# Digital Unix
# Native ATM
# Application Programming Interface

## Programmer's Reference for PVC Operations

## Digital Equipment Corporation

## Version 2.1

## July 31, 1997

## 1. Introduction

This programmer's reference describes the Digital Unix native ATM Application Programmer's Interface (API). Because the API does not yet support Switched Virtual Channels (SVCs), the scope of this document is limited to Permanent Virtual Channel (PVC) operations only.

The ATM API adds a new inter-process communication domain to Digital Unix that is available via the standard sockets interface.

## 2. API Usage (for PVCs)

PVCs, while connection-oriented, require no dynamic connection establishment. Therefore, using PVCs over sockets is quite simple. The normal connection establishment procedures for sockets are not needed with PVCs. It is only necessary to create a socket and execute an IO Control call to associate that socket with a PVC. The act of associating the socket with a PVC causes the PVC to be opened on the physical ATM interface. End-to-end establishment of the PVC (setting up the route over which the PVC's data flows) is the job of a network operator and discussion of it is beyond the scope of this document.

### *2.1 Overview of PVC Socket Operations*

An application that uses ATM PVCs needs to adhere to the following outline for each connection it establishes:

1. Create an ATM socket,
2. Determine how many physical ATM interfaces (adapters) are available on the system,
3. Determine which physical interface it wishes to communicate through,
4. (optional) Inform the socket of the Quality of Service parameters it wishes to use,
5. Associate the socket with a PVC, and
6. Communicate over the established connection.

In the above outline, step 2 may be performed on the first connection and assumed to be the same on subsequent connections. If step 4 is omitted, a default best-effort quality of service shall be used.

## 2.2 Socket Types and Protocols

When creating a socket for use with ATM PVCs, it is necessary to select the ATM address family (AF_AAL). The socket types supported are SOCK_RAW and SOCK_DGRAM. Both raw and datagram sockets provide direct access to the AAL5 layer and provide no transport protocol (SOCK_STREAM is not currently supported by this address family).

The user application need not be concerned with the AAL5 trailer on either send or receive. The trailer is automatically appended and stripped by the ATM kernel subsystem for transmit and receive respectively.

## 2.3 Socket Addressing

AF_AAL sockets use the sockaddr_aal structure (in <netaal/aal.h> ) to specify the address of an ATM connection endpoint. This structure provides addressing for SVCs only and is thus not relevant to PVC operations. PVC operations may be conducted without ever making a call to **bind** or **connect**.

## 2.4 ATM Socket Controls

Many of the ATM specific capabilities of the sockets API are handled through IO control (**ioctl**) calls. The following table describes the controls supported by ATM sockets.

**Table 1 - ATM Socket IO Controls**

| IOCTL Code | Data Type | Access | Description |
|---|---|---|---|
| `SIO_GET_ATM_DEVICE_COUNT` | `unsigned int` | read | Returns the number of physical ATM interfaces available to the user. |
| `SIO_GET_DEVICE_INFO` | `ATM_DEVICE_INFO` | read<br>write | Returns information describing a specific physical ATM interface. |
| `SIO_SET_ATM_PARAMETERS` | `ATM_PARAMS` | write | Sets Quality of Service parameters to be used when the socket sets up an ATM connection. |
| `SIO_ASSOCIATE_PVC` | `ATM_CONNECTION_ID` | write | Connects an unconnected socket to a PVC. When this ioctl completes successfully, the socket is ready for data transfer. |
| `SIO_GET_ATM_CONNECTION_ID` | `ATM_CONNECTION_ID` | read | Retrieves the connection ID associated with a socket. The connection ID contains the device number, the VPI, and VCI of the connection. |

The data types used in the above controls are defined in the following sections.

