

SunATM-155 SBus Cards Manual



Sun Microsystems Computer Company
A Sun Microsystems, Inc. Business
2550 Garcia Avenue
Mountain View, CA 94043 U.S.A.
415 960-1300 FAX 415 969-9131
Part No.: 801-6572-11
Revision A, May 1995

© 1995 Sun Microsystems, Inc.
2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A.

All rights reserved. This product and related documentation are protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or related documentation may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Portions of this product may be derived from the UNIX[®] and Berkeley 4.3 BSD systems, licensed from UNIX System Laboratories, Inc., a wholly owned subsidiary of Novell, Inc., and the University of California, respectively. Third-party font software in this product is protected by copyright and licensed from Sun's font suppliers.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the United States Government is subject to the restrictions set forth in DFARS 252.227-7013 (c)(1)(ii) and FAR 52.227-19.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

TRADEMARKS

Sun, the Sun logo, Sun Microsystems, Sun-4, SunATM, SunDiag, SunView, OpenBoot, AnswerBook, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and certain other countries. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. OPEN LOOK is a registered trademark of Novell, Inc. PostScript and Display PostScript are trademarks of Adobe Systems, Inc. All other product names mentioned herein are the trademarks of their respective owners.

All SPARC trademarks, including the SCD Compliant Logo, are trademarks or registered trademarks of SPARC International, Inc. SPARCstation, SPARCserver, SPARCengine, SPARCstorage, SPARCware, SPARCcenter, SPARCclassic, SPARCcluster, SPARCdesign, SPARC811, SPARCprinter, UltraSPARC, microSPARC, SPARCworks, and SPARCcompiler are licensed exclusively to Sun Microsystems, Inc. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK[®] and Sun[™] Graphical User Interfaces were developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a nonexclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

X Window System is a product of the Massachusetts Institute of Technology.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.



Your SBus card is marked to indicate its FCC, DOC, and VCCI class. Please read the appropriate section that corresponds to the marking on your SBus card before attempting to install it into your system.

FCC Class A Notice — United States

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions:

1. This device may not cause harmful interference.
2. This device must accept any interference received, including interference that may cause undesired operation.

Note – This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

Modifications

Modifications to this device, not approved by Sun Microsystems, Inc., may void the authority granted to the user by the FCC to operate this equipment.

FCC Class B Notice — United States

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions:

1. This device may not cause harmful interference.
2. This device must accept any interference received, including interference that may cause undesired operation.

Note – This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/television technician for help.

Modifications

Modifications to this device, not approved by Sun Microsystems, Inc., may void the authority granted to the user by the FCC to operate this equipment.

DOC Class A Notice — Canada

This digital apparatus does not exceed the Class A limits for radio noise emission for a digital apparatus as set out in the Radio Interference Regulations of the Canadian Department of Communications.

Avis Concernant les Systèmes Appartenant à la Classe A du DOC — Canada

Le présent appareil numérique n'émet pas de bruits radioélectriques dépassant les limites applicables aux appareils numériques de la classe A prescrites dans le Règlement sur le brouillage radioélectrique édicté par le ministère des Communications du Canada.

DOC Class B Notice — Canada

This digital apparatus does not exceed the Class B limits for radio noise emission for a digital apparatus as set out in the Radio Interference Regulations of the Canadian Department of Communications.

Avis Concernant les Systèmes Appartenant à la Classe B du DOC — Canada

Le présent appareil numérique n'émet pas de bruits radioélectriques dépassant les limites applicables aux appareils numériques de la classe B prescrites dans le Règlement sur le brouillage radioélectrique édicté par le ministère des Communications du Canada.

Nippon—Japan

第一種VCCI基準に関するお知らせ

この装置は、第一種情報装置(商工業地域において使用されるべき情報装置)で商工業地域での電波障害防止を目的とした情報処理装置等電波障害自主規制協議会(VCCI)基準に適合しております。

この装置は、第一種または第二種ワークステーションのオプションです。本装置を使用する場合、システムとしての適合レベルは下記の通りです。

第一種ワークステーション：第一種情報装置

第二種ワークステーション：第一種情報装置

本装置を使用する第二種ワークステーションは、第一種情報装置(商工業地域において使用されるべき情報装置)となります。

従って、住宅地域またはその隣接した地域で使用すると、ラジオ、テレビジョン受信機等に受信障害を与えることがあります。

取扱説明書に従って正しい取り扱いをして下さい。

第二種VCCI基準に関するお知らせ

この装置は、第二種情報装置（住宅地域またはその隣接した地域において使用されるべき情報装置）で住宅地域での電波障害防止を目的とした情報処理装置等電波障害自主規制協議会（VCCI）基準に適合しております。

この装置は、第一種または第二種ワークステーションのオプションです。本装置を使用する場合、システムとしての適合レベルは下記の通りです。

第一種ワークステーション：第一種情報装置

第二種ワークステーション：第二種情報装置

本装置を使用する第一種ワークステーションは、第一種情報装置（商工業地域において使用されるべき情報処理装置）となります。従って、住宅地域またはその隣接した地域で使用すると、ラジオ、テレビジョン受信機等に受信障害を与えることがあります。

本装置を使用する第二種ワークステーションは、第二種情報装置（住宅地域またはその隣接地域において使用されるべき情報装置）となります。従って、本装置をラジオ、テレビジョン受信機に近接してご使用になると、受信障害の原因となることがあります。

取扱説明書に従って正しい取り扱いをして下さい。

Contents

1. Introducing the SunATM-155 SBus Cards	1-1
1.1 Hardware Requirements	1-5
1.2 Software Requirements	1-5
2. Installing SunATM-155 Hardware	2-1
2.1 Attaching the Wrist Strap	2-1
2.2 SBus Card Installation	2-2
2.2.1 SunATM-155/MFiber	2-4
2.2.2 SunATM-155/UTP5	2-6
2.3 Testing the SunATM-155 Card Before Booting the System	2-8
3. Installing SunATM-155 Software and Configuring the ATM Interface	3-1
3.1 Installing and Removing Software	3-2
3.1.1 Command Line Utilities	3-2
3.1.2 Graphical User Interface	3-6
3.2 SBus ATM Interface Configuration	3-15
3.2.1 Changes to System Configuration	3-16

3.3	Rebooting the System and Examining Network Interfaces	3-35
A.	Wiring Scheme and Pin Descriptions	A-1
B.	SunATM-155 SBus Cards Specifications	B-1
B.1	Performance Specifications	B-1
B.2	Power Specifications	B-2
B.3	Physical Dimensions	B-2
B.4	Environmental Specifications	B-3
C.	Running Diagnostic Tests	C-1
C.1	Selftest	C-1
C.1.1	Setting the Diag-switch	C-4
C.1.2	Running Selftest	C-4
C.2	SunDiag	C-5
C.2.1	SunDiag Window	C-5
C.2.2	Starting the Test	C-6
D.	Application Programmers' Interface	D-1
D.1	Q.93B API	D-2
D.1.1	Q.93B Driver	D-4
D.1.2	Q.93B User Space API	D-7
D.1.3	Q.93B Kernel Space API	D-25
D.2	Driver API	D-38
E.	Advanced Configurations	E-1
E.1	Flags That Specify Additional Entry Types	E-1
E.2	Flags That Change the Behavior of the Interface	E-3

Figures

Figure 1-1	SunATM-155/MFiber SBus Card and Back Panel	1-3
Figure 1-2	SunATM-155/UTP SBus Card and Back Panel.	1-4
Figure 2-1	Wrapping the Wrist Strap Around Your Wrist.	2-2
Figure 2-2	Handling the SunATM-155 Card	2-2
Figure 2-3	SunATM-155/MFiber Card with Back Panel	2-4
Figure 2-4	SunATM-155/UTP5 Card with Back Panel.	2-6
Figure 3-1	ATM Address Fields.	3-33
Figure A-1	Designation T568B.	A-1
Figure C-1	SunDiag Window	C-6
Figure D-1	ATM Signaling.	D-2
Figure D-2	Message Format	D-3

Tables

Table 1-1	Platform Architecture with Examples of Systems	1-2
Table 3-1	Acronyms and Abbreviations	3-15
Table 3-2	<code>/etc/aarconfig</code> File Flags.	3-22
Table 3-3	Predefined SunATM Variables	3-24
Table A-1	Pin Descriptions for the 96-Pin SBus Connector.	A-2
Table B-1	Performance Specifications	B-1
Table B-2	Power Specifications	B-2
Table B-3	Physical Dimensions	B-2
Table B-4	Environmental Specifications	B-3
Table D-1	Message Meanings	D-3
Table D-2	Messages Between the User and the Q.93B Driver.	D-4
Table E-1	<code>/etc/aarconfig</code> Advanced Configuration Flags.	E-3

Preface

This manual provides information about the SunATM™-155 SBus cards. The manual is organized into three chapters and five appendixes.

When You Need Help with UNIX Commands

This manual may not include specific software commands or procedures. Instead, it names software tasks and refers you to operating system documentation or the handbook that was shipped with your new hardware.

The type of information that you might need to use references for includes:

- Shutting down the system
- Booting the system
- Configuring devices
- Other basic software procedures

Refer to one or more of the following:

- *Solaris 2.x Handbook for SMCC Peripherals* contains Solaris® 2.x software commands.
- On-line *AnswerBook*® for the complete set of documentation supporting the Solaris 2.x software environment.
- Other software documentation that you received with your system.

What Typographic Changes Mean

The following table describes the typographic changes used in this book.

Table P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. machine_name% You have mail.
AaBbCc123	What you type, contrasted with on-screen computer output	machine_name% su Password:
<i>AaBbCc123</i>	Command-line placeholder: replace with a real name or value	To delete a file, type <code>rm filename</code> .
<i>AaBbCc123</i>	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be root to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

Table P-2 Shell Prompts

Shell	Prompt
C shell prompt	machine_name%
C shell superuser prompt	machine_name#
Bourne shell and Korn shell prompt	\$
Bourne shell and Korn shell superuser prompt	#

Notes, Cautions, and Warnings



Warning – This equipment contains lethal voltage. Accidental contact can result in serious injury or death.



Caution – Improper handling by unqualified personnel can cause serious damage to this equipment. Unqualified personnel who tamper with this equipment may be held liable for any resultant damage to the equipment.

Individuals who remove any outer panels to access this equipment must observe all safety precautions and ensure compliance with skill level requirements, certification, and all applicable local and national laws.

Procedures contained in this document must be performed by qualified service-trained maintenance providers.

Note – Before you begin, carefully read each of the procedures in this manual. If you have not performed similar operations on comparable equipment, *do not attempt* to perform these procedures.

Introducing the SunATM-155 SBus Cards



The SunATM-155/MFiber Adapter and SunATM-155/UTP Adapter are two single-wide SBus cards that conform to the specifications of the Asynchronous Transfer Mode (ATM) Forum. The cards offer 155 Mbps network bandwidth over either multimode fiber optic cable or category 5 unshielded twisted pair (UTP) copper wire.

SunATM-155 SBus cards (SunATM-155 cards) are designed for operation in systems that run under the Solaris environment, revision 2.4 or later, or other compatible operating systems. To utilize the SunATM-155 cards, the system also needs to contain OpenBoot™ PROM (OBP) level 2.0 or later. An on-board FCode PROM provides configuration support that identifies the SunATM-155 cards to the system and contains selftest routines.

SunATM-155 SBus cards highlights are:

- Conform to IEEE 1496, offering 32 bit SBus and 32 byte burst support
- Support 155 Mbps operation over:
 - 62.5/125 μ Multimode fiber or
 - UTP Category 5 wire
- Integrate SBus/SAR (Segmentation And Reassembly) ASIC SAHI2 (SBus to ATM Host Interface, version 2) implemented in standard CMOS
- SAR function aligned with ATM Forum specified and International Consultative Committee for Telegraph and Telephone (CCITT) approved AAL 5 (ATM Adaptation Layer)

- Support SONET/SDH (Synchronous Optical NETWORK/Synchronous Digital Hierarchy) physical layer framing structure
- Up to 135 simultaneous transmit channels and up to 1024 simultaneous open receive channels
- Compatible with relevant emerging standards (including existing ATM Forum baseline specifications and CCITT)

Note – Level 2.x boot PROMs (or later) are *required* for systems using the SunATM-155 cards. If lower-level boot PROMs are installed on your system, you must upgrade the boot PROMs before using SunATM-155 cards.

To find the OBP revision level on your system, type `.version` at the `<#0> ok` prompt.

Table 1-1 shows the SBus based Sun-4m and Sun-4c architecture systems that support SunATM-155 SBus cards when running under the Solaris environment, revision 2.4 or later, or other compatible operating systems.

Table 1-1 Platform Architecture with Examples of Systems

Platform Architecture	System Type
Sun-4m	SPARCstation™ 4
	SPARCstation 5
	SPARCstation 10
	SPARCstation 20
	SPARCstation 600 Series
Sun-4d	SPARCserver™ 1000
	SPARCcenter™ 2000
Sun-4c	SPARCstation 2
	SPARCstation IPX

Figure 1-1 shows the SunATM-155/MFiber Adapter SBus card and Figure 1-2 shows the SunATM-155/UTP Adapter SBus card.

