0x0002	AAL2 supported
ATM_CAP_AAL_34	0x0004	AAL3/4 supported
ATM_CAP_AAL_5	0x0008	AAL5 supported
ATM_CAP_AAL_0	0x0010	AAL0 (raw) supported
ATM_CAP_AAL_5_NOTRAILER	0x0020	AAL5 without trailer supported
ATM_CAP_AAL_MASK	0x003f	AAL mask
ATM_CAP_BARANGE	0x0100	Firmware supports variable size buffers (malloc).
ATM_CAP_IN_CKSUM	0x0200	Board's ffirmware does IP checksums.
ATM_CAP_LOOP_TIMING	0x0400	Board does loop timing. Set with `ATMIOC_SETOPT`.
ATM_CAP_DIAG_LOOPBACK	0x0800	Board receives what it sends. Set with `ATMIOC_SETOPT`.
ATM_CAP_LINE_LOOPBACK	0x1000	Board sends what it receives. Set with `ATMIOC_SETOPT`.

Errors

Possible errors include:

EFAULT	An error occurred when the driver was copying the retrieved data to the area specified by the pointer.
ENODEV	The board was not in the UP or DOWN state.
ETIME	The driver's command to the board timed out.

ATMIOC_GETIOSTAT

The ATMIOC_GETIOSTAT *ioctl()* command retrieves driver-internal I/O statistics.

Usage

Use the following format:

```
ioctl (fd_atm, ATMIOC_GETIOSTAT, &iostat);
```

where *iostat* is an atm_iostat_t structure.

Argument Values

The argument is a pointer to an atm_iostat_t structure.

Success or Failure

If successful, ATMIOC_GETIOSTAT returns zero. The out values should be read.

On failure, the *ioctl()* returns -1 with an error stored in errno. See the "Errors" heading for descriptions of individual errors.

Out Values

The retrieved values are written to the argument, summarized in Table 5-5.

Relevant Structures

The `atm_iostat_t` structure, from the *atm_user.h* file, is described in Table 5-5.

Table 5-5 Retrieved Values for ATMIOC_GETIOSTAT

Field	Description
ipkts	Count of total incoming packets over CDEV interfaces
ibytes	Count of total incoming bytes over CDEV interfaces
ierrs	Count of total incoming errors over CDEV interfaces
opkts	Count of total outgoing packets over CDEV interfaces
obytes	Count of total outgoing bytes over CDEV interfaces
oerrs	Count of total outgoing errors over CDEV interfaces
xcmd_dly	Count of commands that were delayed (not immediately placed on the command queue) due to heavy use of the command interface
xmit_dly	Count of transmit commands that were delayed (not immediately placed on the command queue) due to heavy use of the command interface
intrs	Count of host-to-board interrupts
b2hs	Count of board-to-host interrupts
xmit_reqs	Count of transmit requests
h2b_kicks	Number of times host has reset the board
xmit_intrs	Count of transmit interrupts
odone_intrs	Count of transmit done messages sent by board to host. When this count equals the `xmit_reqs` count, all data on the transmit queues has been processed
recv_intrs	Count of receive interrupts
fet_stat	Number of times host has retrieved board status (that is, number of times XCMD_FET_STAT has been called)

Errors

Possible errors include:

EFAULT	An error occurred when the driver was copying the retrieved data to the area specified by the pointer.
ENODEV	The board was not in the UP or DOWN state.

ATMIOC_GETMACADDR

The `ATMIOC_ GETMACADDR` *ioctl()* command reads the media access control (MAC) address from the ATM-OC3c board.

Usage

Use the following format:

`ioctl (`*fd_atm*`, ATMIOC_GETMACADDR, &`*addr*`);`

where *addr* is an array of `atm_macaddr_t` structures.

Argument Values

The argument is a pointer to an `atm_macaddr_t[6]`, an array of 6 unsigned chars.

Success or Failure

If successful, `ATMIOC_GETMACADDR` returns zero. The out values should be read.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Out Values

The retrieved MAC address is written to the call's argument.

Errors

Possible errors include:

EADDRNOTAVAILThe checksum on the retrieved address is not correct.

EFAULT	An error occurred when the driver was copying the retrieved data to the area specified by the pointer.
ENODEV	The board was not in the UP or DOWN state.
ETIME	The driver's command to the board timed out.

ATMIOC_GETOPT

The `ATMIOC_GETOPT` *ioctl()* command retrieves the current settings for the ATM-OC3c board's loopback and clock recover options. Requires superuser access.

Usage

Use the following format:

```
ioctl (fd_atm, ATMIOC_GETOPT, &int);
```

Argument Values

The argument is a pointer to an unsigned integer.

Success or Failure

If successful, `ATMIOC_GETOPT` returns zero. The out values should be read.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Out Values

The retrieved option setting (mask) is written to the location provided in the argument. Table 5-13 summarizes the values and masks that are available. The options are described in Table 5-14. The value for normal operation, which is also the default, is ATM_OPT_LOOP_TIMING (that is, 0x1).

Errors

Possible errors include:

EPERM	The invoker does not have superuser access privileges.
EFAULT	An error occurred when the driver was copying the retrieved data to the area specified by the pointer.
ENODEV	The board was not in the UP or DOWN state.
ETIME	The driver's command to the board timed out.

ATMIOC_GETRATEQ

The `ATMIOC_GETRATEQ` *ioctl()* command retrieves information about one rate queue from the ATM-OC3c board. The board must be in the UP state.

Usage

Use the following format:

`ioctl (`*fd_atm*`, ATMIOC_GETRATEQ, &`*rateq*`);`

where *rateq* is an `atm_rate_q_t` structure.

Argument Values

The argument is a pointer to an `atm_rate_q_t` structure, set up as described in Table 5-6. The *rate_queue_number* field of the argument must be set to one of the values described in Table 5-7.

Table 5-6 Recommended Values for ATMIOC_GETRATEQ's Argument

Fields	Value	Description
rate_queue_number	From Table 5-7	The queue whose rate is to be retrieved.
rate_value	Zero	
	Upon return = out value	Out value: 11-bit code from Table A-1

Table 5-7 Rate Queue Identification Values

Name	int	Description
RQ_A0	0	High priority Bank A, queue 0
RQ_A1	1	High priority Bank A, queue 1
RQ_A2	2	High priority Bank A, queue 2
RQ_A3	3	High priority Bank A, queue 3
RQ_B0	4	Low priority Bank B, queue 0
RQ_B1	5	Low priority Bank B, queue 1
RQ_B2	6	Low priority Bank B, queue 2
RQ_B3	7	Low priority Bank B, queue 3

Success or Failure

If successful, `ATMIOC_GETRATEQ` returns zero. The out value should be read.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Out Values

The retrieved value is written to the least significant word (the *rate_value* field) of the `atm_rate_q_t` structure that is identified by the argument. The *rate_value* is one of the rate codes summarized in the table in the Appendix A.

Relevant Structures

Table 5-6 describes the `atm_rate_q_t` structure, and its definition is included below, as it is in the *atm_b2h.h* file (included in the *atm_user.h* file):

```
typedef struct atm_rate_q {
    u_int rate_queue_number;
    u_int rate_value;
} atm_rate_q_t;
```

Errors

Possible errors include:

EFAULT	An error occurred when the driver was copying the retrieved data to the area specified by the pointer.
EINVAL	The specified rate queue identification number is invalid.
ENODEV	The board is not in the UP state.

ATMIOC_GETSTAT

The `ATMIOC_GETSTAT` *ioctl()* command reads and returns the ATM-OC3c board's operational status.

Usage

Use the following format:

`ioctl (`*fd_atm*`, ATMIOC_GETSTAT, &`*stat*`);`

where *stat* is an `atm_stat_t` structure.

Argument Values

The argument is a pointer to an empty `atm_stat_t` structure (described in Table 5-8).

Success or Failure

If successful, `ATMIOC_GETSTAT` returns zero. The out values should be read.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Out Values

The retrieved statistical data are written to the argument, described in Table 5-8. Figure 5-1 illustrates individual bits within the status fields of the `atm_stat_t` structure.