**Table 2 - ATM_DEVICE_INFO Structure**

| Member | Type | Description |
|---|---|---|
| `device_number` | `unsigned int` | On input: The device number being queried. |
| `driver_name` | `char *` | The device driver name. |
| `driver_name_len` | `unsigned int` | On input: The size of the driver_name buffer.<br>On output: The size of the returned device driver name. |
| `phy_type` | `unsigned int` | A description of the type of physical interface (i.e. T1, DS3, STS3, etc). Refer to the ATM_PHY_* definitions in <netaal/aal.h>. |
| `vc_max` | `unsigned int` | The total number of VCs that may be opened at one time on this interface. |
| `vc_avail` | `unsigned int` | The number of VCs available to be opened on this interface. This is vc_max minus the number of opened VCs on the interface. |
| `vci_max` | `unsigned int` | The maximum number of VCIs supported by the interface. |
| `vpi_max` | `unsigned int` | The maximum number of VPIs supported by the interface. |
| `vci_highest` | `unsigned int` | The highest value that may be used as a VCI on the interface. |
| `vpi_highest` | `unsigned int` | The highest value that may be used as a VPI on the interface. |
| `capabilities` | `unsigned int` | Bit mask indicating this interface's capabilities. Refer to the ATM_CAP_* definitions in <netaal/aal.h>. |
| `hard_mtu` | `unsigned int` | The largest PDU (in bytes) that the interface supports. |
| `peak_max` | `unsigned int` | The maximum peak cell rate in cells-per-second supported by this interface. |
| `burst_max` | `unsigned int` | The maximum burst cell rate in cells-per-second supported by this interface. |
| `sust_max` | `unsigned int` | The maximum sustainable cell rate in cells-per-second supported by this interface. |
| `peak_avail` | `unsigned int` | The peak cell rate in cells-per-second currently available on this interface. |
| `burst_avail` | `unsigned int` | The burst cell rate in cells-per-second currently available on this interface. |
| `sust_avail` | `unsigned int` | The sustainable cell rate in cells-per-second currently available on this interface. |
| `reserved` | `unsigned int` | Fields reserved for future expansion. |

**Table 3 - ATM_PARAMS Structure**

| ATM_PARAMS | | |
|---|---|---|
| **Member** | **Type** | **Description** |
| fqos | unsigned int | Forward Quality of Service class. Refer to Table 6 for a list of QoS class definitions. |
| bqos | unsigned int | Backward Quality of Service class. Refer to Table 6 for a list of QoS class definitions. |
| fmtu | unsigned int | Forward maximum frame size in bytes. |
| bmtu | unsigned int | Backward maximum frame size in bytes. |
| valid_rates | unsigned int | Bit mask indicating which of the specified cell rates are valid. Refer to the ATM_RATE_* definitions in <netaal/aal.h>. |
| fpeakcr[2] | unsigned int | Forward peak cell rate for CLP 0 and CLP 0+1. |
| bpeakcr[2] | unsigned int | Backward peak cell rate for CLP 0 and CLP 0+1. |
| fsustcr[2] | unsigned int | Forward sustainable cell rate for CLP 0 and CLP 0+1. |
| bsustcr[2] | unsigned int | Backward sustainable cell rate for CLP 0 and CLP 0+1. |
| fburstcr[2] | unsigned int | Forward burst cell rate for CLP 0 and CLP 0+1. |
| bburstcr[2] | unsigned int | Backward burst cell rate for CLP 0 and CLP 0+1. |
| bearer_class | unsigned int | Broadband Bearer class for the connection. See Table 6 for a list of bearer class definitions. |
| flags | unsigned int | Bit mask specifying the characteristics of the connection. See Table 8 for a list of flag definitions. |
| reserved | unsigned int | Fields reserved for future expansion. |

**Table 4 - ATM_CONNECTION_ID Structure**

| ATM_CONNECTION_ID | | |
|---|---|---|
| **Member** | **Type** | **Description** |
| device_number | unsigned int | Number from 0 to N-1 where N is the number returned by the SIO_GET_ATM_DEVICE_COUNT ioctl. |
| vpi | unsigned int | The virtual path identifier of the connection. |
| vci | unsigned int | The virtual channel identifier of the connection. |
| reserved | unsigned int | Fields reserved for future expansion. |

## 3. Error Codes

Table 5 lists the possible error codes for each system call.