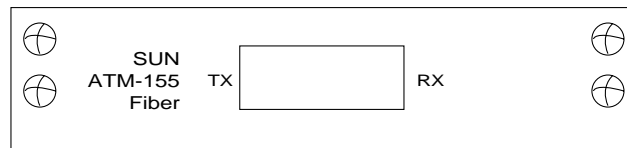
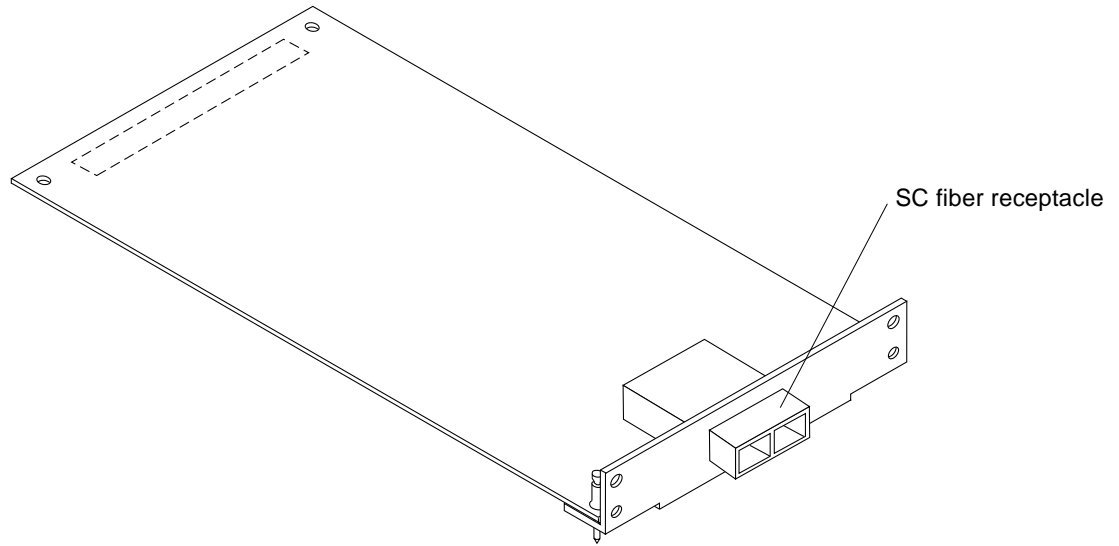


Figure 1-1 SunATM-155/MFiber SBus Card and Back Panel

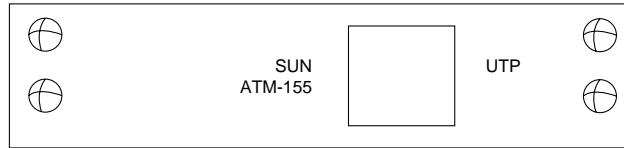
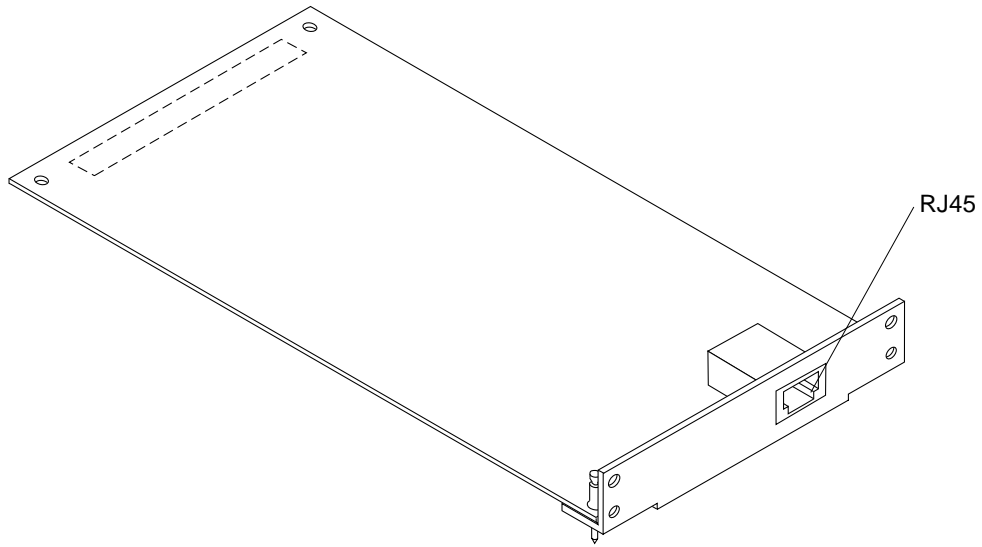


Figure 1-2 SunATM-155/UTP SBus Card and Back Panel

1.1 Hardware Requirements

You need an ATM switch to build an ATM network. To connect the SunATM-155 cards to the ATM switch, you need the following cables:

- SunATM-155/MFiber Adapter - Multimode fiber cable with SC connector
- SunATM-155/UTP Adapter - Category 5 UTP with RJ45 connector

Only one SunATM-155 card is supported per SBus. For example, on desktop machines that have only one SBus per system (even though there may be multiple SBus slots), only one SunATM-155 card is supported per system.

Refer to the manual supplied with the ATM switch for specific instructions about cable connections.

1.2 Software Requirements

Note – Install SunATM-155 hardware first. Then install the software and configure the ATM interface.

SunATM-155 SBus cards are supported on systems running under the Solaris environment, revision 2.4 or later, or other compatible operating systems.

The SunATM CD-ROM that shipped with the SBus card contains *required* driver software that must be installed in order to connect a SunATM-155 SBus card to a network.

Installing SunATM-155 Hardware



Install the SunATM-155 hardware first, before you install the software.

Note – SunATM-155 cards are supported on systems running the Solaris software environment, revision 2.4 or later, or other compatible operating systems.

Only one SunATM-155 card is supported per SBus. For example, on desktop machines that have only one SBus per system (even though there may be multiple SBus slots), only one SunATM-155 card is supported per system.

2.1 Attaching the Wrist Strap

The wrist strap provides grounding for static electricity between your body and the system unit chassis. When used properly, the wrist strap keeps static electricity from building up on your hands.



Caution – If you do not wear a wrist strap, the system components can be damaged by harmful electrical discharge.

- 1. Wrap the wrist strap around your wrist twice with the conductive adhesive tape against your skin.**



Figure 2-1 Wrapping the Wrist Strap Around Your Wrist

2. Peel the liner from the copper foil at the opposite end of the wrist strap.
3. Attach the copper end of the wrist strap to the metal casing of the power supply in the system unit.

2.2 SBus Card Installation

1. Remove the SunATM-155 card from the antistatic bag, spread the bag on a firm surface to provide a protective mat, and place the SunATM-155 card on the bag.

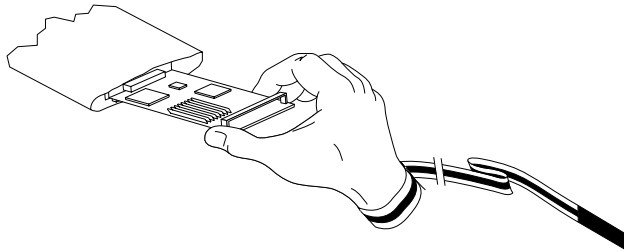


Figure 2-2 Handling the SunATM-155 Card

Note – The SunATM-155/MFiber card is shipped with a rubber plug that keeps the connector free of dust. To install the card, the plug must be removed.

2. Install the SunATM-155 card according to the SBus installation procedures in the hardware installation or service manual for your system.

Note – Do not boot the system until SunATM-155 installation is verified. See “Testing the SunATM-155 Card Before Booting the System” on page 2-8.



Caution – Do not change the SBus slot in which a SunATM-155 card is installed once the system has been booted. The Solaris 2.x software environment remembers the location of each SBus card that has been installed. Switching SBus slots will cause the operating system to assume that you removed your original SunATM-155 card and added a second card to the system. Refer to the online man page about `path_to_inst` for more information.

3. Verify SunATM-155 installation by executing a test command.

See “Testing the SunATM-155 Card Before Booting the System” on page 2-8.

Sections 2.2.1 and 2.2.2 provide an introduction to the physical connectors and wiring characteristics of the SunATM-155/MFiber and SunATM-155/UTP adapters, respectively.

2.2.1 SunATM-155/MFiber

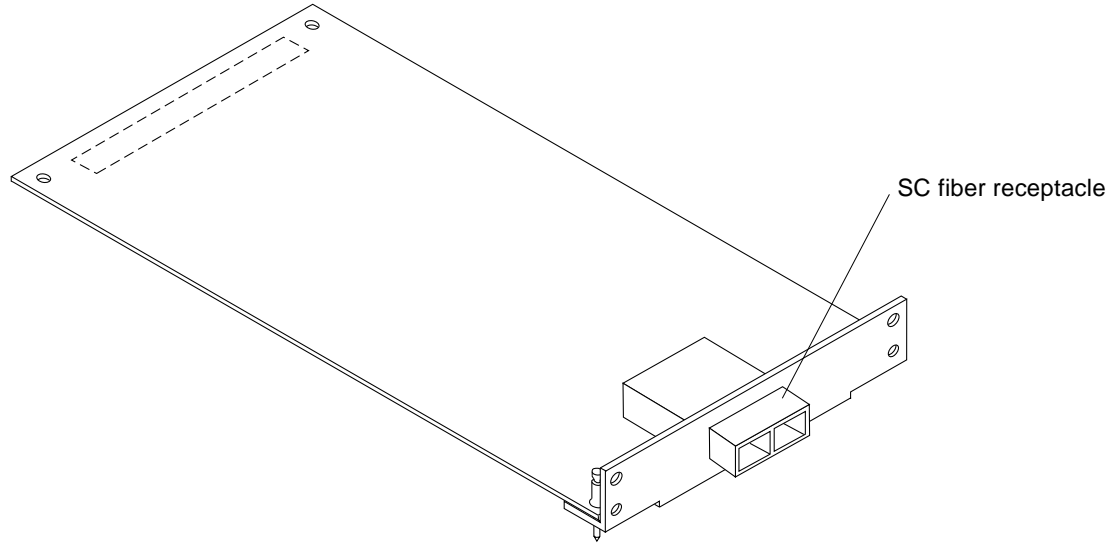


Figure 2-3 SunATM-155/MFiber Card with Back Panel

2.2.1.1 Extender Plate

A sheet metal extender plate is attached to the SunATM-155 SBus card. You must use the extender plate to install this SBus card correctly in some systems. Refer to the hardware installation or service manual that shipped with your system for information about installing SBus cards.

Note – The SunATM-155/MFiber card is shipped with a rubber plug that keeps the connector free of dust. To install the card, the plug must be removed.

2.2.1.2 Wiring Configuration

The SunATM-155/MFiber SBus card is shipped with the SC connector already keyed. As you hold the SBus card with the connector pointed toward you, “transmit” is on the left and “receive” is on the right.

2.2.1.3 Connecting the SBus Card to the Network

- ◆ **Connect one end of the multimode fiber cable into the fiber receptacle on the SBus card and connect the other end to the ATM switch.**

Refer to the installation or users manual supplied with the hardware interface for additional information.

2.2.2 SunATM-155/UTP5

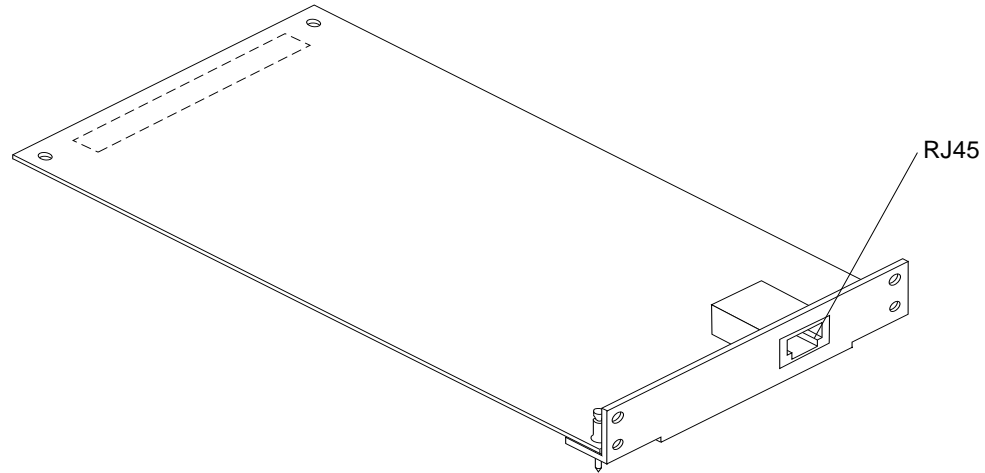


Figure 2-4 SunATM-155/UTP5 Card with Back Panel

2.2.2.1 Extender Plate

A sheet metal extender plate is attached to the SunATM-155 SBus card. You must use the extender plate to install this SBus card correctly in some systems. Refer to the hardware installation or service manual that shipped with your system for information about installing SBus cards.

2.2.2.2 Wiring Configuration

The SunATM-155/UTP SBus card is shipped with the RJ45 connector already keyed for “transmit” (Pair 2, pins 1 and 2) and “receive” (Pair 4, pins 7 and 8) in accordance with the EIA/TIA (T568B) wiring scheme.

2.2.2.3 Connecting the SBus Card to the Network

- ◆ **Plug one end of the Category 5 UTP network cable into the RJ45 receptacle on the SBus card and connect the other end to the ATM switch.**

Refer to the installation or users manual supplied with the hardware interface for additional information.

2.3 Testing the SunATM-155 Card Before Booting the System

After you install the SunATM-155 card, *before booting the system*, verify installation by executing the `show-devs` and `test` commands.

1. Use `show-devs` to find out SBus card information.

The `show-devs [device path]` command displays all devices known to the system directly beneath a given level in the device hierarchy. The `show-devs` command used by itself shows the entire device tree. Examples below show information for a SPARCstation 10 system.

Note – The `/sa@3,0` entry in the lines of text circled indicates the system recognizes the SunATM-155 card plugged into SBus slot 3.

```
<#0> ok show-devs /iommu/sbus
/iommu@f,e0000000/sbus@f,e0001000/sa@3,0
/iommu@f,e0000000/sbus@f,e0001000/SUNW,DBRIe@f,8010000
/iommu@f,e0000000/sbus@f,e0001000/SUNW,bpp@f,4800000
/iommu@f,e0000000/sbus@f,e0001000/ledma@f,400010
/iommu@f,e0000000/sbus@f,e0001000/espdma@f,400000
/iommu@f,e0000000/sbus@f,e0001000/SUNW,DBRIe@f,8010000/mmcodec
/iommu@f,e0000000/sbus@f,e0001000/ledma@f,400010/le@f,c00000
/iommu@f,e0000000/sbus@f,e0001000/espdma@f,400000/esp@f,800000
/iommu@f,e0000000/sbus@f,e0001000/espdma@f,400000/esp@f,800000/st
/iommu@f,e0000000/sbus@f,e0001000/espdma@f,400000/esp@f,800000/sd
<#0> ok
```

```
<#0> ok show-devs
/TI,TMS390Z50@f,f8ffffffc
/eccmemctl@f,0
/virtual-memory@0,0
/memory@0,0
/obio
/iommu@f,e0000000
/openprom
/aliases
/options
/packages
/obio/power@0,a01000
/obio/auxio@0,800000
/obio/SUNW,fdtwo@0,700000
/obio/interrupt@0,400000
/obio/counter@0,300000
/obio/EEPROM@0,200000
/obio/zs@0,0
/obio/zs@0,100000
/iommu@f,e0000000/sbus@f,e0001000
/iommu@f,e0000000/sbus@f,e0001000/sa@3,0
/iommu@f,e0000000/sbus@f,e0001000/SUNW,DBRIe@f,8010000
/iommu@f,e0000000/sbus@f,e0001000/SUNW,bpp@f,4800000
/iommu@f,e0000000/sbus@f,e0001000/ledma@f,400010
/iommu@f,e0000000/sbus@f,e0001000/espdma@f,400000
/iommu@f,e0000000/sbus@f,e0001000/SUNW,DBRIe@f,8010000/mmcodec
/iommu@f,e0000000/sbus@f,e0001000/ledma@f,400010/le@f,c00000
/iommu@f,e0000000/sbus@f,e0001000/espdma@f,400000/esp@f,800000
/iommu@f,e0000000/sbus@f,e0001000/espdma@f,400000/esp@f,800000/st
/iommu@f,e0000000/sbus@f,e0001000/espdma@f,400000/esp@f,800000/sd
/packages/obp-tftp
/packages/deblocker
/packages/disk-label
<#0> ok
```

2. Set the `diag-switch` to “true” to test the ATM SBus hardware and the connection to the hub:

```
ok setenv diag-switch? true
```

3. Use a `test` command to test the specified “sa” device, then check the output messages for any errors.

The example refers to a card installed in slot 3 of a SPARCstation 10 system.

```
ok test /iommu@f,e0000000/sbus@f,e0001000/sa@3,0
```

Note – If the `test` command fails, verify that the SBus card hardware is installed correctly. If necessary, replace the SBus card and/or contact your service provider.