Table 5-8 Values Retrieved by ATMIOC_GETSTAT

Field	Description
hwstate	The current state of the board: 0 = ATM_HWSTATE_PREINIT 1 = ATM_HWSTATE_DEAD 2 = ATM_HWSTATE_DOWN 3 = ATM_HWSTATE_UP These states are described on page 121.
as_rx_packets_ok	Total packets received OK
as_rx_cell_if_parity	Cell error: parity error on cell interface
as_rx_cell_crc_error	Cell error: CRC-10 error
as_rx_cell_seq_err	Cell error: received out of sequence
as_rx_cell_size_err	Cell error: size violates AAL 3 4
as_rx_cell_pkt_term	Cell error: short cell terminated packet
as_rx_pkt_timeout	Received packet error: reassembly never completed
as_rx_pkt_bfr_oflo	Received packet error: reassembly exceeded buffer size
as_rx_pkt_crc_err	Received packet error: packet CRC-32 was bad
as_rx_pkt_new_pkt	Received packet error: new packet arrived, old not done
as_rx_unknown_err	Received packet error: none of the above
as_rx_cbr_cells	CBR cells received
as_rx_raw_cells	Raw cells received
as_rx_bytes_received	Total bytes received without error

Table 5-8 (continued) Values Retrieved by ATMIOC_GETSTAT

Field	Description
as_carrier_losses	SUNI statistics
as_carrier_restorations	SUNI statistics
as_carrier_transitions	SUNI statistics
as_FF_pkt_ok	FFRED completion count: packet sent without error
as_FF_pkt_flush	FFRED completion count packet flushed, flush command
as_FF_pkt_pm_parity	FFRED completion count packet flushed, PM parity error
as_RF_exc_oos_com	RFRED exception: out of sequence COM cell received
as_RF_exc_oos_eom	RFRED exception: out of sequence EOM cell received
as_RF_exc_smlbuf	RFRED exception: no small RX buffer available, packet drop
as_RF_exc_lrgbuf	RFRED exception: no large RX buffer available, packet drop
as_RF_exc_invvci	RFRED exception: cell received with invalid VCI
as_RF_exc_invvpi	RFRED exception: cell received with invalid VPI
as_SONET_sbe	SONET Section BIP-8 errors
as_SONET_lbe	SONET Line BIP-24 errors
as_SONET_lfe	SONET Line FEBEs
as_SONET_pbe	SONET Path BIP-8 errors
as_SONET_pfe	SONET Path FEBEs
as_SONET_chcs	SONET correctable ATM HEC errors
as_SONET_uhcs	SONET non-correctable ATM HEC errors

Table 5-8 (continued) Values Retrieved by ATMIOC_GETSTAT

Field	Description
as_SONET_status	See Table 5-9
as_FF_cells_sent	FFRED's total cells transferred
as_FF_status	See Table 5-11
as_RF_status	See Table 5-11
as_RF_cells_rcvd	RFRED: total non-error cells received
as_RF_dropped_pkt	RFRED: dropped because no free receive buffers
as_RF_err_count	RFRED: cells with CRC errors
as_RF_dropped_cbr	RFRED: lack of space in the CBR queue

The meanings for the bits in the three status fields (defined in the *atm_b2h.h* file) are described in Table 5-9 through Table 5-11, and Figure 5-1.

Table 5-9 Bits in *as_SONET_status* Field

Status Item	Mask	Comments
SONET_junk	0x8cd3001c	Ignore these bits
SONET_losv	0x40000000	Loss-of-signal state (Reg 0x11)
SONET_lofv	0x20000000	Loss-of-frame state (Reg 0x11)
SONET_oofv	0x10000000	Out-of-frame state (Reg 0x11)
SONET_ferf	0x02000000	Far-end-receive-failure (Reg 0x18)
SONET_lais	0x01000000	Line Alarm Indication Signal (Reg 0x18)
SONET_plop	0x00200000	Loss of Path (Reg 0x30)
SONET_pais	0x00080000	Path Alarm Indication Signal (Reg 0x30)
SONET_pyel	0x00040000	Path Yellow Condition (Reg 0x30)
SONET_psl	0x0000ff00	Path Signal Label (C2) (Reg 0x37)
SONET_oocd	0x00000080	Out-of-cell-delineation (Reg 0x50)

Table 5-9 (continued) Bits in *as_SONET_status* Field

Status Item	Mask	Comments
SONET_tsoci	0x00000040	Xmit start-of-cell error (Reg 0x60)
SONET_tfovr	0x00000020	Xmit FIFO Overrun (Reg 0x60)
SONET_rfovr	0x00000002	Recv FIFO Overrun (Reg 0x51)
SONET_rfudr	0x00000001	Recv FIFO Underrun (Reg 0x51)

Table 5-10 Bits in *as_FF_status* Field

Status Item	Mask	Comments
FS_JUNK	0x003f00ff	Ignore these bits. See the F-FRED / SARA-T specification for these bits.
FS_CM_PARERR	0x80000000	
FS_NORM_PM_PARER	0x40000000	
FS_CBR_PM_PARERR	0x20000000	
FS_TCQ_NOT_EMPTY	0x10000000	
FS_TCQ_FULL	0x08000000	
FS_CELL_CTR_OF	0x04000000	
FS_XMIT_DONE	0x02000000	
FS_CBR_DONE	0x01000000	
FS_RQ_BANKA_MIS	0x00800000	
FS_RQ_BANKB_MIS	0x00400000	
FS_OFF_LINE	0x00008000	
FS_PRQ_FULL	0x00004000	
FS_PRQ_EMPTY	0x00002000	
FS_TCQ_EMPTY	0x00001000	

Table 5-10 (continued) Bits in *as_FF_status* Field

Status Item	Mask	Comments
FS_CM_ERROR	0x00000800	
FS_VERSION	0x00000700	

Table 5-11 Bits in *as_RF_status* Field

Status Item	Mask	Comments
RS_JUNK	0x08007000	Ignore these bits. See R-FRED / SARA-R specification for the details.
RS_PKT_CTR_OF	0x80000000	
RS_ERR_CTR_OF	0x40000000	
RS_CBR_CTR_OF	0x20000000	
RS_CELL_CTR_OF	0x10000000	
RS_CM_PARERR	0x04000000	
RS_SML_FREEQ_MT	0x02000000	
RS_LRG_FREEQ_MT	0x01000000	
RS_EXCPQ_FL_I	0x00800000	
RS_PCQ_FL_I	0x00400000	
RS_CBRQ_FL_I	0x00200000	
RS_RAWQ_FL_I	0x00100000	
RS_EXCP_RCVD	0x00080000	
RS_PKT_RCVD	0x00040000	
RS_CBR_RCVD	0x00020000	
RS_RAW_RCVD	0x00010000	
RS_OFF_LINE	0x00008000	

Table 5-11 (continued) Bits in *as_RF_status* Field

Status Item	Mask	Comments
RS_RAWQ_FULL	0x00000800	
RS_RAWQ_EMPTY	0x00000400	
RS_CBRQ_FULL	0x00000200	
RS_CBRQ_EMPTY	0x00000100	
RS_EXCPQ_FULL	0x00000080	
RS_EXCPQ_EMPTY	0x00000040	
RS_PCQ_FULL	0x00000020	
RS_PCQ_EMPTY	0x00000010	
RS_LRGQ_FULL	0x00000008	
RS_LRGQ_EMPTY	0x00000004	
RS_SMLQ_FULL	0x00000002	
RS_SMLQ_EMPTY	0x00000001	