**Table 5 - ATM Socket Error Codes**

| System Call | Error Code | Meaning |
|---|---|---|
| socket | ENOBUFS | Insufficient memory buffers to create socket. |
| | EAFNOSUPPORT | The AF_AAL domain does not exist. If this error occurs, the ATM Sockets API has not been properly installed. |
| SIO_GET_ATM_DEVICE_COUNT | <none> | This system call will never fail. |
| SIO_GET_DEVICE_INFO | ENODEV | The physical device specified does not exist. |
| SIO_SET_ATM_PARAMETERS | ENODEV | The physical device specified does not exist. |
| | EISCONN | The socket is already in the connected state. |
| SIO_ASSOCIATE_PVC | ENODEV | The physical device specified does not exist. |
| | EISCONN | The socket is already in the connected state. |
| | ENOMEM | Insufficient memory to associate the PVC. |
| | EADDRINUSE | The requested connection ID is already in use on this device. It may be in use by another socket or another convergence module (i.e. a LAN Emulation Client or a Logical IP Subnet). |
| | ENOTCONN | The connection establishment failed. |
| SIO_GET_ATM_CONNECTION_ID | ENOTCONN | The socket is not connected and therefore has no connection ID. |
| send | ENOBUFS | The transmit queue is full and the message was discarded. The user may attempt to retransmit the message. |
| | ENOMEM | Insufficient memory to complete the send operation. |
| | ENOTCONN | The socket is not in the connected state. This could happen because no connection was established or because the link failed. |
| | ENETUNREACH | The send operation failed for a reason not stated above. |
| recv | * | Error semantics for recv are identical to those in other address families (i.e. IF_ENET). |

# 4. Quality of Service Parameters

## 4.1 Default Parameter Values

If an ATM socket is created and connected without ever calling the SIO_ATM_PARAMETERS ioctl, a default set of parameters shall be used in the establishment of that connection. See Table 6 for a list of the possible values and their defaults.

**Table 6 - QoS Parameter Values and Defaults**

| Parameter | Variations | Possible Values | Default |
|---|---|---|---|
| qos | Forw/Back | ATM_QC_NONE<br>ATM_QC_CLASSA<br>ATM_QC_CLASSB<br>ATM_QC_CLASSC<br>ATM_QC_CLASSD<br>ATM_QC_CLASSX<br>ATM_QC_CLASSY | ATM_QC_NONE |
| max_frame_size | Forw/Back | 48 - 65535 | 8192 |
| peak_cell_rate | Forw/Back<br>CLP0/CLP0+1 | 0 - line_rate | line_rate |
| sust_cell_rate | Forw/Back<br>CLP0/CLP0+1 | 0 - line_rate | (not present) |
| burst_tolerance | Forw/Back<br>CLP0/CLP0+1 | 0 - line_rate | (not present) |
| bearer_class | --- | ATM_BC_CLASSA<br>ATM_BC_CLASSC<br>ATM_BC_CLASSX | ATM_BC_CLASSX |
| flags | --- | ATM_FLAG_BEI<br>ATM_FLAG_FTAG<br>ATM_FLAG_BTAG<br>ATM_FLAG_PACING<br>ATM_FLAG_CBR<br>ATM_FLAG_VBR<br>ATM_FLAG_TIMING<br>ATM_FLAG_NOTIMING<br>ATM_FLAG_CLIPPING | ATM_FLAG_BEI |

### 4.2 The ATM_DEFAULTS macro

The API provides a macro called ATM_DEFAULTS which is used to initialize the ATM parameters structure to the default values. Refer to the sample code at the end of this specification for an example of its use.

ATM_DEFAULTS allows the application programmer to specify the parameters by simply changing the salient values while leaving the majority of the fields to their default values. The most commonly used sets of parameter values will differ only slightly from the defaults.

### 4.3 Valid Rates Bit Mask

Depending on the type of service required for a connection, zero or more of the specifiable cell rates must be supplied by the user process. Because not all cell rates are needed at all times, the 'valid_rates' bit mask is provided as a means for the user to specify which cell rates are specified and which are "not present".