4. Set the `diag-switch` to “false” to turn off diag:

```
ok setenv diag-switch? false
```

5. Boot the system.

Installing SunATM-155 Software and Configuring the ATM Interface



Install the SunATM-155 hardware first; then install the software and configure the ATM interface.

Before beginning the software installation, you need the *IP hostname(s)* and the *IP addresses* for your ATM interface(s). You will be prompted to enter these during software installation.

ATM does not currently support broadcast, therefore, `ypbind` with the `-broadcast` option cannot be used to automatically locate the ypservers on the ATM subnet.

If you are planning to run `nis` over ATM, you must specify the list of ypservers via the `ypinit -c`. See `ypinit` for details of setting up the ypservers. It is important to note that the IP addresses of the ypservers must be available in the `/etc/hosts` file.

Since ATM does not currently support multicast, hosts cannot use `in.rdisc` to locate routers on the ATM subnet. `in.rdisc` uses IP multicasting to automatically locate routers and pick the best router among many. Hosts cannot use RIP (`in.routed`) since RIP uses broadcast. Routes to the routers in the ATM subnet have to be exclusively added. You may also specify one router as a default router to provide connectivity outside of the ATM subnet. See details of the `route` command to add specific router entries and to add a default router.

3.1 *Installing and Removing Software*

Note – SunATM-155 cards are supported on systems running the Solaris software environment, revision 2.4 or later (or other compatible operating systems).

Under the Solaris 2.4 environment, all unbundled software is delivered as “packages,” and can be installed by using either:

- Command line utilities, or
- Graphical user interface program (swmtool(1))

You must use one of these methods to install the required driver software using the SunATM CD-ROM that came with the SBus card.

Note – The SunATM Device Drivers (SUNWatm) and SunATM Runtime Support Software (SUNWatmu) packages are required for SunATM-155 cards.

3.1.1 *Command Line Utilities*

The primary commands used to add, remove, or check any packages are:

pkgadd(1M)	to add packages
pkgrm(1M)	to remove packages
pkgchk(1M) and pkginfo(1M)	to check installation

3.1.1.1 *Adding Software Packages Using pkgadd*

1. **Become superuser:**

```
% su
Password:
#
```

2. Before adding a package, insert the CD-ROM in its caddy and mount the CD-ROM:

```
# mkdir /cdrom
```

The volume manager should mount the CD on the directory:

```
/cdrom/sunatm_1_0
```

The last directory in this path is determined by information on the CD.

3. To add software packages named SUNWatm, SUNWatmu, and SUNWatma, after becoming superuser, enter:

```
# /usr/sbin/pkgadd -d /cdrom/sunatm_1_0\  
SUNWatm SUNWatmu SUNWatma
```

Multiple packages can be added by separating package names with a space.

Note – The SunATM Device Drivers (SUNWatm) and SunATM Runtime Support Software (SUNWatmu) packages are required for SunATM-155 SBus cards. SunATM Interim API Support Software (SUNWatma) is only needed if you want to use the Application Programmers' Interface (API).

4. Respond to the questions about your system configuration, as prompted, during package installation:

- Is the hardware installed [y,n,?,q]

Answer “y” for yes. Install the SunATM-155 SBus card before beginning the software installation.

- How many SBus ATM (sa) interfaces do you wish to install [0-14,?,q]

Enter the number of SBus cards being installed.

- What host name do you wish to use for sa0:

Enter the IP host name you wish to use for the ATM interface.

- What ip address do you wish to use for (host name):

Enter the IP address to be assigned to the ATM interface.

- Do you want to continue with installation of this package [y,n,?]

Enter “y” for yes to proceed with the installation.

```
Is the hardware installed [y,n,?,q] y
How many SBus ATM (sa) interfaces do you wish to install [0-14,?,q] 1
What host name do you wish to use for sa0: chances
What ip address do you wish to use for chances [129.146.101.94]
## Processing package information.
## Processing system information.
## 5 package pathnames are already properly installed.
## Verifying package dependencies.
## Verifying disk space requirements.
## Checking for conflicts with packages already installed.
## Checking for setuid/setgid programs.

This package contains scripts that will be executed with super-user
permission during the process of installing this package.

Do you want to continue with the installation of this package [y,n,?] y
```

The specified packages will be installed as follows:

- SunATM Device Drivers (SUNWatm) will go into /kernel/drv
- SunATM Runtime Support Software (SUNWatmu) will go into /opt/SUNWatm/bin
- SunATM Interim API (SUNWatma) will go into /usr/include/atm and /usr/lib

Note – Man pages contained in the SUNWatmu package will go into /opt/SUNWatm/man (Add this path to the MANPATH environment variable.) Interim API examples will go into /opt/SUNWatm/examples.

When the device on which the package resides is not specified, `pkgadd` checks the default spool directory (`/var/spool/pkg`). If the package is not there, installation fails. The `-d` option allows you to specify a different spool directory, and the name specified after `-d` must be a full pathname to a device or directory (as shown in the examples).

When `pkgadd` encounters a problem, information about the problem is displayed with the following prompt:

```
Do you want to continue with this installation?
```

You should respond with either `yes`, `no`, or `quit`. If more than one package has been specified, `no` stops the installation of the package being installed but informs `pkgadd` to continue with installation of the other packages. `quit` tells `pkgadd` to stop installation of all packages.

3.1.1.2 *Checking Installation of a Package Using `pkgchk`*

Once the package is installed, you can use the `pkgchk` command to check the installation completeness:

```
# /usr/sbin/pkgchk SUNWatm
```

Multiple packages can be specified at the command line by separating the package names with a space. If no package identifier is specified, the entire contents of the machine are checked.

3.1.1.3 *Checking ATM Software Installation Using `pkginfo`*

Check the ATM software installation by using the `pkginfo` command:

```
# /usr/bin/pkginfo | grep SUNWatm
```

3.1.1.4 *Removing Software Packages Using `pkgrm`*

You can remove one or more packages with the following command:

```
# /usr/sbin/pkgrm SUNWatm SUNWatma SUNWatmu
```

In this example, `pkgrm` removes the packages identified as `SUNWatm` (SunATM Device Drivers), `SUNWatma` (SunATM Interim API Support Software), and `SUNWatmu` (SunATM Runtime Support Software).

3.1.2 *Graphical User Interface*

Software Manager, a Solaris 2.4 OPEN LOOK application, installs and removes software packages. For additional information about using Software Manager, refer to relevant installation documentation for the Solaris release you are using.

3.1.2.1 *Adding Packages By Using Software Manager*

- 1. Become superuser, insert the CD-ROM in its caddy, and mount the CD-ROM:**

```
% su
Password:
# mkdir /cdrom
```

The volume manager should mount the CD on the directory:

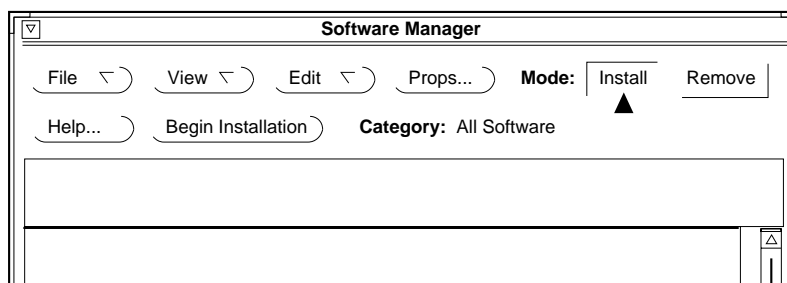
```
/cdrom/sunatm_1_0
```

The last directory in this path is determined by information on the CD.

- 2. Run the Software Manager application on your system.**

```
% su
Password:
#/usr/sbin/swmtool &
```

3. Select “Install” on the Software Manager window that is displayed.



4. Click on “Props...”. In the “Properties” window that pops up:

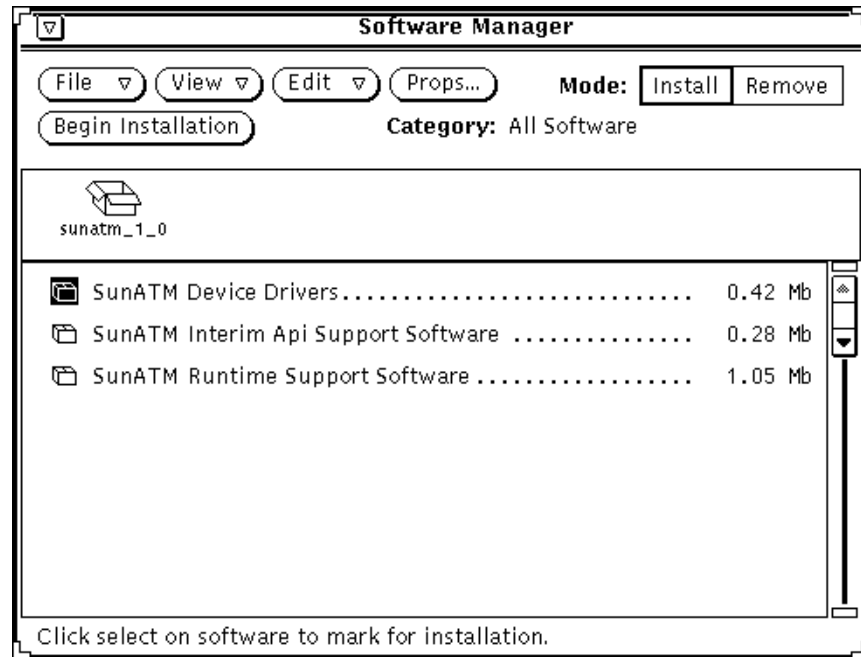
- a. **Select** Category: Source Media
- b. **Select** Media Type: Mounted Directory
- c. **Enter for** Directory Name: `/cdrom/sunatm_1_0`
- d. **Press the** Apply button.

After you press the Apply button, the SunATM-155 package icons should appear in the Software Manager window.

5. Select the software packages to be installed by clicking on the icons.

The package being installed in the example that follows is *SunATM Device Drivers*.

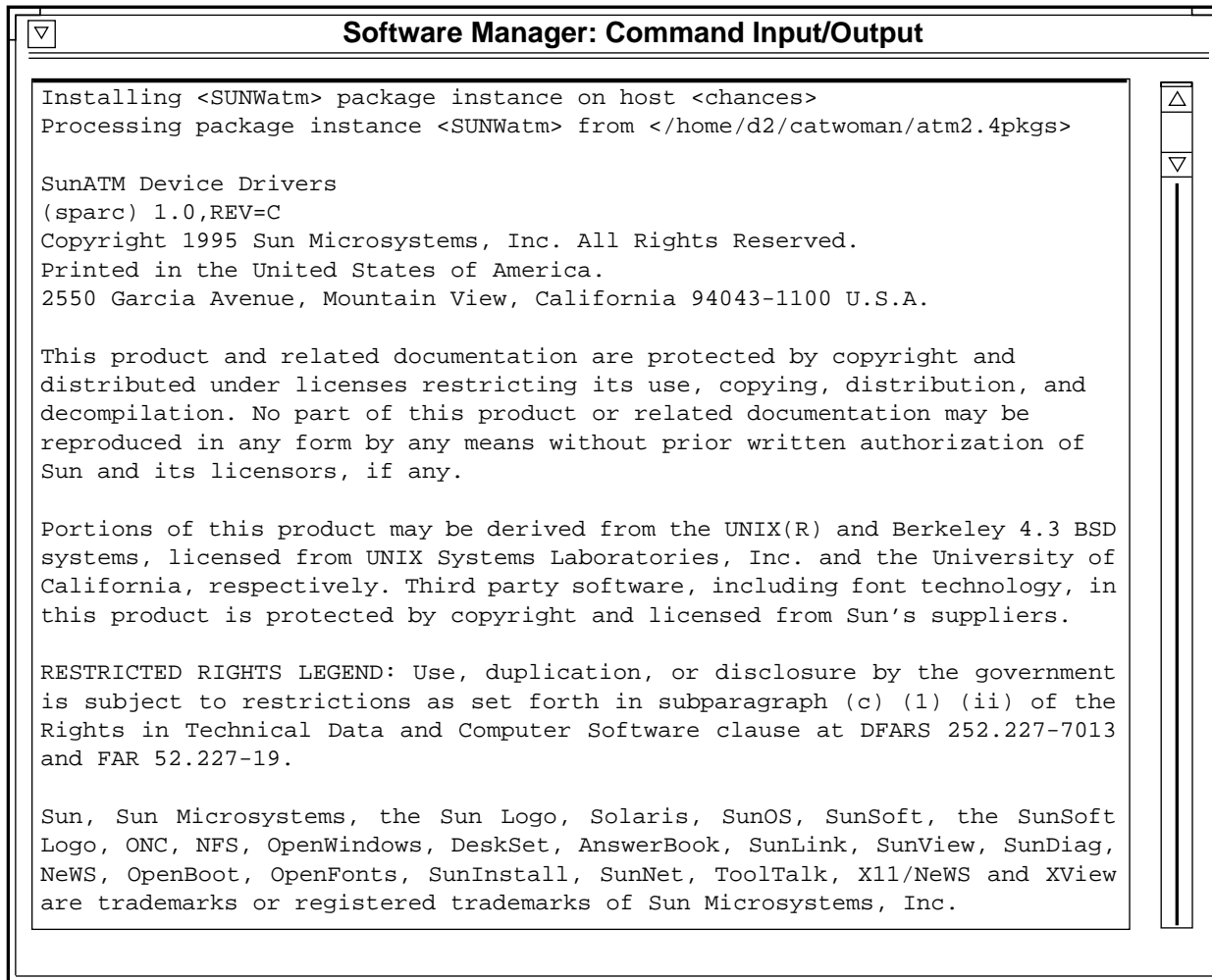
Note – The SunATM Device Drivers (SUNWatm) and SunATM Runtime Support Software (SUNWatmu) packages are required for SunATM-155 SBus cards. SunATM Interim API Support Software (SUNWatma) is only needed if you want to use the Application Programmers’ Interface (API).