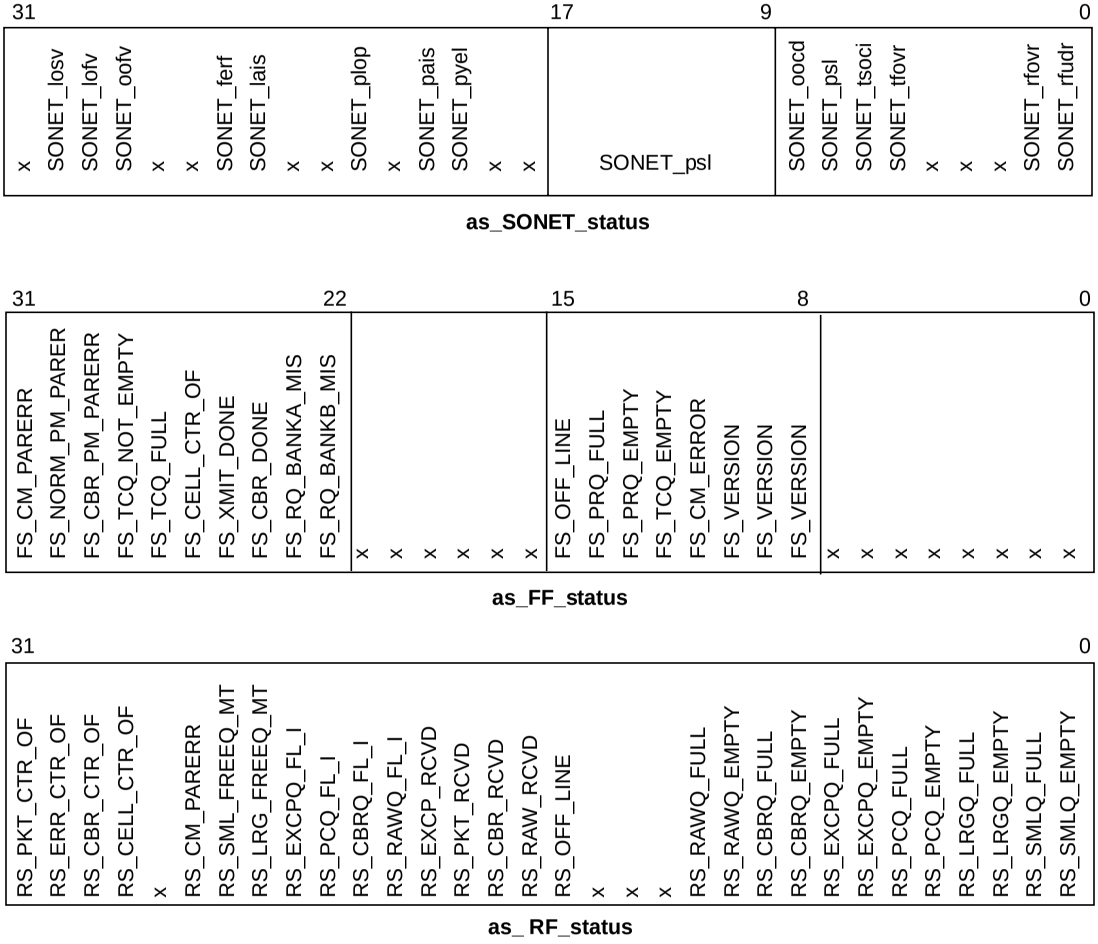


Figure 5-1 Bit Descriptions for Status Fields Within *atm_stat_t*

Relevant Structures

The `atm_stat_t` structure is described in Table 5-8, and included below as it is defined in the *atm_user.h* file and its included *atm_b2h.h* file:

```
typedef struct atm_stat {
    u_int hwstate;
    u_int as_rx_packets_ok;
    u_int as_rx_cell_if_parity;
    u_int as_rx_cell_crc_error;
    u_int as_rx_cell_seq_err;
    u_int as_rx_cell_size_err;
    u_int as_rx_cell_pkt_term;


u_int as_rx_pkt_timeout;
    u_int as_rx_pkt_bfr_oflo;
    u_int as_rx_pkt_crc_err;
    u_int as_rx_pkt_new_pkt;
    u_int as_rx_unknown_err;


u_int as_rx_cbr_cells;
    u_int as_rx_raw_cells;
    u_int as_rx_bytes_received;


u_int as_carrier_losses;
    u_int as_carrier_restorations;
    u_int as_carrier_transitions;


u_int as_FF_pkt_ok;
    u_int as_FF_pkt_flush;
    u_int as_FF_pkt_pm_parity;


u_int as_RF_exc_oos_com;
    u_int as_RF_exc_oos_eom;
    u_int as_RF_exc_smlbuf;
    u_int as_RF_exc_lrgbuf;
    u_int as_RF_exc_invvci;
    u_int as_RF_exc_invvpi;
```

```
u_int as_SONET_sbe;
    u_int as_SONET_lbe;
    u_int as_SONET_lfe;
    u_int as_SONET_pbe;
    u_int as_SONET_pfe;
    u_int as_SONET_chcs;
    u_int as_SONET_uhcs;
    u_int as_SONET_status;

    u_int as_FF_cells_sent;
    u_int as_FF_status;
    u_int as_RF_status;
    u_int as_RF_cells_rcvd;
    u_int as_RF_dropped_pkt;
    u_int as_RF_dropped_cbr;
} atm_stat_t;

/* bit fields in SONET_status */
#define SONET_junk 0x8cd3001c
#define SONET_losv 0x40000000
#define SONET_lofv 0x20000000
#define SONET_oofv 0x10000000
#define SONET_ferf 0x02000000
#define SONET_lais 0x01000000
#define SONET_plop 0x00200000
#define SONET_pais 0x00080000
#define SONET_pyel 0x00040000
#define SONET_psl 0x0000ff00
#define SONET_oocd 0x00000080
#define SONET_tsoci 0x00000040
#define SONET_tfovr 0x00000020
#define SONET_rfovr 0x00000002
#define SONET_rfudr 0x00000001

/* bits in FF_status */
#define FS_JUNK 0x003f00ff
#define FS_CM_PARERR 0x80000000
#define FS_NORM_PM_PARER 0x40000000
#define FS_CBR_PM_PARERR 0x20000000
#define FS_TCQ_NOT_EMPTY 0x10000000
#define FS_TCQ_FULL 0x08000000
#define FS_CELL_CTR_OF 0x04000000
#define FS_XMIT_DONE 0x02000000
#define FS_CBR_DONE 0x01000000
#define FS_RQ_BANKA_MIS 0x00800000
```

```
#define FS_RQ_BANKB_MIS 0x00400000
#define FS_OFF_LINE 0x00008000
#define FS_PRQ_FULL 0x00004000
#define FS_PRQ_EMPTY 0x00002000
#define FS_TCQ_EMPTY 0x00001000
#define FS_CM_ERROR 0x00000800
#define FS_VERSION 0x00000700

/* bits in RF_status */
#define RS_JUNK 0x08007000
#define RS_PKT_CTR_OF 0x80000000
#define RS_ERR_CTR_OF 0x40000000
#define RS_CBR_CTR_OF 0x20000000
#define RS_CELL_CTR_OF 0x10000000
#define RS_CM_PARERR 0x04000000
#define RS_SML_FREEQ_MT 0x02000000
#define RS_LRG_FREEQ_MT 0x01000000
#define RS_EXCPQ_FL_I 0x00800000
#define RS_PCQ_FL_I 0x00400000
#define RS_CBRQ_FL_I 0x00200000
#define RS_RAWQ_FL_I 0x00100000
#define RS_EXCP_RCVD 0x00080000
#define RS_PKT_RCVD 0x00040000
#define RS_CBR_RCVD 0x00020000
#define RS_RAW_RCVD 0x00010000
#define RS_OFF_LINE 0x00008000
#define RS_RAWQ_FULL 0x00000800
#define RS_RAWQ_EMPTY 0x00000400
#define RS_CBRQ_FULL 0x00000200
#define RS_CBRQ_EMPTY 0x00000100
#define RS_EXCPQ_FULL 0x00000080
#define RS_EXCPQ_EMPTY 0x00000040
#define RS_PCQ_FULL 0x00000020
#define RS_PCQ_EMPTY 0x00000010
#define RS_LRGQ_FULL 0x00000008
#define RS_LRGQ_EMPTY 0x00000004
#define RS_SMLQ_FULL 0x00000002
#define RS_SMLQ_EMPTY 0x00000001
```

Errors

Possible errors include:

EFAULT	An error occurred when the driver was copying the retrieved data to the area specified by the pointer.
ENOMEM	The driver was unable to place a command on the host-to-board command queue due to lack of memory.