**Table 7 - Valid Rates Bit Mask Values**

| Mask Value | Associated Cell Rate Parameter |
|---|---|
| ATM_VR_FPEAK0 | `fpeakcr[0]` |
| ATM_VR_FPEAK1 | `fpeakcr[1]` |
| ATM_VR_BPEAK0 | `bpeakcr[0]` |
| ATM_VR_BPEAK1 | `bpeakcr[1]` |
| ATM_VR_FSUST0 | `fsustcr[0]` |
| ATM_VR_FSUST1 | `fsustcr[1]` |
| ATM_VR_BSUST0 | `bsustcr[0]` |
| ATM_VR_BSUST1 | `bsustcr[1]` |
| ATM_VR_FBURST0 | `fburstcr[0]` |
| ATM_VR_FBURST1 | `fburstcr[1]` |
| ATM_VR_BBURST0 | `bburstcr[0]` |
| ATM_VR_BBURST1 | `bburstcr[1]` |

### 4.4 Connection Flags

**Table 8 - Connection Flag Definitions**

| Flag Value | Meaning |
|---|---|
| ATM_FLAG_BEI | Best Effort Indicator - This connection shall use the appropriate best-effort mechanism depending on the capablilties of the network. The default mechanism is Unspecified Bit Rate (UBR). Available Bit Rate (ABR) and/or Digital's FLOWmaster may be used if these mechanisms are supported by the network. |
| ATM_FLAG_FTAG | Forward Tagging - This flag effects the way SVCs are signalled. It has no meaning for PVCs. |
| ATM_FLAG_BTAG | Backward Tagging - This flag effects the way SVCs are signalled. It has no meaning for PVCs. |
| ATM_FLAG_PACING | VC Pacing - This flag causes the connection to be signalled as best-effort but set up locally as a CBR connection. This allows a flow to be paced so as not to overrun the network. PVCs are set up as CBR. |
| ATM_FLAG_CBR | Constant Bit Rate - This flag causes the connection to be signalled as a CBR connection. PVCs are set up as CBR. |
| ATM_FLAG_VBR | Variable Bit Rate - This flag causes the connection to be signalled as a VBR connection. PVCs are aer up as VBR. |
| ATM_FLAG_TIMING | This flag effects the way SVCs are signalled. It has no meaning for PVCs. |
| ATM_FLAG_NOTIMING | This flag effects the way SVCs are signalled. It has no meaning for PVCs. |
| ATM_FLAG_CLIPPING | Susceptibility to Clipping - This flag effects the way SVCs are signalled. It has no meaning for PVCs. |

# 5. Helpful Hints

## 5.1 Allocating large amounts of CBR bandwidth

The Digital Unix ATM subsystem limits the amount of link bandwidth that can be reserved to 10% of the maximum link speed. This limit is configurable and may be changed temporarily or permanently. To change the limit temporarily, use the following commands:

```
sysconfig -r atm fsysresvlim=F
sysconfig -r atm bsysresvlim=B
```

where F is the forward reservation limit in percent of link and B is the backward reservation limit. To make these changes persistent through reboots, add the following lines to the end of /etc/sysconfigtab:

```
atm:
     fsysresvlim=F
     bsysresvlim=B
```

## 5.2 Working with the sb_max value.

The Digital UNIX sockets subsystem limits the amounts of socket buffer space requested for send and receive to the value of sb_max that is defined by the operating system. However, Digital UNIX also allows users to override the OS defined value temporarily or permanently. Users of the ATM sockets API should be aware that the API allocates 32767 bytes for the send socket buffer and 65536 bytes for the receive socket buffer. Therefore, lowering sb_max, temporarily or permanently, below the receive value causes a socket create call to fail.

To change the limit temporarily, use the following commands:

```
sysconfig -r atm sb_max=SIZE
```

Where SIZE is the number of bytes desired.

To make the change permanent, add the following lines to the end of the /etc/sysconfigtab file.

sockets:

```
     sb_max=SIZE
```

# 6. Examples

## *6.1 Example Program Descriptions*

There are 7 example programs included with kit version 1.10. The executable name and a brief description are given below.