Note – Select only the software packages to be installed at this time, and make sure all other packages are deselected. To deselect a package, click once on its icon. To deselect all the packages, choose “Deselect All” from the Edit menu.

6. Press the “Begin Installation” button.

The Software Manager: Command Input/Output window is displayed.



The screenshot shows a window titled "Software Manager: Command Input/Output". The text inside the window is as follows:

```
Installing <SUNWatm> package instance on host <chances>
Processing package instance <SUNWatm> from </home/d2/catwoman/atm2.4pkgs>

SunATM Device Drivers
(sparc) 1.0,REV=C
Copyright 1995 Sun Microsystems, Inc. All Rights Reserved.
Printed in the United States of America.
2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A.

This product and related documentation are protected by copyright and
distributed under licenses restricting its use, copying, distribution, and
decompilation. No part of this product or related documentation may be
reproduced in any form by any means without prior written authorization of
Sun and its licensors, if any.

Portions of this product may be derived from the UNIX(R) and Berkeley 4.3 BSD
systems, licensed from UNIX Systems Laboratories, Inc. and the University of
California, respectively. Third party software, including font technology, in
this product is protected by copyright and licensed from Sun's suppliers.

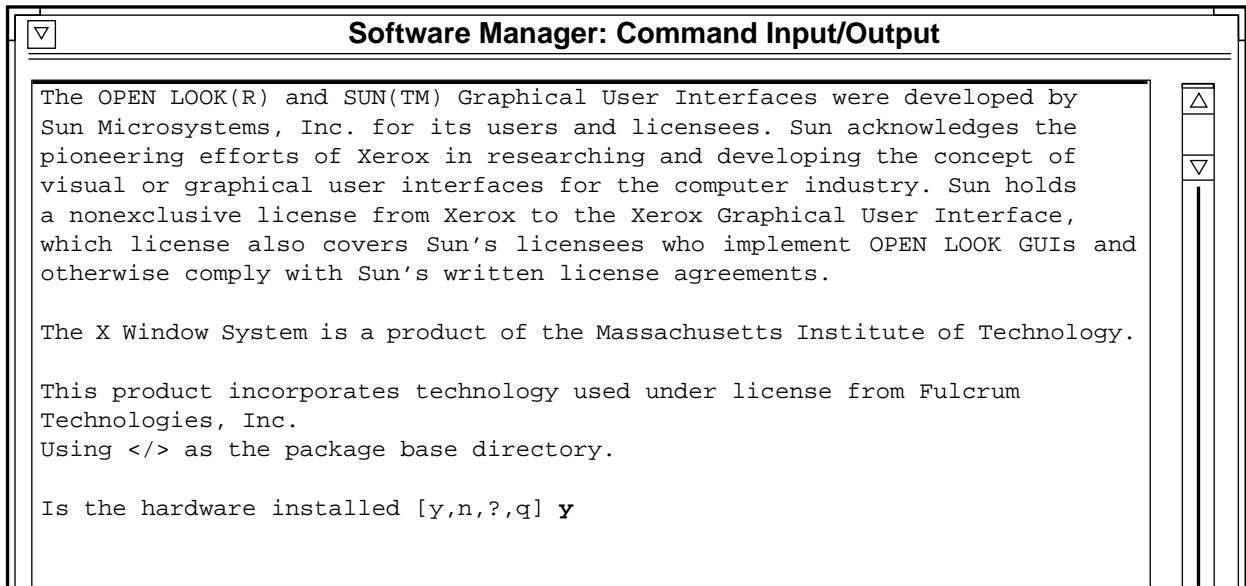
RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government
is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the
Rights in Technical Data and Computer Software clause at DFARS 252.227-7013
and FAR 52.227-19.

Sun, Sun Microsystems, the Sun Logo, Solaris, SunOS, SunSoft, the SunSoft
Logo, ONC, NFS, OpenWindows, DeskSet, AnswerBook, SunLink, SunView, SunDiag,
NeWS, OpenBoot, OpenFonts, SunInstall, SunNet, ToolTalk, X11/NeWS and XView
are trademarks or registered trademarks of Sun Microsystems, Inc.
```

7. Respond to the questions about your system configuration, as prompted by Software Manager, during package installation:

- Is the hardware installed [y,n,?,q]

Answer “y” for yes. Install the SunATM-155 SBus card before beginning the software installation.



- How many SBus ATM (sa) interfaces do you wish to install [0-14,?,q]
Enter the number of SBus cards being installed.
- What host name do you wish to use for sa0:
Enter the IP host name you wish to use for the ATM interface.
- What ip address do you wish to use for (host name):
Enter the IP address to be assigned to the ATM interface.
- Do you want to continue with installation of this package [y,n,?]
Enter “y” for yes to proceed with the installation.

```

Software Manager: Command Input/Output

The X Window System is a product of the Massachusetts Institute of Technology.

This product incorporates technology used under license from Fulcrum
Technologies, Inc.
Using </> as the package base directory.

Is the hardware installed [y,n,?,q] y

How many SBus ATM (sa) interfaces do you wish to install [0-14,?,q] 1

What host name do you wish to use for sa0: chances

What ip address do you wish to use for chances [129.146.101.94]
## Processing package information.
## Processing system information.
## 5 package pathnames are already properly installed.
## Verifying package dependencies.
## Verifying disk space requirements.
## Checking for conflicts with packages already installed.
## Checking for setuid/setgid programs.

This package contains scripts that will be executed with super-user permission
during the process of installing this package.

Do you want to continue with the installation of this package [y,n,?] y

```

The selected packages will be installed as follows:

- SunATM Device Drivers (SUNWatm) will go into /kernel/drv
- SunATM Runtime Support Software (SUNWatmu) will go into /opt/SUNWatm/bin
- SunATM Interim API (SUNWatma) will go into /usr/include/atm and /usr/lib

Note – Man pages contained in the SUNWatmu package will go into /opt/SUNWatm/man (Add this path to the MANPATH environment variable.)
 Interim API examples will go into /opt/SUNWatm/examples.

The screenshot shows a window titled "Software Manager: Command Input/Output". The window contains a text area with the following text:

```

Installing SunATM Device Drivers as <SUNWatm>

## Installing part 1 of 1.
/etc/aarconfig
/etc/rc2.d/S79atm
/kernel/drv/aar
/kernel/drv/aar.conf
/kernel/drv/atmip
/kernel/drv/q93b
/kernel/drv/q93b.conf
/kernel/drv/qcc
/kernel/drv/qcc.conf
/kernel/drv/sa
/kernel/drv/sscop
[ verifying class <base> ]
## Executing postinstall script.
postinstall configuration (sa):

Create /etc/hostatm.sa0

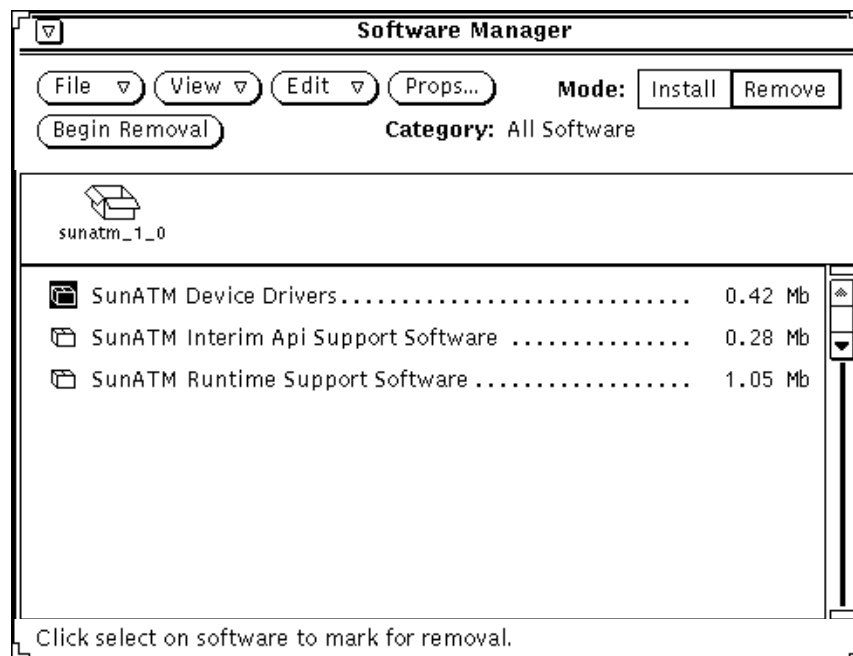
Please edit /etc/aarconfig to add IP-ATM address configuration.

Installation of <SUNWatm> was successful.
  
```

3.1.2.2 Removing Software Packages

Use the Software Manager “Remove” mode to remove software packages. For additional information, refer to relevant documentation for the Solaris release you are using.

- 1. Select the software package to be removed by clicking on the icon.**
Make sure all software that you *do not want removed is deselected*.
- 2. Press the “Begin Removal” button.**
The Software Manager: Command Input/Output window is displayed.
- 3. Respond to the questions, as prompted by Software Manager, during package removal.**



```

Software Manager: Command Input/Output

Removing <SUNWatm> package instance from host <chances>
The following package is currently installed:
  SUNWatm      SunATM Device Drivers
                (sparc) 1.0,REV=C
Do you want to remove this package [y,n,?,q] y
## Removing installed package instance <SUNWatm>
This package contains scripts that will be executed with super-user permission
during the process of removing this package.
Do you want to continue with the removal of this package [y,n,?,q] y
## Verifying package dependencies.
## Processing package information.
## Removing pathnames in class <base>
/sbin <shared pathname not removed>
/kernel/drv/sscop
/kernel/drv/sa
/kernel/drv/qcc.conf
/kernel/drv/qcc
/kernel/drv/q93b.conf
/kernel/drv/q93b
/kernel/drv/atmip
/kernel/drv/aar.conf
/kernel/drv/aar
/kernel/drv <shared pathname not removed>
/kernel <shared pathname not removed>
/etc/rc2.d/S79atm
/etc/rc2.d <shared pathname not removed>
/etc/aarconfig
/etc <shared pathname not removed>
## Executing postremove script.
## Updating system information.
Removal of <SUNWatm> was successful.

```

The SunATM Device Drivers (SUNWatm) package was removed from the host *chances* in this example.

Note – The complete list of installed software displayed in the Software Manager window will then be updated to reflect the changes you made.

3.2 SBus ATM Interface Configuration

Note – SunATM-155 cards are supported on systems running the Solaris software environment, revision 2.4 or later (or other compatible operating systems).

See Table 3-1 for some of the acronyms and abbreviations used in this section.

Table 3-1 Acronyms and Abbreviations

Abbreviation:	Meaning:
AFI	Authority and Format Identifier
ATM ARP	ATM Address Resolution Protocol
B-LLI	Broadband - Lower Layer Information
ESI	End System Identifier
ICD	International Code Designator
IDI	Initial Domain Identifier
ILMI	Interim Local Management Interface
LIS	Logical IP Subnetwork
LLC/SNAP	Logical Link Control/Subnetwork Attach Point
NSAP	Network Service Access Point
PVC	Permanent Virtual Connection
RFC	Request for Comments
SNMP	Simple Network Management Protocol
SSCOP	Service Specific Connection Oriented Protocol
SVC	Switched Virtual Connection
VC	Virtual Connection
VCC	Virtual Channel Connection
VPC	Virtual Path Connection

3.2.1 Changes to System Configuration

3.2.1.1 Selecting SONET or SDH Framing Interface

The default framing interface is SONET, however, SDH is also supported. To change from the default SONET to SDH:

1. **Add the following line to the `/etc/system` file:**

```
set sa:sa_sdh = 1
```

2. **Reboot the system.**

Changes will not be in effect until the system is rebooted.

3.2.1.2 Editing the `/etc/aarconfig` File

TCP/IP and UDP/IP are supported over ATM transparently, therefore, the establishment of an ATM connection to carry the TCP/IP and UDP/IP packets is hidden from IP. When IP attempts to resolve an IP address to the physical address for an ATM interface, the IP address must first be resolved to an ATM address. A connection must then be established to the ATM address; this connection is identified by a virtual circuit identifier, or VC. This process is done via the ATM ARP (RFC 1577, "Classical IP and ARP over ATM") mechanism. See the description of the ATM ARP software, Section 3.2.1.6, "ATM ARP Daemon (aarpd)," for details on setting up the ATM ARP server and ATM ARP clients in a subnet.

The `/etc/aarconfig` file is a generic file that must appear on every SunATM system. It allows you to specify IP to ATM address translation, PVCs to destinations, and specify the VCI to an ATM ARP server. The environment allows for a mix of PVCs and SVCs.

Each time the `/etc/aarconfig` file is modified, you must restart the ATM ARP daemon (aarpd). It is recommended that aarpd be restarted by sending a SIGHUP signal to the process. aarpd is in the `/opt/SUNWatm/bin` directory. In addition, if the local address has been modified, the address registration daemon ilmids (see Section 3.2.1.10, "ATM Address Registration Daemon (ILMID)") must also be restarted in order to register the new address with the switch. ilmids is in the `/opt/SUNWatm/bin` directory. ilmids must be killed and restarted; a SIGHUP signal cannot be used.

Every node, or client, will have both an IP address and either an ATM address or a VCI. See Section 3.2.1.9, “ATM Address and Address Registration,” for ATM addressing schemes information.

Code Example 3-1 shows the format of the `/etc/aarconfig` file. The flags describe the options. See Table 3-2. All the flags required for most standard configurations are described in this section. If you have unusual configuration requirements, such as back-to-back connections over SVC, or interoperability issues, see Appendix E, “Advanced Configurations.”