ATMIOC_SETCONF

The `ATMIOC_SETCONF` *ioctl()* command configures the ATM-OC3c board. The new configuration takes effect when the board is next brought into the UP state. This command is available only to the superuser.

Usage

Use the following format:

`ioctl (`*fd_atm*`, ATMIOC_SETCONF, &`*conf*`);`

where *conf* is an `atm_conf_t` structure.

Argument Values

The pointer to *conf* identifies an instance of an `atm_conf_t` structure. The desired configuration values must be in the `atm_conf_t` structure, as described in Table 5-12.

Table 5-12 Recommended Values for ATMIOC_SETCONF's Argument

Field	Recommended Setting	Comments
sign	ATM_MAGIC	ATM-OC3c board's signature.
vers	varies	ATM_MIN_VERS, ATM_VERS_MASK, ATM_CKSUM_VERS as defined in *sys/atm_b2h.h.* ATM-OC3c board's / FLASH EPROMs version.
flags	0x1E28	Flags indicating various functions for which the ATM-OC3c board and its firmware's are capable. For example: 0x0008 = ATM_CAP_AAL_5, board uses AAL5 0x0200 =ATM_CAP_IN_CKSUM, board does IP checksum (the full set of values are in *sys/atm_b2h.h*)

Table 5-12 (continued) Recommended Values for ATMIOC_SETCONF's

Field	Recommended Setting	Comments
xtype	2	Transmission type: 1 =XT_UNKNOWN 2 =XT_STS3C, SONET STS-3c PHY at 155.52 Mbps 3 =XT_DS3=3, DS3 PHY at 44.736 Mbps 4 =XT_4B5B=4, 4B/5B encoding PHY at 100 Mbps 5 =XT_8B10B, 8B/10B encoding PHY at 155.52 Mbps
mtype	4	Media type: 1 =MT_UNKNOWN 2 =MT_COAX, coax cable 3 =MT_SMF, single-mode fiber 4 =MT_MMF, multi-mode fiber 5 =MT_STP, Shielded twisted pair 6 =MT_UTP, Unshielded twisted pair
maxvpibits	8	Maximum number of bits that can be used for a VPI. Range of possible values is 0 to 8.
maxvcibits	16	Maximum number of bits that can be used by a VCI. Range of possible values is 0 to 16.
hi_pri_qs	4	Number of high priority rate queues supported by the board.
lo_pri_qs	4	Number of low priority rate queues supported by the board.
xmt_large_size	12K	Size (in bytes) of large-sized transmit buffers.
xmt_large_bufs	78	Number of large-sized transmit buffers.
xmt_small_size	2K	Size (in bytes) of small-sized transmit buffers.
xmt_small_bufs	78	Number of small-sized transmit buffers.
rcv_large_size	12K	Size (in bytes) of large-sized receive buffers.
rcv_large_bufs	69	Number of large-sized receive buffers (for AAL5).
rcv_small_size	0	Size (in bytes) of small-sized receive buffers (for AAL3/4).
rcv_small_bufs	0	Number of small-sized receive buffers (for AAL3/4).
reserved	not valid	Reserved for future use.

Success or Failure

If successful, `ATMIOC_SETCONF` returns zero.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Relevant Structures

The `atm_conf_t` structure is explained in Table 5-12.

Errors

Possible errors include:

EFAULT	An error occurred during a copy of the data.
ENODEV	The board was not in the UP or DOWN state.
EPERM	The invoker does not have root (superuser) access privileges.
ETIME	The driver's call to the board timed out.

ATMIOC_SETOPT

The `ATMIOC_SETOPT` *ioctl()* command configures the ATM-OC3c board's loopback and clock recover options. The board starts functioning with the new options almost immediately. These options are useful only for testing purposes. This command is available only to the superuser.

Caution: This command is intended for testing purposes. Altering the options to anything other than the default (ATM_OPT_LOOP_TIMING) makes the board dysfunctional for normal operation.

Usage

Use the following format:

`ioctl (`*fd_atm*`, ATMIOC_SETOPT, `*opt*`);`

where *opt* is a `u_int`.

Argument Values

The *opt* is an unsigned integer that sets (enables) the bit or bits controlling the board options. The normal and default setting is ATM_OPT_LOOP_TIMING. Table 5-13 summarizes other values and the masks that are available. The options are described in Table 5-14.

Table 5-13 Recommended Values for ATMIOC_SETOPT's Argument

Possible Values	Can Be Combined With	Do Not Combine With
ATM_OPT_LOOP_TIMING (This is the default.)	Normal operation or ATM_OPT_LINE_LOOPBACK	ATM_OPT_DIAG_LOOPBACK
ATM_OPT_DIAG_LOOPBACK	nothing	ATM_OPT_LOOP_TIMING or ATM_OPT_LINE_LOOPBACK
ATM_OPT_LINE_LOOPBACK	ATM_OPT_LOOP_TIMING	Normal operation or ATM_OPT_DIAG_LOOPBACK

Table 5-14 ATM-OC3c Board's Options

Mask	Option	Description
0x1	Loop Timing (ATM_OPT_LOOP_TIMING)	When Loop Timing is enabled (bit 0 is set to 1), the board's logic obtains its SONET transmission clock from the clock signal recovered from the incoming ODL. Typically, this option is enabled for situations when the port is attached to an ATM switch, such as normal operation or Line Loopback testing. When Loop Timing is disabled (bit 0 is set to 0), the board uses its own clock (from the on-board crystal). This option must be disabled for Diagnostic Loopback testing. It is also appropriate to disable this option when the port's output line is attached to its own input line or when the port is attached to another ATM system that is not a switch.
0x2	Diagnostic Loopback (1 in Figure 5-2) (ATM_OPT_DIAG_LOOPBACK)	When Diagnostic Loopback is enabled (bit 1 is set to 1), the SUNI chip's internal loopback path is enabled, so that the R-FRED receives from the F-FRED. This option must be disabled for normal operation and when Line Loopback is enabled. Refer to Figure 5-2.
0x4	Line Loopback (2 in Figure 5-2) (ATM_OPT_LINE_LOOPBACK)	When Line Loopback is enabled (bit 2 is set to 1), the SUNI chip's external loopback path is enabled, so that the SUNI transmits to the outgoing ODL exactly what it receives from the incoming ODL. This option must be disabled for normal operation and when Diagnostic Loopback is enabled. Refer to Figure 5-2.

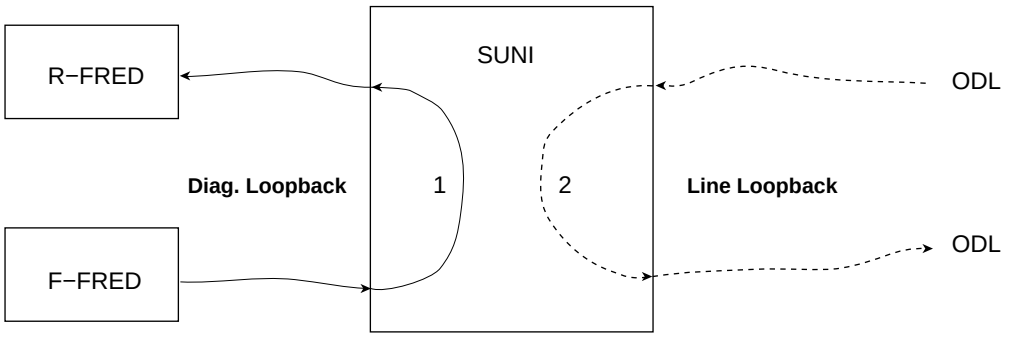


Figure 5-2 Loopback Options for ATM-OC3c Board

Success or Failure

If successful, `ATMIOC_SETOPT` returns zero.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Errors

Possible errors include:

ENODEV	The board was not in the UP or DOWN state.
EPERM	The invoker does not have root (superuser) access privileges.
ETIME	The driver's call to the board timed out.