api_probe - This application probes the system for all installed ATM adapters and dumps each adapter's configuration data. The usage is:

```
api_probe
```

api_xmt - The api_xmt application opens a UBR circuit and continuously transmits a packet of length 8192 bytes. The usage is:

```
api_xmt   driver_name   vpi   vci
```

api_xmt_pace - This application is very similar to api_xmt, except that it paces the UBR circuit. Note that the ATM driver/adapter must support UBR pacing for this application to function correctly. The usage is:

```
api_xmt_pace   driver_name   vpi   vci   kbps
```

api_xmt_cbr - The api_xmt_cbr function open a CBR circuit, requests a percentage of the available bandwidth, and continuously transmits a packet with a programmable size. Note that the final zero is a reserved argument. The usage is:

```
api_xmt_cbr   driver_name   vpi   vci   packet_size   kbps   0
```

api_rcv - The api_rcv application is used in conjunction with the api_xmt application. That is, the api_rcv application receives traffic from the api_xmt application when both applications are using the same vpi, vci combination. The usage is:

```
api_rcv   driver_name   vpi   vci
```

api_rcv_pace - The api_rcv_pace application is used in conjunction with the api_xmt_pace application. That is, the api_rcv_pace application receives traffic from the api_xmt application when both applications are using the same vpi, vci combination. The usage is:

api_rcv_pace   driver_name   vpi   vci

api_rcv_cbr - The api_rcv application is used in conjunction with the api_xmt application. That is, the api_rcv application receives traffic from the api_xmt application when both applications are using the same vpi, vci combination. The usage is:

api_rcv_cbr   driver_name   vpi   vci   packet_size   kbps   0

Makefile - All executables are built with this file. For example, make api_xmt builds the api_xmt application from the xmt.c source.

## 7. Example Code

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <netaal/aal.h>
#include <sys/errno.h>


main ()
   {
   int                 sock;
   char                buff[100];
   char                driver[16];
   ATM_CONNECTION_ID   conn_id;
   ATM_PARAMS          atm_params;
   ATM_DEVICE_INFO     device_info;
   unsigned int        device_count;

   printf ("[ Creating Client Socket ]\n");

   sock = socket (AF_AAL, SOCK_DGRAM, 0);
   if (sock == -1)
      {
      printf ("[ socket call failed, error: %d ]\n", errno);
      exit (1);
      }

   ioctl (sock, SIO_GET_ATM_DEVICE_COUNT, &device_count);
   printf ("[ There are %d ATM Interfaces in the System ]\n",
           device_count);
   if (device_count == 0)
      exit (1);

   device_info.device_number   = 0;
   device_info.driver_name     = driver;
   device_info.driver_name_len = 16;
   if (ioctl (sock, SIO_GET_DEVICE_INFO, &device_info) == -1)
      {
      printf ("[ ioctl call failed, error: %d ]\n", errno);
      exit (1);
      }

   printf ("[ Driver Name is ’%s’ ]\n", driver);

   printf ("[ Assigning ATM Parameters (optional)]\n");

   /* Initialize atm_info to all default values. */
   ATM_DEFAULTS (&atm_params);

   /* Modify atm_info to describe a CBR connection at 64Kbits/sec */
   atm_params.fqos        = ATM_QC_CLASSA;
   atm_params.bqos        = ATM_QC_CLASSA;
   atm_params.valid_rates = ATM_VR_FPEAK1 | ATM_VR_BPEAK1;
   atm_params.fpeakcr[1]  = 167;   /* 64000 bits/sec */
   atm_params.bpeakcr[1]  = 167;
   atm_params.flags       = ATM_FLAG_CBR;
```

```
if (ioctl (sock, SIO_SET_ATM_PARAMETERS, &atm_params) == -1)
   {
   printf ("[ ioctl call failed, error: %d ]\n", errno);
   exit (1);
   }

printf ("[ Associating PVC ]\n");

conn_id.device_number = 0;
conn_id.vpi           = 0;
conn_id.vci           = 1000;

if (ioctl (sock, SIO_ASSOCIATE_PVC, &conn_id) == -1)
   {
   printf ("[ ioctl call failed, error: %d ]\n", errno);
   exit (1);
   }

printf ("[ Sending Text Message ]\n");

strcpy (buff, "Socket Message - Via ATM Datagram Communication");
send (sock, buff, strlen (buff), 0);

printf ("[ Awaiting Reply ]\n");

while (recv (sock, buff, 100, 0) < 0);

printf ("[ Message Received: '%s' ]\n", buff);
}
```