In the IP-ATM address table shown in the `/etc/aarconfig` file:

- *Interface* is the last part of the device name in `/dev` (`sa0`, for example).
- *Hostname* is either an IP address in “dot” notation or the name of a host that should be locally available unless a non-ATM network connection also exists.
- *ATM address* consists of 20 octets with each octet represented by a one- or two-digit hexadecimal number and separated by colons.
- The *VCI* field is a positive decimal integer.
- An unused field is denoted by a “-”

Code Example 3-1 /etc/aarconfig File

```
#pragma ident  "@(#)aarconfig 1.13      95/03/13 SMI"
#-----
#
# IP-ATM address table -- (used by aarpd)
#
# Format
# -----
# Interface      Hostname      ATM_Address      VC      Flags
#
# Comments are represented by '#' at the beginning of the line.
# Unused fields are represented by '-'.
#
# Flags are:
#   l - assigns ATM address to local machine on ARP client or system that
#       doesn't use an ARP server
#   L - assigns ATM address to local machine on ARP server
#   t - adds an entry in the local table of IP to ATM addresses
#   s - creates an entry to the ARP Server
#   a - restricts access to the specified address on ARP server
#
#
# Variables
# -----
# Since ATM addresses are very long, the use of variables to represent
# portions of or entire ATM addresses is permitted.  Variables are set
# as follows:
#
# set switch1 = 45:00:00:00:00:00:00:00:0f:00:00:00:00
#
# A set of predefined variables is provided.  They are:
#
#   prefix: the 13-byte Network Prefix associated with the local switch.
#
#   mac: the 6-byte MAC address assigned to this host or interface.
#
#   sel: the default 1-byte Selector for this interface.
#
#   macsel: the concatenation of $mac:$sel.
#
#   myaddress: the concatenation of $prefix:$mac:$sel.  Should only be
#               used in 'l' and 'L' entries.
```

Code Example 3-1 /etc/aarconfig File (Continued)

```

#
#   anymac: wild card representing any 6-byte ESI.  Should only be used
#           in 'a' entries.
#
#   anymacsel: wild card representing any 7-byte ESI and Selector.  Should
#              only be used in 'a' entries.
#
#   sunmacselN: the concatenation of one of a set of reserved MAC addresses
#               (identified by N, which is a number in the range 0 - 199)
#               and $sel.  Since this is meant to be used as a server address,
#               which may apply across several switches, it does not include
#               a prefix.  The prefix must be explicitly specified.
#
#   localswitch_server: the concatenation of $prefix, a reserved MAC address,
#                       and $sel.  May be used as the server address in a
#                       one-switch network.  On the server, use of this as
#                       the local address also implies restricted access to
#                       hosts on this switch only.
#
# Use of these variables is demonstrated in the examples that follow.
#
# Examples
# -----
# Local Host - ARP Client
# -----
# The Hostname and VC fields are illegal for the local host entry.
#
# sa0 - $myaddress - l
# sa0 - $localswitch_server - s
#
# Local Host - ARP Server
# -----
# The Hostname and VC fields are illegal for the local host entry.
#
# sa0 - $localswitch_server - L
#
# SVC
# ---
# sa0 host1 $switch1:08:00:20:13:00:10:00 - t
# sa0 host2 $switch1:08:00:20:13:00:11:00 - t

```

Code Example 3-1 /etc/aarconfig File (Continued)

```

#
#
# PVC
# ---
# sa0 pvc_host1 - 110 t
# sa0 pvc_host2 - 111 t
#
#
# ARP Server over PVC (on clients' machines)
# -----
# The Hostname field is illegal for the server entry.
#
# sa0 - - 200 s
#
#
# ARP Server over SVC (on clients' machines)
# -----
# The Hostname field is illegal for the server entry.
#
# sa0 - $prefix:$sunmacsel0 - s
#
#
# ARP Server restricting access to 2 subnets
# -----
#
# sa0 - $prefix:$sunmacsel0 - L
# sa0 - $prefix:$anymacsel - a
# sa0 - $switch1:$anymacsel - a
#
#
# Add IP to ATM_Address or PVC entries below
#-----

```

Description of flags:

- I* Represents the ATM address of the local interface on ARP clients or systems not using an ARP server for ATM address resolution, and can be used to assign an ATM address to the host. *Hostname* should not appear; *ATM Address* should be provided if and only if SVCs are used. If an *s* entry is provided to use an ARP server (see below), *ATM Address* must be provided (a server is meaningful only in an SVC environment). See Table 3-2.
- L* Represents the IP and ATM address of the local interface on an ARP server. *Hostname* should not appear; *ATM Address* is required. See Table 3-2.

Note – If a client *aarconfig* does not contain an *I* entry for a given interface, *aarpd* will provide a default entry equivalent to:

```
san - $myaddress - 1
```

Where *n* is the physical interface number. An *L* entry must appear for each physical interface that represents an ARP server.

- t* Represents an IP to ATM address/VCI entry. *aarpd* adds these entries into the local table. Any *t* entries on the server must contain *ATM Address* and may also contain *VCI* if PVC communication between the server and client is desired. In addition, there are some cases when a *t* entry may be useful on an ARP client system. If a client wants to communicate with another system over PVCs, the PVC to be used is provided in a *t* entry containing *VCI*; or if a client wishes to cache frequently used addresses to avoid frequent ARP requests, a *t* entry containing *ATM Address* may be provided. See Table 3-2.

Note – In order for the two clients to communicate over PVCs, corresponding PVC connections must also be established in the ATM switch fabric.

- a* On an ARP server, *a* represents an address that may have access to this ARP server. If no *a* entries appear in a servers' *aarconfig* file, access to the server is unrestricted. Including *a* entries allows access to be restricted to known hosts. As an alternative to listing individual addresses, the ATM address field may contain a prefix, followed by the wildcard *\$anymacsel*, which matches any 7-byte ESI/Selector combination following the given

prefix. This allows access by any host connected to the switch specified by the given prefix. *Hostname* and *VCI* should not appear; *ATM Address* is required. See Table 3-2.

- s Specifies a connection to the ATM ARP server. Either *ATM Address* or *VCI* (in the case of a PVC connection) should appear, but not both. *Hostname* should not appear. The *s* entry is required on all clients that want to communicate with the server for ATM address resolution. See Table 3-2.

Note – Although SunATM supports PVC connections to a server for ARP traffic, RFC 1577 does not specify this case. For interoperability with other implementations, connections to the server should use SVCs.

Note – Corresponding ATM connections between the ATM ARP server and the client have to be established in the ATM switch fabric in order for the ATM ARP requests and replies to be exchanged over PVCs.

Table 3-2 /etc/aarconfig File Flags

* Interface	Host	ATM Address	VCI	Flags	*
required	illegal	SVC only	illegal	l	local information
required	illegal	required	illegal	L	local information on server
required	illegal	required	illegal	a	access list entry
required	required	or ¹	or ¹	t	permanent table entry
required	illegal	xor ²	xor ²	s	server address/PVC

¹or – Means one or the other required, and both are also legal.

²xor – Means one or the other required, but both are illegal.

Note – Entries in the *aarconfig* file must be grouped in a designated order: the local (*l* or *L*) entry must be first, the table (*t*) entries next, and then the server (*s*) entries. Other flags may appear in any order. Also, the ordering need only be maintained among entries for each physical interface; for example, all of the *sa0* entries may appear first, and then all of the *sa1* entries, etc.

Note – Additional flags that can be used for advanced configurations, such as back-to-back connections over SVC, or used to change the behavior of the interface, are described in Appendix E, “Advanced Configurations.”

3.2.1.3 *Using Variables in the /etc/aarconfig File*

Because the prefix portion of an ATM address specifies the ATM switch, a number of hosts specified in an `aarconfig` file may have ATM addresses which share the same prefix. To simplify setting up the `aarconfig` file, one can define variables that contain part of an ATM address.

A variable's name is an identifier consisting of a collection of no more than 32 letters, digits, and underscores (`_`). The value associated with the variable is denoted by a dollar sign (`$`) followed immediately by the variable name.

Note – Variables may only be used in the ATM address field. They may not be used in any of the other fields in an entry.

Multiple variables may be concatenated to represent a single ATM address expression. A colon must be used to concatenate the variables. Thus, if one variable, `v1`, is set to `11:22` and another, `v2`, is set to `33:44`, the sequence `$v1:$v2` represents `11:22:33:44`. Hexadecimal numbers may also be included with variables in the expression. The expression `45:$v1:$v2` would have the value `45:11:22:33:44`.

Variables are defined in the `aarconfig` file according to the following format:

```
set VARIABLE = EXPRESSION
```

where `VARIABLE` is the name of a variable and `EXPRESSION` is an expression concatenating one or two-digit hexadecimal numbers and/or the values of variables that have been previously defined. The equal sign is optional, but the variable and expression must be separated by either whitespace (spaces or tabs), an equal sign, or both.

Several predefined variables are built into the SunATM software. These variables are summarized in Table 3-3.

Table 3-3 Predefined SunATM Variables

Variable	Description
<code>prefix</code>	The 13-byte prefix associated with the local switch.
<code>mac</code>	The 6-byte MAC address associated with the local host or interface.
<code>sel</code>	The default 1-byte selector for the local interface.
<code>macsel</code>	The concatenation of <code>\$mac:\$sel</code> .
<code>myaddress</code>	The concatenation of <code>\$prefix:\$mac:\$sel</code> , resulting in the default address for the local interface.
<code>anymac</code>	A wild card representing any 6-byte ESI. Should only be used in <i>a</i> entries.
<code>anymacsel</code>	A wild card representing any 7-byte ESI and Selector combination. Should only be used in <i>a</i> entries.
<code>sunmacselN</code>	The concatenation of one of a series of reserved MAC addresses and <code>\$sel</code> to create a block of reserved ATM ARP server addresses. <i>N</i> should be a decimal number in the range 0 - 199.
<code>localswitch_server</code>	The concatenation of <code>\$prefix</code> , a unique reserved MAC address, and <code>\$sel</code> . When used as a server address, restricts server access to clients connected to the local switch only.

In most network configurations, the ATM address assigned to the local interface will be `myaddress`; using this variable in the *l* entry makes it possible to use identical `aarconfig` files on all clients using a given server.

The `sunmacselN` variables may be used in conjunction with a `prefix` as well-known server addresses which are not bound to a particular system. As an example, consider the case where a server which supports 50 clients fails. If the ATM address of the server is specific to that particular server, the *s* entry must be changed on all 50 clients in order to switch to a backup server. However, if instead the ATM address used for that server is `$prefix:$sunmacsel3`, this address is not only guaranteed to be unique, since it uses reserved MAC addresses, it is also possible to simply assign that address to the backup server on the same switch by changing the *l* entry to an *s* entry on one system, and bring up a new server with no changes to the clients.

Note – The `sunmacselN` variables do not include a prefix since a client and server may be on different switches and thus have different local prefix values.

In the case of a single-switch network, `localswitch_server` may be used as a well-known server address. Not only does it include the prefix associated with the local switch with a unique MAC address and appropriate Selector, it also restricts server access to clients on the local switch. Thus any host with a network prefix other than that of the local switch will be refused a connection to the ARP server if the ARP server's address is `$localswitch_server`.

Several rules apply to the use of variables in the `aarconfig` file:

1. Two variables cannot follow each other in an expression without an intervening colon. Thus `$v1:$v2` is legal while `$v1$v2` is not.
2. Fields in each line in the `aarconfig` file are separated by whitespace. Therefore variables should not be separated from the rest of an ATM address with whitespace. For example, `$v1: $v2` is illegal.
3. Once a variable is defined by a set command, it may not be redefined later in the `aarconfig` file.
4. The reserved variable names may not be set. These names include `prefix`, `mac`, `sel`, `macsel`, `myaddress`, `anymac`, `anymacsel`, `sunmacselN` (where `N` is a number between 0 and 199), and `localswitch_server`.

Note – The ESI portion of `localswitch_server` and the `sunmacselN` variables is a reserved MAC address. The hexadecimal values of the reserved addresses are:

```
localswitch_server      08:00:20:75:48:08
sunmacselN base        08:00:20:75:48:10
```

To calculate the ESI portion for a `sunmacselN` address, simply add the value of `N` (converted to a hexadecimal number) to the `sunmacselN` base address. For example, the ESI portion of `sunmacsel20` would be `08:00:20:75:48:10 + 0x14 = 08:00:20:75:48:24`.

3.2.1.4 Sample Configurations

The following examples demonstrate entries in the `/etc/aarconfig` file for several typical network configurations.

Although some of the examples show only one sample `aarconfig` file, similarly configured files must appear on each system. Example 2 shows the files for each of the three systems in the configuration.

1. SVC-only: Clients use the default address and access to the ARP server is restricted to clients on the local switch only.

- a. The `/etc/aarconfig` file on a client:

Interface	Host	ATM Address	VCI	Flag
sa0	-	\$localswitch_server	-	s

- b. The `/etc/aarconfig` file on the server:

Interface	Host	ATM Address	VCI	Flag
sa0	-	\$localswitch_server	-	L

2. PVC-only: *hosta* is connected to *hostb* and *hostc* over PVCs. There is no ARP server.

- a. `/etc/aarconfig` on *hosta*:

Interface	Host	ATM Address	VCI	Flag
sa0	-	-	-	l
sa0	hostb	-	100	t
sa0	hostc	-	101	t

- b. on *hostb*:

Interface	Host	ATM Address	VCI	Flag
sa0	-	-	-	l
sa0	hosta	-	100	t
sa0	hostc	-	102	t

c. on *hostc*:

Interface	Host	ATM Address	VCI	Flag
sa0	-	-	-	l
sa0	hosta	-	101	t
sa0	hostb	-	102	t

3. SVC-only: *hosta* uses SVCs to connect to *hostb* and *hostc*. All hosts are connected to the same switch; there is no ARP server.

Interface	Host	ATM Address	VCI	Flag
sa0	-	\$myaddress	-	l
sa0	hostb	\$prefix:08:00:20:d5:08:a8:00	-	t
sa0	hostc	\$prefix:08:00:20:21:20:c3:00	-	t

4. PVC/SVC mix: *hosta* uses a SVC to connect to *hostb*, and a PVC to connect to *hostc*. *hostb* is not on the local switch; there is no ARP server.

Interface	Host	ATM Address	VCI	Flag
sa0	-	\$myaddress	-	l
sa0	hostb	45:00:00:00:00:00:00:0f:00:00:00:00::08:00:20:d5:08:a8:00-	-	t
sa0	hostc	-	100	t

5. ARP server: Hosts are connected to an ATM ARP server that resolves addresses. Access is restricted to the local switch subnet and one additional switch subnet.

a. /etc/aarconfig on *hosta*:

Interface	Host	ATM Address	VCI	Flag
sa0	-	\$myaddress	-	l
sa0	-	\$prefix:\$sunmacsel0	-	s

b. `/etc/aarconfig` on server:

Interface	Host	ATM Address	VCI	Flag
sa0	-	\$prefix:\$sunmacsel0	-	L
sa0	-	\$prefix:\$anymacsel	-	a
sa0	-	45:00:00:00:00:00:00:00:0f:00:00:00:00:\$anymacsel	-	a

3.2.1.5 *The `/etc/hostatm.sa#` File*

In addition to the `aarconfig` file, a `hostatm.sa#` file (where # is the physical interface number) must appear in `/etc` for each physical interface. The `hostatm.sa#` file must contain the hostname associated with the physical interface identified by `sa#`. The file is created by the installation script for each interface that is specified during software installation; other interfaces may be added later by manually creating this file.

The `hostatm.sa#` file is used much like the `hostname.xx#` file that is traditionally associated with network interfaces. However, because ATM software has many new and different modules, its configuration (plumbing) must be handled differently. The `ifconfig` program in the Solaris 2.4 operating environment does not know about these differences; hence the need for a different identification of ATM interfaces. A `hostname.sa#` file must not appear in `/etc`.