ATMIOC_SETRATEQ

The `ATMIOC_SETRATEQ` *ioctl()* command sets the transmission rate for an individual rate queue. The new setting starts operating immediately. The board must be in the UP or DOWN state and the rate queue must be free (that is, not currently associated with any open VC).

See "Characteristics of the ATM-OC3c Hardware" in Chapter 1 for a description of the transmission rate queues and how they are managed by the IRIS ATM driver.

Usage

Use the following format:

`ioctl (`*fd_atm*`, ATMIOC_SETRATEQ, &`*rateq*`);`

where *rateq* is an `atm_rate_q_t` structure.

Argument Values

The pointer identifies an `atm_rate_q_t` structure that should be set up as shown in Table 5-15. The Rate Code (*rate_value*) must be one of the codes from the table in Appendix A.

Table 5-15 Recommended Values for ATMIOC_SETRATEQ's Argument

Field of atm_rate_q_t	Recommended Value	Comment
rate_queue_number	From Table 5-16	The rate queue identification number.
rate_value	0 or a code from Table A-1.	A code from Table A-1. To unlock the rate queue, thus making it available to the driver for dynamic resetting, set the field to zero.

Table 5-16 Rate Queue Identification Numbers

rate_queue_number	int	Description
RQ_A0	0	High priority Bank A, queue 0
RQ_A1	1	High priority Bank A, queue 1
RQ_A2	2	High priority Bank A, queue 2
RQ_A3	3	High priority Bank A, queue 3
RQ_B0	4	Low priority Bank B, queue 0
RQ_B1	5	Low priority Bank B, queue 1
RQ_B2	6	Low priority Bank B, queue 2
RQ_B3	7	Low priority Bank B, queue 3

Success or Failure

If successful, `ATMIOC_SETRATEQ` returns zero.

On failure, the *ioctl()* returns -1 with an error stored in `errno`. See the "Errors" heading for descriptions of individual errors.

Errors

Possible errors include:

EBUSY	The specified rate queue currently is servicing one or more VCs. The queue must be freed (that is, torn down) before it can be reconfigured.
EFAULT	An error occurred when the driver was copying the data.
EINVAL	The specified rate queue identification number is invalid.
ENODEV	The board is not in the UP state.
ENOMEM	The driver was unable to place the command on the host-to-board command queue due to lack of memory.
EPERM	The invoker does not have root (superuser) access privileges.

Appendix A

Rate Queue Information

To configure the transmission rate queues on the IRIS ATM-OC3c board, use the codes from the left (Code) column of Table A-1. The right (Cells per Second) column of the table summarizes the rate (in number of ATM cells per second) that each code configures.

One ATM cell consists of 53 bytes: 48 bytes of user payload and 5 bytes of ATM overhead. If you are interested in a different rate metric than cells per second, the formulas below can be used to make the conversion. The cells-per-second value in each formula is a value from the "Cells per Second" column in Table A-1.

- To calculate payload-bits per second, use: cells-per-second * 384
- To calculate payload-bytes per second, use: cells-per-second * 48
- To calculate VCC-bits per second, use: cells-per-second * 424
- To calculate VCC-bytes per second, use: cells-per-second * 53

Table A-1 Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x701	306
0x702	308
0x703	309
0x704	310
0x705	311
0x706	313
0x707	314
0x708	315

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x709	316
0x70A	318
0x70B	319
0x70C	320
0x70D	322
0x70E	323
0x70F	324
0x710	326
0x711	327
0x712	328
0x713	330
0x714	331
0x715	332
0x716	334
0x717	335
0x718	337
0x719	338
0x71A	340
0x71B	341
0x71C	343
0x71D	344
0x71E	346
0x71F	347
0x720	349
0x721	350
0x722	352
0x723	354

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x724	355
0x725	357
0x726	358
0x727	360
0x728	362
0x729	363
0x72A	365
0x72B	367
0x72C	369
0x72D	370
0x72E	372
0x72F	374
0x730	376
0x731	377
0x732	379
0x733	381
0x734	383
0x735	385
0x736	387
0x737	389
0x738	391
0x739	393
0x73A	395
0x73B	397
0x73C	399
0x73D	401
0x73E	403

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x73F	405
0x740	407
0x741	409
0x742	411
0x743	413
0x744	416
0x745	418
0x746	420
0x747	422
0x748	425
0x749	427
0x74A	429
0x74B	432
0x74C	434
0x74D	436
0x74E	439
0x74F	441
0x750	444
0x751	446
0x752	449
0x753	452
0x754	454
0x755	457
0x756	460
0x757	462
0x758	465
0x759	468

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x75A	471
0x75B	473
0x75C	476
0x75D	479
0x75E	482
0x75F	485
0x760	488
0x761	491
0x762	494
0x763	498
0x764	501
0x765	504
0x766	507
0x767	511
0x768	514
0x769	517
0x76A	521
0x76B	524
0x76C	528
0x76D	531
0x76E	535
0x76F	539
0x770	543
0x771	546
0x772	550
0x773	554
0x774	558

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x775	562
0x776	566
0x777	570
0x778	574
0x779	579
0x77A	583
0x77B	587
0x77C	592
0x77D	596
0x77E	601
0x77F	606
0x780	610
0x781	615
0x782	620
0x783	625
0x784	630
0x785	635
0x786	640
0x787	646
0x788	651
0x789	657
0x78A	662
0x78B	668
0x78C	673
0x78D	679
0x78E	685
0x78F	691

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x790	698
0x791	704
0x792	710
0x793	717
0x794	723
0x795	730
0x796	737
0x797	744
0x798	751
0x799	758
0x79A	766
0x79B	774
0x79C	781
0x79D	789
0x79E	797
0x79F	805
0x7A0	814
0x7A1	822
0x7A2	831
0x7A3	840
0x7A4	849
0x7A5	859
0x7A6	868
0x7A7	878
0x7A8	888
0x7A9	898
0x7AA	908

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x7AB	919
0x7AC	930
0x7AD	941
0x7AE	953
0x7AF	965
0x7B0	977
0x7B1	989
0x7B2	1002
0x7B3	1015
0x7B4	1028
0x7B5	1042
0x7B6	1056
0x7B7	1070
0x7B8	1085
0x7B9	1100
0x7BA	1116
0x7BB	1132
0x7BC	1149
0x7BD	1166
0x7BE	1184
0x7BF	1202
0x7C0	1221
0x601	1225
0x602	1230
0x603	1235
0x7C1	1240
0x605	1245

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x606	1250
0x607	1255
0x7C2	1260
0x609	1265
0x60A	1270
0x60B	1276
0x7C3	1281
0x60D	1286
0x60E	1291
0x60F	1297
0x7C4	1302
0x611	1308
0x612	1313
0x613	1319
0x7C5	1324
0x615	1330
0x616	1335
0x617	1341
0x7C6	1347
0x619	1353
0x61A	1359
0x61B	1365
0x7C7	1371
0x61D	1377
0x61E	1383
0x61F	1389
0x7C8	1395

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x621	1401
0x622	1408
0x623	1414
0x7C9	1420
0x625	1427
0x626	1433
0x627	1440
0x7CA	1447
0x629	1453
0x62A	1460
0x62B	1467
0x7CB	1474
0x62D	1481
0x62E	1488
0x62F	1495
0x7CC	1502
0x631	1510
0x632	1517
0x633	1524
0x7CD	1532
0x635	1539
0x636	1547
0x637	1555
0x7CE	1563
0x639	1570
0x63A	1578
0x63B	1586

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x7CF	1594
0x63D	1603
0x63E	1611
0x63F	1619
0x7D0	1628
0x641	1636
0x642	1645
0x643	1653
0x7D1	1662
0x645	1671
0x646	1680
0x647	1689
0x7D2	1698
0x649	1708
0x64A	1717
0x64B	1727
0x7D3	1736
0x64D	1746
0x64E	1756
0x64F	1766
0x7D4	1776
0x651	1786
0x652	1796
0x653	1806
0x7D5	1817
0x655	1827
0x656	1838

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x657	1849
0x7D6	1860
0x659	1871
0x65A	1883
0x65B	1894
0x7D7	1905
0x65D	1917
0x65E	1929
0x65F	1941
0x7D8	1953
0x661	1965
0x662	1978
0x663	1990
0x664	2003
0x665	2016
0x666	2029
0x667	2042
0x7DA	2056
0x669	2070
0x66A	2083
0x66B	2097
0x7DB	2111
0x66D	2126
0x66E	2140
0x66F	2155
0x7DC	2170
0x671	2185

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x672	2201
0x673	2216
0x7DD	2232
0x675	2248
0x676	2264
0x677	2281
0x7DE	2298
0x679	2315
0x67A	2332
0x67B	2350
0x7DF	2367
0x67D	2385
0x67E	2404
0x67F	2422
0x7E0	2441
0x681	2461
0x682	2480
0x683	2500
0x7E1	2520
0x685	2541
0x686	2561
0x687	2583
0x7E2	2604
0x689	2626
0x68A	2648
0x68B	2671
0x7E3	2694

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x68D	2717
0x68E	2741
0x68F	2765
0x7E4	2790
0x691	2815
0x692	2841
0x693	2867
0x7E5	2894
0x695	2921
0x696	2948
0x697	2976
0x7E6	3005
0x699	3034
0x69A	3064
0x69B	3094
0x7E7	3125
0x69D	3157
0x69E	3189
0x69F	3222
0x7E8	3255
0x6A1	3289
0x6A2	3324
0x6A3	3360
0x7E9	3397
0x6A5	3434
0x6A6	3472
0x6A7	3511

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x7EA	3551
0x6A9	3592
0x6AA	3634
0x6AB	3676
0x7EB	3720
0x6AD	3765
0x6AE	3811
0x6AF	3858
0x7EC	3906
0x6B1	3956
0x6B2	4006
0x6B3	4058
0x7ED	4112
0x6B5	4167
0x6B6	4223
0x6B7	4281
0x7EE	4340
0x6B9	4401
0x6BA	4464
0x6BB	4529
0x7EF	4596
0x6BD	4664
0x6BE	4735
0x6BF	4808
0x7F0	4883
0x501	4902
0x502	4921

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x503	4941
0x6C1	4960
0x505	4980
0x506	5000
0x507	5020
0x6C2	5040
0x509	5061
0x50A	5081
0x50B	5102
0x6C3	5123
0x50D	5144
0x50E	5165
0x50F	5187
0x7F1	5208
0x511	5230
0x512	5252
0x513	5274
0x6C5	5297
0x515	5319
0x516	5342
0x517	5365
0x6C6	5388
0x519	5411
0x51A	5435
0x51B	5459
0x6C7	5482
0x51D	5507

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x51E	5531
0x51F	5556
0x520	5580
0x7F2	5580
0x521	5605
0x522	5631
0x523	5656
0x6C9	5682
0x525	5708
0x526	5734
0x527	5760
0x6CA	5787
0x529	5814
0x52A	5841
0x52B	5869
0x6CB	5896
0x52D	5924
0x52E	5952
0x52F	5981
0x7F3	6010
0x531	6039
0x532	6068
0x533	6098
0x6CD	6127
0x535	6158
0x536	6188
0x537	6219

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x6CE	6250
0x539	6281
0x53A	6313
0x53B	6345
0x6CF	6378
0x53D	6410
0x53E	6443
0x53F	6477
0x7F4	6510
0x541	6545
0x542	6579
0x543	6614
0x6D1	6649
0x545	6684
0x546	6720
0x547	6757
0x6D2	6793
0x549	6831
0x54A	6868
0x54B	6906
0x6D3	6944
0x54D	6983
0x54E	7022
0x54F	7062
0x7F5	7102
0x551	7143
0x552	7184

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x553	7225
0x6D5	7267
0x555	7310
0x556	7353
0x557	7396
0x6D6	7440
0x559	7485
0x55A	7530
0x55B	7576
0x6D7	7622
0x55D	7669
0x55E	7716
0x55F	7764
0x7F6	7813
0x561	7862
0x562	7911
0x563	7962
0x6D9	8013
0x565	8065
0x566	8117
0x567	8170
0x6DA	8224
0x569	8278
0x56A	8333
0x56B	8389
0x6DB	8446
0x56D	8503

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x56E	8562
0x56F	8621
0x7F7	8681
0x571	8741
0x572	8803
0x573	8865
0x6DD	8929
0x575	8993
0x576	9058
0x577	9124
0x6DE	9191
0x579	9259
0x57A	9328
0x57B	9398
0x6DF	9470
0x57D	9542
0x57E	9615
0x57F	9690
0x7F8	9766
0x581	9843
0x582	9921
0x583	10000
0x6E1	10081
0x585	10163
0x586	10246
0x587	10331
0x6E2	10417

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x589	10504
0x58A	10593
0x58B	10684
0x6E3	10776
0x58D	10870
0x58E	10965
0x58F	11062
0x7F9	11161
0x591	11261
0x592	11364
0x593	11468
0x6E5	11574
0x595	11682
0x596	11792
0x597	11905
0x598	12019
0x6E6	12019
0x599	12136
0x59A	12255
0x59B	12376
0x6E7	12500
0x59D	12626
0x59E	12755
0x59F	12887
0x7FA	13021
0x5A1	13158
0x5A2	13298

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x5A3	13441
0x6E9	13587
0x5A5	13736
0x5A6	13889
0x5A7	14045
0x6EA	14205
0x5A9	14368
0x5AA	14535
0x5AB	14706
0x6EB	14881
0x5AD	15060
0x5AE	15244
0x5AF	15432
0x7FB	15625
0x5B1	15823
0x5B2	16026
0x5B3	16234
0x6ED	16447
0x5B5	16667
0x5B6	16892
0x5B7	17123
0x6EE	17361
0x5B9	17606
0x5BA	17857
0x5BB	18116
0x6EF	18382
0x5BD	18657

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x5BE	18939
0x5BF	19231
0x7FC	19531
0x401	19608
0x402	19685
0x403	19763
0x5C1	19841
0x405	19920
0x406	20000
0x407	20080
0x5C2	20161
0x409	20243
0x40A	20325
0x40B	20408
0x5C3	20492
0x40D	20576
0x40E	20661
0x40F	20747
0x6F1	20833
0x411	20921
0x412	21008
0x413	21097
0x5C5	21186
0x415	21277
0x416	21368
0x417	21459
0x5C6	21552

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x419	21645
0x41A	21739
0x41B	21834
0x5C7	21930
0x41D	22026
0x41E	22124
0x41F	22222
0x6F2	22321
0x421	22422
0x422	22523
0x423	22624
0x5C9	22727
0x425	22831
0x426	22936
0x427	23041
0x5CA	23148
0x429	23256
0x42A	23364
0x42B	23474
0x5CB	23585
0x42D	23697
0x42E	23810
0x42F	23923
0x6F3	24038
0x431	24155
0x432	24272
0x433	24390

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x5CD	24510
0x435	24631
0x436	24752
0x437	24876
0x5CE	25000
0x439	25126
0x43A	25253
0x43B	25381
0x5CF	25510
0x43D	25641
0x43E	25773
0x43F	25907
0x7FD	26042
0x441	26178
0x442	26316
0x443	26455
0x5D1	26596
0x445	26738
0x446	26882
0x447	27027
0x5D2	27174
0x449	27322
0x44A	27473
0x44B	27624
0x5D3	27778
0x44D	27933
0x44E	28090

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x44F	28249
0x6F5	28409
0x451	28571
0x452	28736
0x453	28902
0x5D5	29070
0x455	29240
0x456	29412
0x457	29586
0x5D6	29762
0x459	29940
0x45A	30120
0x45B	30303
0x5D7	30488
0x45D	30675
0x45E	30864
0x45F	31056
0x6F6	31250
0x461	31447
0x462	31646
0x463	31847
0x5D9	32051
0x465	32258
0x466	32468
0x467	32680
0x5DA	32895
0x469	33113