As with any network interface using IP, the hostname of an interface must also be associated with an IP address. This is done for SunATM interfaces in the same way it is done for other network interfaces: either with an entry in `/etc/hosts`, or with an entry on a naming server running a naming service such as NIS.

3.2.1.6 ATM ARP Daemon (aarpd)

Depending on the `aarconfig` file, the ATM ARP daemon (`aarpd`) will run as either a server or a client. As a server, `aarpd` is responsible for handling ATM ARP requests originating from its clients. An ATM server has to be configured for each subnet. The ATM ARP server code conforms to RFC 1577: clients send ATM ARP requests to the server to resolve a destination IP address to an ATM address. The server then replies to ATM ARP requests by sending an ATM ARP response. If the server does not have the IP to ATM address entry, then it replies with NAK.

The file `/etc/aarconfig` is also used by the ATM ARP server. All the IP to ATM address entries specified in the file will be entered into a kernel resident table. Additional entries in the kernel table will be added dynamically using the inverse ARP process. When a client connects to the server, the server will send an inverse ARP request back to the client to obtain the client's IP address. When a response is received, an entry will be created for that client. The daemon will also respond to client ARP requests. The daemon looks up a kernel IP to ATM address entry and responds to an ATM ARP request with either an ATM ARP reply or ATM ARP NAK (if there is no entry in the table). Note that an ATM ARP client uses the VC specified in the `/etc/aarconfig` file to communicate with the server; or, if an ATM address is specified, it establishes an SVC connection to communicate with the server.

While dynamic entries in the ARP server's table make network administration less complex, it also creates a security problem. Any host may register with the ARP server and, therefore, gains access to the subnet. To resolve this issue, a list of hosts or networks may optionally be provided with `a` entries in the server's `/etc/aarconfig` file. If no `a` entries appear, any host will be allowed to connect to the server. If any `a` entries exist, only those hosts whose addresses match those specified will be allowed to connect.

Although the `a` entry requires a complete ATM address, multiple addresses can be referenced in a single entry using the provided wildcards. The variable `anymacsel` represents any ESI and Selector combination. When combined with a 13-byte network prefix, it creates an entry that matches any host on the switch specified by the given prefix. Additionally, a question mark (?) may be used as a wildcard with finer granularity, representing one or two hexadecimal digits within any colon-separated field. Thus, `$prefix:$anymac:?` is equivalent to both `$prefix:$anymac:??` and `$prefix:$anymacsel`, and it will allow addresses with any selector value. However, `$prefix:$anymac:0?` will allow only addresses with selectors in the range of

00 to 0f. Finally, using the predefined variable `localswitch_server` as the server's ATM address implies the existence of an entry with the ATM address `$prefix:$anymacsel`.

The advantage to having an ATM ARP server in the subnet is that it represents a known source for all address resolutions. It is the only host which a client must know about to have IP addresses resolved to ATM connections, and it allows for access control in the ATM network.

When the `/etc/aarconfig` file has been modified on a system, it is necessary to restart `aarpd`. This can be done automatically by sending a `SIGHUP` signal to `aarpd`. Restarting `aarpd` in this manner is preferred because it causes less disruption to network services than killing `aarpd` and restarting it manually. `ilmid` must also be restarted if the local address is changed so the new address is registered with the switch. `ilmid` must be killed and restarted (a `SIGHUP` signal cannot be used).

Note - For better caching, all clients have the option of adding to their configuration file the IP to ATM address information for other clients. This can benefit clients who communicate frequently because it eliminates having to go through the ATM ARP server for IP to ATM address resolution.

Note - If a host has multiple SunATM cards, the host may be a server for one IP subnet and a client for another. This is handled transparently by `aarpd`.

3.2.1.7 IP and ATM Signaling

ATM is connection oriented, so a connection must be established between two communicating entities before data transfer can begin. IP is inherently connectionless. The implementation on the host has to reconcile the differences in these two paradigms.

SunATM Signaling architecture allows ATM connections to be established transparently to IP. IP itself sees the ATM interface just as it sees any traditional network interface. Every SunATM interface has a subnet IP address. During the process of bringup of an ATM interface, appropriate signaling and resolution modules are plumbed. All the TCP/IP and UDP/IP applications run without modifications, and all the utilities associated with the network interfaces also run without modification and display similar results (for

example, `netstat`, `ifconfig` utilities, etc.), with one exception. Because of the different plumbing of the ATM modules, the `plumb` and `unplumb` options of `ifconfig` will not work on ATM interfaces; the `atmplumb` command should be used instead. IP treats the ATM interface as a subnet, choosing the interface used to send a packet out based on the IP address of the destination and on the IP address and netmask of the interface itself.

The transparency to IP is enabled by the way the ATM modules are plumbed on bringup. When IP receives a packet to send over the ATM interface, IP checks its own tables for address resolution. If the address is not resolved, IP sends up an ARP request, which is serviced by the ATM Address Resolution modules, including the ATM ARP Daemon described in Section 3.2.1.6, “ATM ARP Daemon (`arpd`),” on page 3-29. When those modules have located the desired ATM address, either in their own tables or by contacting the ARP server, they will then use the signaling modules to open a call to that destination. The VC for that call is then returned to IP; IP is now able to send the data directly to the driver.

SunATM signaling conforms to the UNI 3.0 spec of the ATM Forum. Q.2931 runs on top of SSCOP and uses VC 5 for signaling as specified in the Forum spec.

UNI 3.0 uses ATM addresses for signaling. Every ATM interface will have an ATM address in addition to the IP address.

3.2.1.8 *ATM Address Resolution*

Traditional TCP/IP and UDP/IP applications use IP addresses for communicating to a destination. In order for these applications to run as before, there is now a need to resolve these IP addresses into ATM addresses. The ATM address is then used in signaling to establish an ATM connection to the destination. An ATM connection in turn is represented by a VPI/VCI. The host must use this returned VPI/VCI to send packets to the destination representing the ATM connection.

ATM address resolution, also called ATM ARP, follows RFC 1577, the classic draft that describes the ATM ARP process.

RFC 1577 is based upon the existence of an ATM ARP server on every subnet. Every client of the subnet communicates with the ATM ARP server to derive an ATM address of the destination from the IP address of the destination. The ATM ARP server holds the IP to ATM address information for all hosts in the

ATM subnet. It is likely that initial ATM configurations will not rely on dynamic ATM address resolution since it requires the presence of an ATM ARP server on every subnet. Also, there are no specified standards for providing redundant ATM ARP servers for a subnet. As specified, the ATM ARP server would constitute a single point of failure in the system. From a practical standpoint, however, early configurations may take the course of having the IP to ATM address data base in every system, thus avoiding the IP to ATM address resolution step altogether. The RFC requires the use of a router to pass data between subnets.

SunATM software facilitates this by providing ATM utilities that will allow configurations to specify IP to ATM addresses in `/etc/aarconfig` files. The `aarpd` uses the information in `/etc/aarconfig` to create IP to ATM address resolution tables. Dynamic entries into a server's resolution table are also supported.

Code Example 3-1 shows the format of the `/etc/aarconfig` file for specifying the IP to ATM address. It is important for the file to be consistent on all systems in the subnet. See Section 3.2.1.2, "Editing the `/etc/aarconfig` File," on page 3-16.

The ATM Address Resolution software contains several timers which control its resource usage. Two key timers include the timer controlling the teardown of inactive connections to peers or the server, and the timer controlling the deletion from the local kernel table of ARP entries received from peers or the server. You may choose to lengthen or shorten these timers based on a particular network's needs. This is done with system variables, which may be added to the `/etc/system` file. After changing this file, you must reboot the system for the changes to take effect. Set the variables as follows:

```
set aar:aar_max_quiescent_qcc = m
set aar:aar_max_quiescent_ace = n
```

where *m* and *n* represent the number of 30 second intervals each timer should last. The first variable, `aar_max_quiescent_qcc`, determines how many intervals to wait before tearing down a connection to a peer or server. The receipt or transmission of a packet on the connection resets the timer. The second variable, `aar_max_quiescent_ace`, determines how many intervals to wait before removing an arp cache entry from the local table. This applies only to entries that are created via the ATM ARP process, and not to those entries that are loaded from the `/etc/aarconfig` file. Receipt or transmission of a packet from or to the peer, represented by the entry, resets the timer.

3.2.1.9 ATM Address and Address Registration

ATM addresses, like NSAP addresses, are 20 octets long. The End System Identifier (ESI) field within the ATM address is a unique 6 octet value; this can be the IEEE hardware MAC address conventionally associated with every network interface. Selector field (SEL) is 1 octet long. The 13 octets that make up the rest of the ATM address should be derived from the ATM switch fabric to which the interface will connect. Every ATM switch fabric is configured with a 13 octet ATM address field. On a SunATM host, the prefix associated with the local switch fabric is represented by the variable `prefix`. Its value will be obtained by the system at configuration time.

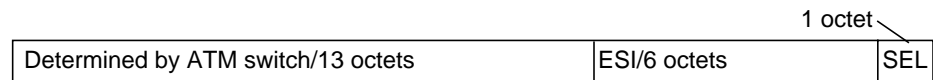


Figure 3-1 ATM Address Fields

UNI 3.0 spec specifies the Interim Local Management Interface (ILMI) service interface for a client to learn and register its ATM address. The ILMI service interface is based on the use of SNMP over AAL5. In the SunATM software package, ILMI service is provided by an address registration daemon, `ilmid`.

3.2.1.10 ATM Address Registration Daemon (ILMID)

Address registration with a switch is controlled by `ilmid`. When an ATM interface is brought up at boot time, `ilmid` is also started. `ilmid` then begins an exchange of messages with the switch: relaying local address information (the 7 octet ESI and selector) to the switch, and receiving the 13 octet network prefix information from the switch.

To obtain the local address, `ilmid` first checks the `I` or `L` entry in the `/etc/aarconfig` file. If an ESI and selector are specified there, they are used. If `$myaddress`, `$macsel`, `$mac` is used, `ilmid` obtains the system mac address from the ATM driver.

3.2.1.11 Encapsulation of IP Packets

SunATM interface encapsulates the IP packets as LLC_SNAP, as specified in RFC 1483, "Multiprotocol Encapsulation over ATM Adaptation Layer 5."

SETUP messages, when originating a call, carry a B-LLI information element. The B-LLI information element indicates that the call is LAN LLC (ISO 8802/2). The incoming call has to carry the B-LLI information element for the SunATM interface to accept the call.

3.3 Rebooting the System and Examining Network Interfaces

1. Reboot the system using the `boot -r` command.

The `-r` option is required by the Solaris 2.4 software environment when installing new hardware, and the `-v` option causes boot messages to be printed in verbose mode to allow the user to see that the SunATM clients are correctly recognized.

To boot from the default boot device use:

```
ok boot -rv
```

Note – For Solaris 2.x, use `boot -r` whenever the physical configuration of the system is changed.

2. Execute `ifconfig -a` and `netstat -i` commands to examine the state of all network interfaces.

You can also use `/usr/sbin/ping` or `/usr/sbin/spray` commands to see if a network interface is active.

Examples of output for `ifconfig -a`, `ping`, and `netstat -i` follow.

```
zardoaz% /sbin/ifconfig -a
lo0: flags=849<UP,LOOPBACK,RUNNING,MULTICAST> mtu 8232
    inet 127.0.0.1 netmask ff000000
sa0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
    inet 129.144.10.57 netmask ffffffff broadcast 129.144.10.255
```

```
zardoaz% /usr/sbin/ping zelda
zelda is alive
```

```
zardoz% netstat -i
Name Mtu  Net/Dest      Address      Ipkts  Ierrs  Opkts  Oerrs  Collis  Queue
lo0  8232  loopback      localhost    87     0      87     0      0       0
sa0  1500  umtv15-010-n zardoz       421873 0      60009  1      183     0
```



Caution – Do not change the SBus slot in which a SunATM-155 card is installed once the system has been booted. The Solaris 2.x software environment remembers the location of each SBus card that has been installed. Switching SBus slots will cause the operating system to assume that you removed your original SunATM-155 card and added a second card to the system. Refer to the online man page about `path_to_inst` for more information.

Wiring Scheme and Pin Descriptions

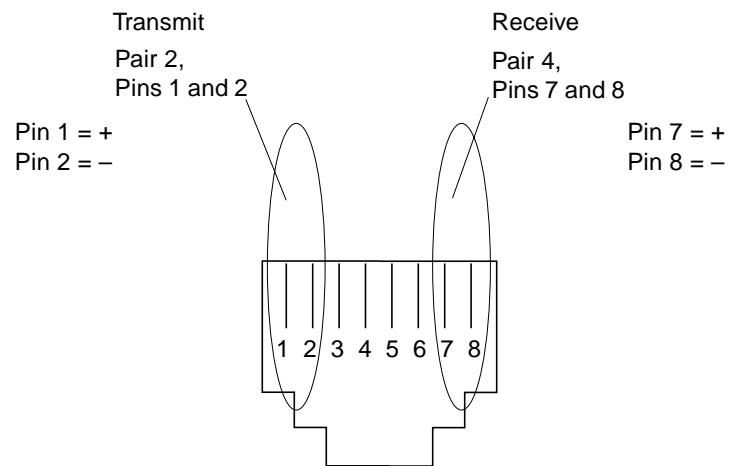


Figure A-1 Designation T568B

Table A-1 Pin Descriptions for the 96-Pin SBus Connector

Pin	Description	Pin	Description	Pin	Description	Pin	Description
1	GND	2	BR\	49	Clk	50	BG\
3	Sel\	4	IntReq1\	51	AS\	52	GND
5	D00	6	D02	53	D01	54	D03
7	D04	8	IntReq2\	55	D05	56	+5V
9	D06	10	D08	57	D07	58	D09
11	D10	12	IntReq3\	59	D11	60	GND
13	D12	14	D14	61	D13	62	D15
15	D16	16	IntReq4\	63	D17	64	+5V
17	D19	18	D21	65	D18	66	D20
19	D23	20	IntReq5\	67	D22	68	GND
21	D25	22	D27	69	D24	70	D26
23	D29	24	IntReq6\	71	D28	72	+5V
25	D31	26	Siz0	73	D30	74	Siz1
27	Siz2	28	IntReq7\	75	Rd	76	GND
29	PA00	30	PA02	77	PA01	78	PA03
31	PA04	32	LErr\	79	PA05	80	+5V
33	PA06	34	PA08	81	PA07	82	PA09
35	PA10	36	Ack0\	83	PA11	84	GND
37	PA12	38	PA14	85	PA13	86	PA15
39	PA16	40	Ack1\	87	PA17	88	+5V
41	PA18	42	PA20	89	PA19	90	PA21
43	PA22	44	Ack2\	91	PA23	92	GND
45	PA24	46	PA26	93	PA25	94	PA27
47	DtaPar	48	-12V	95	Reset\	96	+12V

SunATM-155 SBus Cards Specifications



B.1 Performance Specifications

Table B-1 Performance Specifications

Feature	Specification
SBus Clock	25 MHz max., 12.5 MHz min.
Max SBus Burst Transfer Rate	34 Mbytes/sec (approximately)
Steady State SBus Transfer Rate	5 Mbytes/sec
SBus Data/Address Lines	D (31:0)/PA (27:0)
SBus Modes	Master/Slave
Capacitance per SBus Signal Line	≤20 pF
SBus Parity	Yes
SBus Version	Rev B.0
SBus Burst Sizes	16/32

B.2 Power Specifications

Table B-2 Power Specifications

Specification	Measurement
Power Dissipation	9.5 Watt max.
Voltage Tolerance	+/- 5%
Ripple	≤ 100 mV
Operational Current	5V, 2.0 Amps

B.3 Physical Dimensions

Table B-3 Physical Dimensions

Dimension	Measurement
Length	5.78 in. (146.70 mm)
Width	3.3 in. (83.82 mm)

B.4 Environmental Specifications

Table B-4 Environmental Specifications

Condition	Operating Specification	Storage Specification
Temperature	0 to 70°C (+32 to +131°F)	-25 to 70°C (-25 to +131°F)
Relative Humidity	5 to 85% non-condensing (40°C, wet bulb temperature)	0 to 95% non-condensing 40°C /hour
Altitude	-1000 to +15,000 ft.	-1000 to +50,000 ft.
Shock	5g, 1/2 sine wave, 11 msec	30g, 1/2 sine wave, 11 msec
Vibration, pk to pk displacement	0.005 in. max. (5 to 32 Hz)	0.1 in. max (5 to 17 Hz)
Vibration, peak acceleration	0.25g (5 to 500 Hz) (Sweep Rate = 1 octave/min.)	1.0g (5 to 500 Hz) (Sweep Rate = 1 octave/min.)