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x46A	33333
0x46B	33557
0x5DB	33784
0x46D	34014
0x46E	34247
0x46F	34483
0x6F7	34722
0x471	34965
0x472	35211
0x473	35461
0x5DD	35714
0x475	35971
0x476	36232
0x477	36496
0x5DE	36765
0x479	37037
0x47A	37313
0x47B	37594
0x5DF	37879
0x47D	38168
0x47E	38462
0x47F	38760
0x7FE	39063
0x481	39370
0x482	39683
0x483	40000
0x5E1	40323

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x485	40650
0x486	40984
0x487	41322
0x5E2	41667
0x489	42017
0x48A	42373
0x48B	42735
0x5E3	43103
0x48D	43478
0x48E	43860
0x48F	44248
0x6F9	44643
0x491	45045
0x492	45455
0x493	45872
0x5E5	46296
0x495	46729
0x496	47170
0x497	47619
0x5E6	48077
0x499	48544
0x49A	49020
0x49B	49505
0x5E7	50000
0x49D	50505
0x49E	51020
0x49F	51546

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x6FA	52083
0x4A1	52632
0x4A2	53191
0x4A3	53763
0x5E9	54348
0x4A5	54945
0x4A6	55556
0x4A8	56818
0x5EA	56818
0x4A9	57471
0x4AA	58140
0x4AB	58824
0x5EB	59524
0x4AD	60241
0x4AE	60976
0x4AF	61728
0x6FB	62500
0x4B1	63291
0x4B2	64103
0x4B3	64935
0x5ED	65789
0x4B5	66667
0x4B6	67568
0x4B7	68493
0x5EE	69444
0x4B9	70423
0x4BA	71429

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x4BB	72464
0x5EF	73529
0x4BD	74627
0x4BE	75758
0x4BF	76923
0x7FF	78125
0x4C1	79365
0x4C2	80645
0x4C3	81967
0x5F1	83333
0x4C5	84746
0x4C6	86207
0x4C7	87719
0x5F2	89286
0x4C9	90909
0x4CA	92593
0x4CB	94340
0x5F3	96154
0x4CD	98039
0x4CE	100000
0x4CF	102041
0x6FD	104167
0x4D1	106383
0x4D2	108696
0x4D3	111111
0x5F5	113636
0x4D5	116279

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x4D6	119048
0x4D7	121951
0x5F6	125000
0x4D9	128205
0x4DA	131579
0x4DB	135135
0x5F7	138889
0x4DD	142857
0x4DE	147059
0x4DF	151515
0x6FE	156250
0x4E1	161290
0x4E2	166667
0x4E3	172414
0x5F9	178571
0x4E5	185185
0x4E6	192308
0x4E7	200000
0x4E8	208333
0x5FA	208333
0x4E9	217391
0x4EA	227273
0x4EB	238095
0x5FB	250000
0x4ED	263158
	Do not count on exceeding the rate (in aggregate) listed the cell above this line.

Table A-1 (continued) Rates Available for Rate Queues on ATM-OC3c Board

Code	ATM Cells per Second
0x4EE	277778
0x4EF	294118
0x6FF	312500
0x4F1	333333
0x5FF	353207

Appendix B

International Alphabet 5

This appendix contains the International Alphabet 5 (IA5) character set.

Table B-1 Binary Values for IA5 Characters

Character	Binary Value (hexadecimal notation)
Control @, NULL	0x00
Control A, SOH	0x01
Control B, STX	0x02
Control C, ETX	0x03
Control D, EOT	0x04
Control E, ENQ	0x05
Control F, ACK	0x06
Control G, BELL	0x07
Control H, Backspace	0x08
Control I, HTAB	0x09
Control J, Line feed	0x0A
Control K, VT	0x0B
Control L, Form feed	0x0C
Control M, Carriage return	0x0D
Control N, SO	0x0E
Control O, SI	0x0F
Control P, DLE	0x10
Control Q, DC1	0x11

Table B-1 (continued) Binary Values for IA5 Characters

Character	Binary Value (hexadecimal notation)
Control R, DC2	0x12
Control S, DC3	0x13
Control T, DC4	0x14
Control U, NAK	0x15
Control V, SYN	0x16
Control W, ETB	0x17
Control X, Cancel	0x18
Control Y, EM	0x19
Control Z, SUB	0x1A
Control [, Escape	0x1B
Control \, FS	0x1C
Control J. GS	0x1D
Control Control, RS	0x1E
Control _, US	0x1F
Space	0x20
! (exclamation mark)	0x21
" (neutral double quotation mark)	0x22
# (number or pound sign)	0x23
$ (dollar sign)	0x24
% (percent sign)	0x25
& (ampersand)	0x26
’ (apostrophe)	0x27
((left parenthesis)	0x28
) (right parenthesis)	0x29
* (asterisk)	0x2A
+ (plus, add)	0x2B

Table B-1 (continued) Binary Values for IA5 Characters

Character	Binary Value (hexadecimal notation)
, (comma)	0x2C
– (hyphen, minus)	0x2D
. (period)	0x2E
/ (slash, solidus)	0x2F
0 (zero)	0x30
1	0x31
2	0x32
3	0x33
4	0x34
5	0x35
6	0x36
7	0x37
8	0x38
9	0x39
: (colon)	0x3A
; (semicolon)	0x3B
< (less than)	0x3C
= (equal)	0x3D
> (greater than)	0x3E
? (question mark)	0x3F
@ (commercial at sign)	0x40
A	0x41
B	0x42
C	0x43
D	0x44
E	0x45

Table B-1 (continued) Binary Values for IA5 Characters

Character	Binary Value (hexadecimal notation)
F	0x46
G	0x47
H	0x48
I	0x49
J	0x4A
K	0x4B
L	0x4C
M	0x4D
N	0x4E
O	0x4F
P	0x50
Q	0x51
R	0x52
S	0x53
T	0x54
U	0x55
V	0x56
W	0x57
X	0x58
Y	0x59
Z	0x5A
[(left bracket)	0x5B
\ (back slash)	0x5C
] (right bracket)	0x5D
^ (up arrow)	0x5E
_ (under score)	0x5F

Table B-1 (continued) Binary Values for IA5 Characters

Character	Binary Value (hexadecimal notation)
‘ (accent grave)	0x60
a	0x61
b	0x62
c	0x63
d	0x64
e	0x65
f	0x66
g	0x67
h	0x68
i	0x69
j	0x6A
k	0x6B
l	0x6C
m	0x6D
n	0x6E
o	0x6F
p	0x70
q	0x71
r	0x72
s	0x73
t	0x74
u	0x75
v	0x76
w	0x77
x	0x78
y	0x79

Table B-1 (continued) Binary Values for IA5 Characters

Character	Binary Value (hexadecimal notation)
z	0x7A
{ (left curly bracket)	0x7B
| (vertical bar)	0x7C
} (right curly bracket)	0x7D
~ (tilde)	0x7E
Delete	0x7F

Cause and Diagnostic Codes

This appendix describes the information that is returned with ATM signalling requests. The cause codes that are described are provided as out values (in the `reject_reason_t` data structure or in the *cause* field of other data structures) for many of the ATM Signalling commands. The value in the *cause* field matches the numbers assigned by the ATM UNI standard to the message texts.

Table C-1 lists the cause codes (content of *cause* field) that are used by implementations that conform to the *ATM User-Network Interface Specification* (ATM UNI) standard. The "Comments" column points out codes that are specific to particular versions of the ATM UNI (for example, 3.0 and 3.1). Table C-2 lists implementation-specific (local) cause codes used by the IRIS ATM Signalling software. Table C-3 summarizes the diagnostic information that accompanies some of the cause codes. IRIS ATM does not currently pass these up to the higher-layer applications.

Table C-1 ATM UNI Cause Codes

Text for ATM UNI Cause	Value for *cause* Field	Comments
Unallocated / Unassigned Number	1	Additional information may be supplied. See Table C-3.
No Route to Specified Transit Network	2	
No Route to Destination	3	Additional information may be supplied. See Table C-3.
Unacceptable VPCI_VCI	10	
Normal_3.1	16	Not used with UNI 3.0. Used only with UNI 3.1

Table C-1 (continued) ATM UNI Cause Codes

Text for ATM UNI Cause	Value for *cause* Field	Comments
User Busy	17	
No User Responding	18	
Call Rejected	21	Additional information may be supplied. See Table C-3.
Number Changed	22	Additional information may be supplied. See Table C-3.
User Rejects Calls With Calling Line Identification Restriction (CLIR)	23	
Destination Out of Order	27	
Invalid Number Format	28	
Response to STATUS ENQUIRY	30	
Normal_3.0	31	Used only with UNI 3.0. Not used with UNI 3.1
Requested VPCI/VCI Unavailable	35	
VPCI Assignment Failure	36	
User Cell Rate Unavailable	37	Not used with UNI 3.0. Used only with UNI 3.1
Network Out of Order	38	
Temporary Failure	41	
Access Information Discarded	43	Additional information may be supplied. See Table C-3.
No VPCI/VCI Available	45	
Resource Unavailable, Unspecified	47	

Table C-1 (continued) ATM UNI Cause Codes

Text for ATM UNI Cause	Value for *cause* Field	Comments
QOS Unavailable	49	Additional information may be supplied. See Table C-3.
User Cellrate Unavailable	51	Used only with UNI 3.0 Not used with UNI 3.1 Additional information may be supplied. See Table C-3.
Bearer Capability Not Authorized	57	
Bearer Capability Not Presently Available	58	
Service or Option Unavailable, Unspecified	63	
Bearer Capability Not Implemented	65	
Unsupported Combination of Traffic Parameters	73	
AAL Parameters Cannot Be Supported	78	Not used with UNI 3.0. Used only with UNI 3.1
Invalid Call Reference	81	
Identified Channel Does Not Exist	82	Additional information may be supplied. See Table C-3.
Incompatible Destination	88	Additional information may be supplied. See Table C-3.
Invalid Endpoint Reference	89	
Invalid Transit Network Selection	91	
Too Many Pending Add Party Requests	92	
AAL Parameters Cannot Be Supported	93	Used only with UNI 3.0. Not used with UNI 3.1

Table C-1 (continued) ATM UNI Cause Codes

Text for ATM UNI Cause	Value for *cause* Field	Comments
Mandatory Information Element Missing	96	Additional information may be supplied. See Table C-3.
Message Type Nonexistent or Not Implemented	97	Additional information may be supplied. See Table C-3.
Information Element Nonexistent or Not Implemented	99	Additional information may be supplied. See Table C-3.
Invalid Information Element Contents	100	Additional information may be supplied. See Table C-3.
Message Not Compatible With Call State	101	Additional information may be supplied. See Table C-3.
Recovery On Timer Expiry	102	Additional information may be supplied. See Table C-3.
Incorrect Message Length	104	
Protocol Error, Unspecified	111	

Table C-2 SGI Cause Codes

Text for SGI Cause	Value for *cause* Field	Comments
CAUSE_LOCALERROR	128	Local Error: unknown driver or signalling-daemon error
CAUSE_ALREADY	129	Registration denied: BLLI already taken, or application already registered
CAUSE_INVALBESTEFFORT	130	Best Effort requires that both directions be Best Effort & QOS_0
CAUSE_INVALCELLRATE	131	Invalid *cellrate* field
CAUSE_INVALBLLI	132	Invalid broadband low layer information (*blli*) code specified
CAUSE_INVALBEARERCLASS	133	Invalid bearer class
CAUSE_INVALADDRESSFMT	134	Invalid address format
CAUSE_NOTMULTI	135	Add or drop party on a point-to-point call
CAUSE_PARTYHANDLEINUSE	136	Trying to add a party using a party handle that has already been used
CAUSE_INVALPARTYHANDLE	137	Request was dropped because the party handle was not found

Table C-3 ATM UNI Diagnostics

Accompanying ATM UNI Cause	ATM UNI Diagnostic Provided	Diagnostic Values
Unallocated / Unassigned Number	One octet	The diagnostics[a] provide the following information: a description of the *condition*, whether the condition is *normal* or *abnormal*, and *who* supplied the diagnostic: *condition n/a who* 0x80 Unknown normal provider 0x81 Permanent normal provider 0x82 Transient normal provider 0x84 Unknown abnormal provider 0x85 Permanent abnormal provider 0x86 Transient abnormal provider 0x88 Unknown normal user 0x89 Permanent normal user 0x8A Transient normal user 0x8C Unknown abnormal user 0x8D Permanent abnormal user 0x8E Transient abnormal user
Call Rejected	Two octets	The diagnostics provide the following information: the first octet contains the *reason*, and a description of the *condition*.The second octet contains either user-specific values or the identifier for the ATM UNI information element (IE), whichever is appropriate. *reason condition* 0x80 user-specific unknown 0x81 user-specific permanent 0x82 user-specific transient 0x84 IE missing unknown 0x85 IE missing permanent 0x86 IE missing transient 0x88 IE missing unknown 0x89 IE missing permanent 0x8A IE missing transient

Table C-3 (continued) ATM UNI Diagnostics

Accompanying ATM UNI Cause	ATM UNI Diagnostic Provided	Diagnostic Values
No Route to Destination	One octet	Same as “Unallocated Number.”
Number Changed	6 to 25 octets	The new destination address formatted with a Called Party Number information element.
Access Information Discarded	One or more octets	Each octet specifies one ATM UNI information element identifier.
QOS Unavailable	One octet	Same as “Unallocated Number.”
User Cell Rate Unavailable	One or more octets	Each octet specifies one subfield identifier from the ATM UNI User Cell Rate information element.
Identified Channel Does Not Exist	4 octets	Most significant two octets specify VPCI value. Least significant two octets specify VCI value.
Incompatible Destination	1 octet	The ATM UNI information element identifier.
Mandatory Information Element Missing	1 or more octets	Each octet is one ATM UNI information element identifier.
Message Type Nonexistent or Not Implemented	One octet	Specifies one ATM UNI message type: for example, SETUP, RELEASE, CONNECT.
Information Element Nonexistent or Not Implemented	1 or more octets	Each octet is one ATM UNI information element identifier.
Invalid Information Element Contents	1 or more octets	Each octet is one ATM UNI information element identifier.

Table C-3 (continued) ATM UNI Diagnostics

Accompanying ATM UNI Cause	ATM UNI Diagnostic Provided	Diagnostic Values
Message Not Compatible With Call State	One octet	Specifies one ATM UNI message type: for example, SETUP, RELEASE, CONNECT.
Recovery On Timer Expiry	Three octets	Each octet specifies one IA5 character to indicate one numeral identifying an ATM UNI timer. For example, for the timer called “T308,” the first octet specifies “3,” the second “0,” and the third “8.”

a. IRIS ATM does not currently pass these diagnostics up to higher-layer applications.

Index

Tell Us About This Manual

As a user of Silicon Graphics documentation, your comments are important to us. They help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics to comment on:

- General impression of the document
- Omission of material that you expected to find
- Technical errors
- Relevance of the material to the job you had to do
- Quality of the printing and binding

Please include the title and part number of the document you are commenting on. The part number for this document is 007-2334-002.

Thank you!

Three Ways to Reach Us

The **postcard** opposite this page has space for your comments. Write your comments on the postage-paid card for your country, then detach and mail it. If your country is not listed, either use the international card and apply the necessary postage or use electronic mail or FAX for your reply.

If **electronic mail** is available to you, write your comments in an e-mail message and mail it to either of these addresses:

- If you are on the Internet, use this address: techpubs@sgi.com
- For UUCP mail, use this address through any backbone site: *[your_site]*!sgi!techpubs

You can forward your comments (or annotated copies of manual pages) to Technical Publications at this **FAX** number:

415 965-0964