Running Diagnostic Tests



The diagnostic tests available for the SunATM-155 cards are selftest and the SunDiag™ system exerciser.

C.1 Selftest

The SunATM-155 SBus cards selftest verifies correct operation of the SBus card. The selftest consists of a suite of tests that reside in the FCode PROM on the card. The code is written in Forth programming language and can only be run under OpenBoot PROM (OBP) version 2.x or later.

Note – To find the OBP revision level on your system, type `.version` at the `<#0> ok` prompt.

The SunATM-155 selftest does not automatically run after power on or reset, but you can use selftest any time you want to determine the status of the hardware.

Note – Selftest does not require connection to the network. Selftest will test internal loopback (up to SUNI).

- As a Sun SPARCstation system is powered up, the following banner is displayed:

```
SPARCstation 10, Type 4 keyboard
Rom Rev 2.4, 16MB memory installed
Ethernet address 8:0:20:8:42:7, Host ID 51000007

Type b (boot), c (command), n (new command)
>n
ok
```

- Check that the PROM version is 2.x or later.
 - If the system is set up to automatically boot, press key combination L1-A to stop it.
 - If the system is not already at the *ok* prompt, select *n* for the new command mode to get to the *ok* prompt.
- To check that the system has a SunATM-155 SBus card installed, and in which SBus slot, look for the *sa* device in the following command:

```
ok show-devs
...
/iommu@f,e0000000/sbus@f,e0001000
/iommu@f,e0000000/sbus@f,e0001000/sa@3,0
...
ok
```

- To display the OBP set of environment variables, type the following command:

```
ok printenv
selftest-#megs      1
...
auto-boot?         true
...
fcode-debug        false
...
ok
```

- To change an OBP environment variable, for example *auto-boot?* to *false*, type the `setenv` command as follows, then type *reset* for the change to take effect:

```
ok setenv auto-boot? false
ok reset
```

- To browse the OBP device tree:
 - Type `cd` to get to a specific working directory.
 - Type `words` to find all the Forth words available in that directory.

```
ok cd /sbus/sa@3,0
ok words
close      open      reset
selftest
ok
```

- To examine the definition of a word, use the `see` command as follows:

```
ok see selftest
: selftest
  (ffd988d0) (ffd98c28) (ffd99564) (ffd98c60) swap (ffd5fb9c)
  (ffd988d0)
  if
    (ffd995ec)
  else
    -1
  then
;
ok
```

For more information on using the OBP commands, refer to the Open Boot Command Reference manual, part no. 800-6076-xx.

C.1.1 Setting the Diag-switch

The selftest can be invoked from the `ok` prompt on a Sun system that has OBP 2.x or later as follows:

1. Set the diag-switch to true.

```
ok setenv diag-switch? true
```

Note – Optional. If the `diag-switch` is not set to “true,” reduced tests are run.

2. Test the card. See Section C.1.2, “Running Selftest.”

As each test is executed, the resulting status of either *pass* or *fail* will be displayed. The tests will run sequentially and will stop when any test encounters an error.

3. Set the diag-switch to false, if necessary, after running tests.

```
ok setenv diag-switch? false
```

C.1.2 Running Selftest

- ♦ **To test the SunATM-155 card on a SPARCstation 10 system, at the `ok` prompt type:** `test /iommu/sbus/sa@<slot#>,0`

```
ok test /iommu/sbus/sa@<slot#>,0
```

Note – If the `test` command fails, verify that the SBus card hardware is installed correctly. If necessary, replace the SBus card and/or contact your service provider.

C.2 SunDiag

SunDiag is an online system exerciser that runs diagnostic hardware tests. It is used primarily with the OpenWindows software interface that enables you to quickly and easily set test parameters to run tests.

C.2.1 SunDiag Window

Note – Examples in this section show SunDiag running in the OPEN LOOK® environment. SunDiag, run in the SunView™ environment, will look different.

♦ **To start SunDiag, cd to the sundiag directory and type the `sundiag` command:**

```
zardoz# cd /opt/SUNWdiag/bin
zardoz# sundiag
```

The SunDiag window, the primary interface for running SunDiag, will be displayed on your screen (see Figure C-1).

The SunDiag window is divided into four small windows:

- The system status window at the upper-left of the screen displays the status of the tests.
- The performance monitor panel in the upper-middle of the screen displays the performance statistics for the system that is under test.
- The control panel on the right includes buttons, exclusive choice, toggle, and pop-up menus that allow you to select test parameters and options.
- The console window at the bottom-left displays test messages and allows you, as a superuser, to use operating system commands.

Refer to the SunDiag manual that came with your operating system for further details.

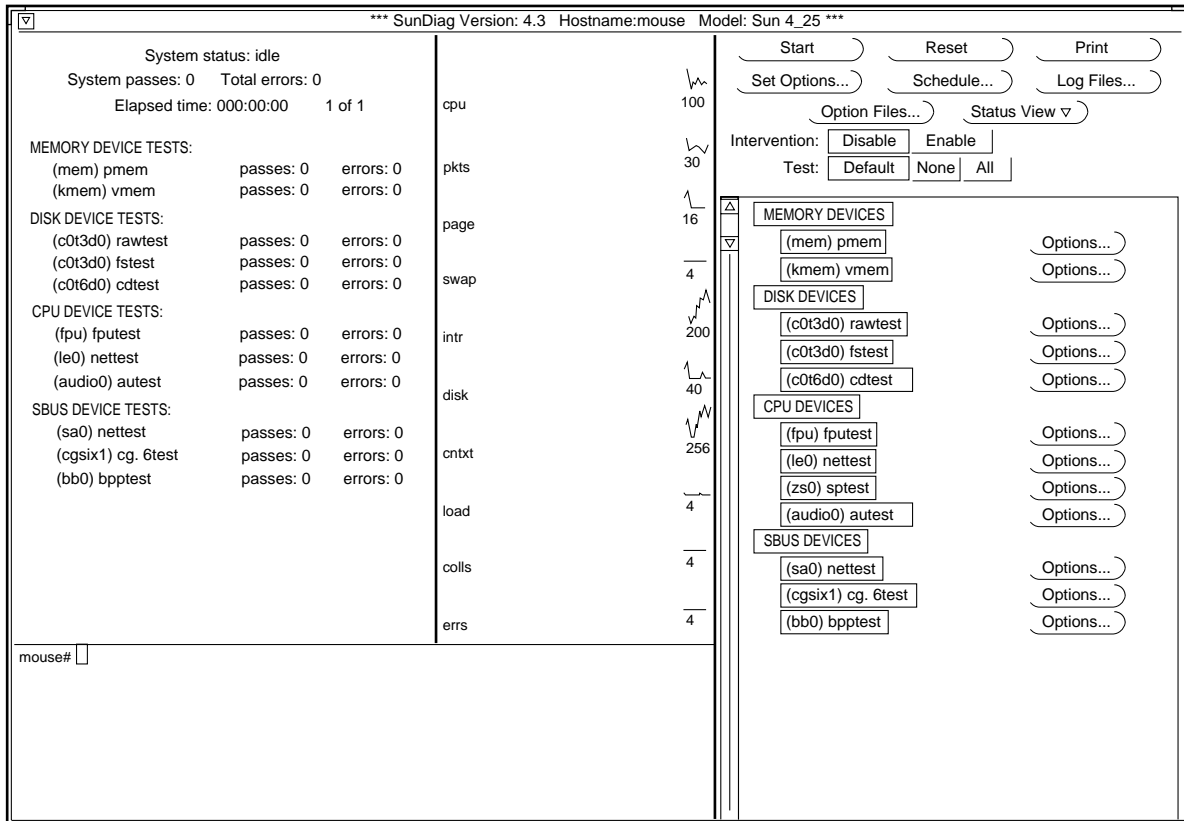


Figure C-1 SunDiag Window

C.2.2 Starting the Test

1. Review the information in the control panel that identifies the devices available for testing.
2. Make sure *Enable* is selected for the “Intervention:” setting.
3. Select “SBUS DEVICES.”
 - a. select the (sa0) nettest button under “SBUS DEVICES.”
 - b. click on the *Options* button for (sa0) nettest.

-
4. **In the pop-up option window that is displayed:**
 - a. **enter “Target Host:” information.**
Enter the name of the target host for sa0.
 - b. **change the “Delay” time from the default setting of 120, if appropriate.**
 - c. **click on the *Apply* button in the pop-up option window.**
 5. **Click on the *Start* button in the control panel.**
 6. **Watch the console window for messages.**
 7. **To interrupt a test or to stop after a test is completed, click on the *Stop* button.**

Note – If no problems are identified during the testing, the SunATM-155 card is ready for operation in your system.

8. **To exit SunDiag, use your mouse to *Quit* the SunDiag window.**

Application Programmers' Interface



The Application Programmers' Interface (API) that is provided with this software release is an interim API to be used until the ATM Forum standardizes an API.

Note – Be aware that since this is an interim API, it can be changed at any time.

Note – For historical reasons, Q.93B and Q.2931 are used interchangeably.

The interim API that Sun provides:

- will cover Q.93B to setup and tear down connections
- will also deal directly with the driver

Each API set, Q.93B and driver, contains both a user API and a kernel API.

The API, called Q.2931 Call Control (qcc), consists of two sets of similar functions: one for applications running in the kernel, and one for applications running in user space. Each set provides functions to build and parse the messages required to setup and tear down connections. Table D-2 lists those message types. Figure D-2 shows the message format that should be used by kernel applications; user space applications should use two strbuf structures instead of the two mblks.

The qcc man pages that provide more details on how to use these functions are:

user space	qcc_bld (3), qcc_parse (3), and qcc_len (3)
kernel space	qcc_bld (9F) and qcc_parse (9F)

The man page for the signaling driver is q93b (7).

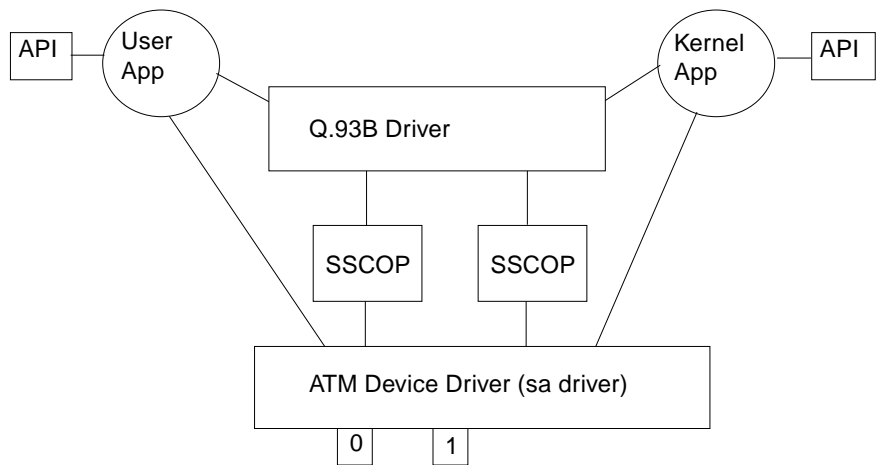


Figure D-1 ATM Signaling

D.1 Q.93B API

The Q.93B driver is an M-to-N MT safe (D_MP) mux driver. On the upper side, the Streams interface is the Q_Primitives that consists of two mblocks. The first mblock is the header of the type M_PROTO, and the second mblock is the raw Q.93B message of the type M_DATA.

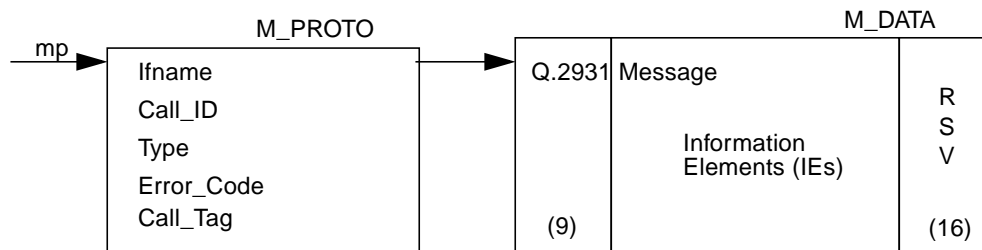


Figure D-2 Message Format

Table D-1 Message Meanings

	Explanation
Ifname	A null-terminated string containing the device name (for example, sa0)
Call_ID	A unique number from Q.93B per interface.
Type	The same as the Q.93B message type except there is a local Non-Q.93B message type SETUP_ACK. The SETUP_ACK message is used to provide the Call_ID to the user.
Error_Code	The error returned from Q.93B when an erroneous message is received from the user. The exact same mblock chain shall be returned to the user with the Error_Code field set. The user shall always clear this field
Call_Tag	A number assigned by the calling application layer to a SETUP message. When a SETUP_ACK is received from Q.93B, the Call_ID has been set; the Call_Tag field may be used to identify the ack with the original request. From that point on, the Call_ID value should be used to identify the call.

The upper layer shall leave the Q.93B header portion (9 bytes) of the Q.93B message in the second mblock blank. The Q.93B driver will fill in the Q.93B header. The upper layer should reserve 16 bytes at the end of the second mblock for the layer 2 protocol for performance.

D.1.1 Q.93B Driver

Table D-2 Messages Between the User and the Q.93B Driver

Message Type	Direction
SETUP	BOTH
SETUP_ACK	UP
CALL_PROCEEDING	BOTH
CONNECT	BOTH
CONNECT_ACK	UP
RELEASE	DOWN
RELEASE_COMPLETE	BOTH
STATUS_ENQUIRY	DOWN
STATUS	UP
RESTART	BOTH
RESTART_ACK	BOTH
UP is from Q.93B to user; DOWN is from user to Q.93B	

▼ Setup Procedure

When the user decides to make a call, the user sends a SETUP message down to Q.93B and waits for a SETUP_ACK from Q.93B. After SETUP_ACK is received, the user waits for either a CALL_PROCEEDING, CONNECT, or RELEASE_COMPLETE message from Q.93B (Q.93B ignores all other messages). After the CONNECT message is received, the user can use the virtual channel.

When the user receives a SETUP message from Q.93B, the user shall respond with either a CALL_PROCEEDING, CONNECT, or RELEASE_COMPLETE message to Q.93B. After the CONNECT_ACK message is received, the user can use the virtual channel.

▼ Release Procedure

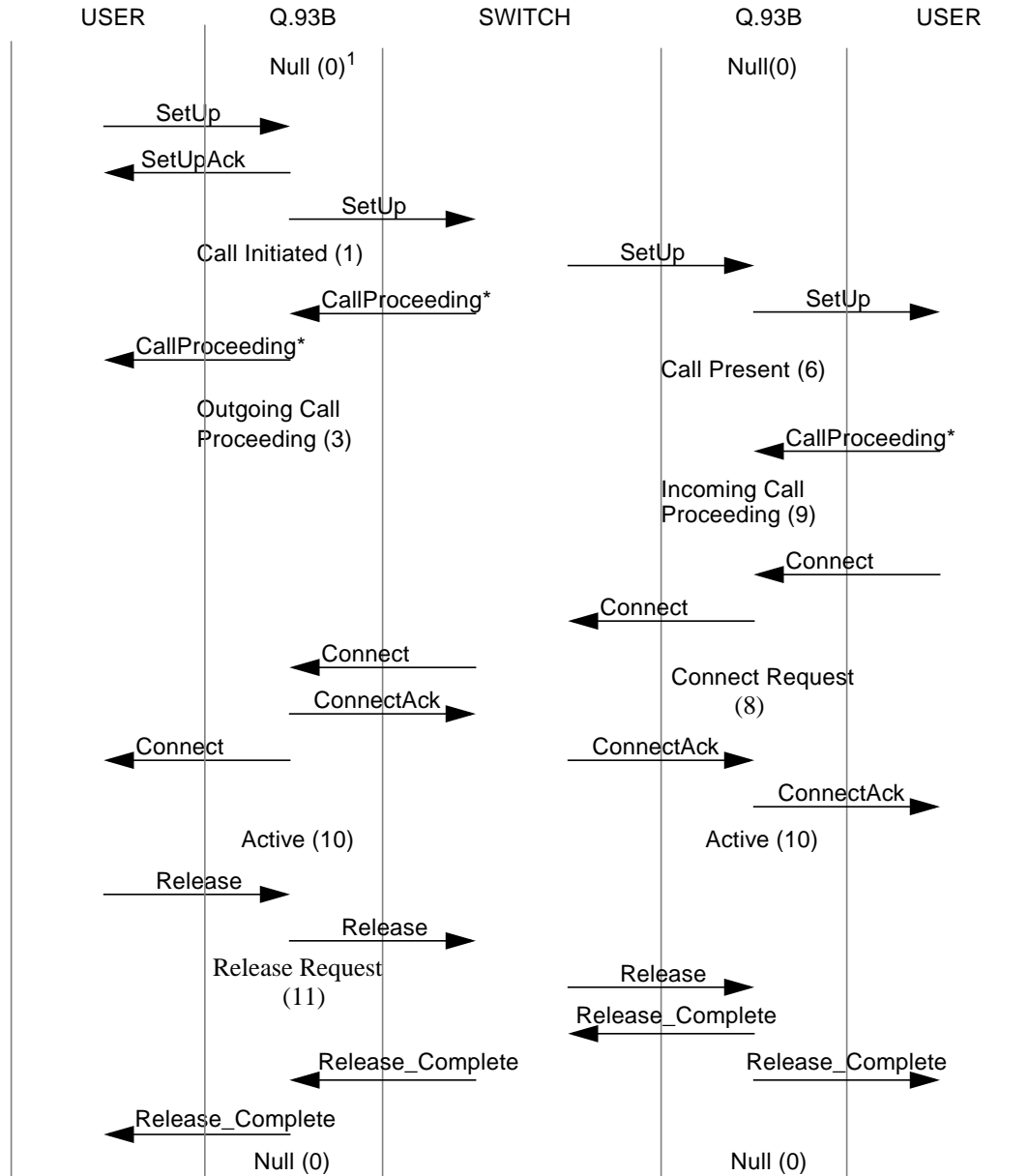
To clear an active call or a call in progress, the user sends a RELEASE message down to Q.93B and waits for a RELEASE_COMPLETE from Q.93B. Anytime the user receives a RELEASE_COMPLETE message from Q.93B, the user shall release the virtual channel if the call is active or in progress.

Q.93B never sends a RELEASE message to the user. The user only sends the RELEASE_COMPLETE message right after sending a SETUP_ACK message to Q.93B to reject the call in response to a SETUP message from Q.93B. At any other time, to reject or tear down a call, the user shall send a RELEASE message to Q.93B.

▼ Exception Conditions

When a message is received from the user with an unattached port, Q.93B shall return the message with the Error_Code BAD_PORT.

Normal Call Setup and Tear Down



¹ XX(n): Q.93B State Name (Q.93B State Number)

* Optional

D.1.2 Q.93B User Space API

Q.93B user space consists of all `qcc_bld`, `qcc_len`, `qcc_parse`, and `qcc_util` functions under section 3.

See the `qcc_bld (3)`, `qcc_len (3)`, `qcc_parse (3)`, and `qcc_util (3)` man pages that follow.

qcc_bld(3)**C Library Functions****qcc_bld(3)****NAME**

qcc_bld, qcc_bld_setup, qcc_bld_call_proceeding,
qcc_bld_connect, qcc_bld_release, qcc_bld_release_complete,
qcc_bld_status, qcc_bld_status_enquiry, qcc_bld_restart,
qcc_bld_restart_ack - build Q.2931 messages

SYNOPSIS

```
cc [ flag ... ] file ... -latm [ library ... ]
```

```
#include <atm/types.h>
```

```
#include <atm/qcc.h>
```

```
int qcc_bld_setup(strbuf_t *ctlp, strbuf_t *datap,  
                 char *ifname, int calltag, int vci, int forward_sdusize,  
                 int backward_sdusize, atm_addr_t *src_addrp,  
                 atm_addr_t *dst_addrp, int sap, int endpt_ref);
```

```
int qcc_bld_call_proceeding(strbuf_t *ctlp, strbuf_t *datap,  
                            char *ifname, int callid, int vci, int endpt_ref);
```

```
int qcc_bld_connect(strbuf_t *ctlp, strbuf_t *datap,  
                   char *ifname, int callid, int vci,  
                   int forward_sdusize, int backward_sdusize,  
                   int endpt_ref);
```

```
int qcc_bld_release(strbuf_t *ctlp, strbuf_t *datap,  
                   char *ifname, int callid, int cause);
```

```
int qcc_bld_release_complete(strbuf_t *ctlp,  
                             strbuf_t *datap, char *ifname, int callid,  
                             int cause);
```

```
int qcc_bld_status_enquiry(strbuf_t *ctlp, strbuf_t *datap,  
                           char *ifname, int callid, int endpt_ref);
```

```
int qcc_bld_status(strbuf_t *ctlp, strbuf_t *datap,  
                  char *ifname, int callid, int callstate, int cause,  
                  int endpt_ref, int endpt_state);
```

qcc_bld(3)

```
int qcc_bld_restart(strbuf_t *ctlp, strbuf_t *datap,  
                  char *ifname, int callid, int vci, int rstall);
```

```
int qcc_bld_restart_ack(strbuf_t *ctlp, strbuf_t *datap,  
                       char *ifname, int callid, int vci, int rstall);
```

MT-LEVEL

Safe.

AVAILABILITY

The functionality described in this man page is available in the SUNWatma package included with the SunATM adapter board. The libatm.a library, which is located in /usr/lib, must be included at compile time as indicated in the synopsis.

DESCRIPTION

These functions build the various messages that make up the Q.2931 protocol which is used for ATM signalling. A full description of the message format and use can be found in the ATM Forum's User Network Interface Specification, V3.0. The functions may be used by processes which are running in user space.

In general, no error checking is performed on the data that is passed in. Whatever data is passed in will be placed in the message that is built without examination. The only exceptions to this are mentioned in the function descriptions.

Each function requires a minimum of 4 parameters: ctlp and datap, which are pointers to strbuf_t buffers; ifname, which is a string containing the physical interface (such as sa0); and an integer, either calltag or callid, depending on the message type. calltag is used in the setup message only; it is a reference number that is assigned by the calling application. callid is used in all other messages; it is assigned by the lower layer and will be sent up to the user, with the calltag, in the setup_ack message.

qcc_bld(3)

ctlp and datap make up the control and data portions of the constructed message, corresponding to the M_PROTO and M_DATA blocks of the message that will be passed downstream. The buffer fields in the structures which ctlp and datap point to (ctlp->buf and datap->buf) must be allocated before calling a qcc_bld* function; size information may be obtained using the qcc_bld*_datalen() functions (see qcc_len(3)).

After successful return from a qcc_bld* function, the message may be passed down an open stream using the putmsg(2) function, with ctlp and datap as the buffer parameters for putmsg.

Other parameters for each function depend on the type of information required for each message type, and are defined in the paragraphs describing each function call.

After a message has been built, the user may add IEs that are not built into the message; however, the size information returned by the qcc_len functions only includes the IEs documented here. The user must allocate enough additional space and correct the message length value in the Q.2931 header if additional IEs are required in the message.

qcc_bld_setup() constructs a setup message containing some or all of the following Information Elements: AAL parameters, ATM user cell rate, broadband bearer capability, called party number, calling party number, quality of service parameter, and endpoint reference. The user must pass in the forward and backward sdu sizes for the AAL parameter IE, an ATM address for the destination for the called party number IE, and one for itself for the calling party number IE (atm_address_t format is defined in the <atm/qcc.h> header file). The value passed in the sap parameter is placed in a broadband higher layer IE. The higher layer IE indicates the sap to which received messages should be directed. If the user passes in a positive vci, a connection identifier IE will be included; if the user passes in a non-negative endpt_ref value (0 is valid), an endpoint reference IE is included. The endpoint reference IE indicates that this is a point-to-multipoint call; point-to-multipoint calls will be supported starting with the second release of SunATM software.

qcc_bld(3)

`qcc_bld_call_proceeding()` includes a connection identifier IE if a positive `vci` is passed in, and an endpoint reference IE if a non-negative `endpt_ref` is passed in. An endpoint reference IE should only appear if the call is a point-to-multipoint call; point-to-multipoint calls will be supported starting with the second release of SunATM software.

`qcc_bld_connect()` includes an AAL parameters IE, requiring the `forward_` and `backward_sdusize` values, a connection identifier IE if a positive `vci` value is passed in, and an end-point reference IE if a non-negative `endpt_ref` value is passed in. An endpoint reference IE should only appear if the call is a point-to-multipoint call; point-to-multipoint calls will be supported starting with the second release of SunATM software.

`qcc_bld_release()` includes a cause IE for which the user must pass in a cause value. The possible values can be found in the `<atm/qcc.h>` header file. The same is true for `qcc_bld_release_complete()`.

`qcc_bld_status_enquiry()` includes only an endpoint reference IE if a non-negative `endpt_ref` value is passed in. An end-point reference IE should only appear if the call is a point-to-multipoint call; point-to-multipoint calls will be supported starting with the second release of SunATM software.

`qcc_bld_status()` includes a call state IE, requiring the user pass in the `callstate` parameter; possible values can be found in the `<atm/qcc.h>` header file. It also includes a cause IE; the cause value must also be passed in. Its possible values may also be found in the `<atm/qcc.h>` header file. Finally, if the call is a point-to-multipoint call, endpoint reference and endpoint state IEs may also be included; they are included if a non-negative `endpt_ref` value is passed in. The `endpt_state` parameter is used in the endpoint state IE; possible party state values may be found in `<atm/qcc.h>`. Point-to-multipoint calls will be supported starting with the second release of SunATM software.

`qcc_bld_restart()` includes a restart indicator IE, which is used to determine whether an individual call or all calls on an interface should be restarted. If `rstall` is 0, only the call identified by `vci` should be restarted; in this case, a connection identifier IE will also be included. If `rstall` is non-zero, all calls will be restarted. The same format applies to the `qcc_bld_restart_ack()` function.

qcc_bld(3)

RETURN VALUES

All functions return 0 on success and -1 on error.

EXAMPLES

The following code fragment builds a setup message and sends it downstream.

```
#include <atm/limits.h>
#include <atm/qcc.h>

char  ifname[QCC_MAX_IFNAME_LEN] = "sa0";
int   calltag = 0x1234;
int   vci = 0x100;
int   forward_sdusize = 0x2378;
int   backward_sdusize = 0x2378;
int   sap = 0x100;

atm_addr_t  src_addr = {
    0x45, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x0f, 0x00, 0x00, 0x00, 0x00,
    0x08, 0x00, 0x20, 0x1a, 0xe1, 0x53, 0x00
};

atm_addr_t  dst_addr = {
    0x45, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x0f, 0x00, 0x00, 0x00, 0x00,
    0x08, 0x00, 0x20, 0x1a, 0xb6, 0xb9, 0x00
};

struct strbuf  ctl, data;
char          ctlbuf[QCC_MAX_CTL_LEN];
char          databuf[QCC_MAX_DATA_LEN];

ctl.buf = ctlbuf;
data.buf = databuf;
ctl.maxlen = QCC_MAX_CTL_LEN;
data.maxlen = QCC_MAX_DATA_LEN;
```

qcc_bld(3)

```
qcc_bld_setup(&ctl, &data, ifname, calltag, vci,
             forward_sdusize, backward_sdusize,
             &src_addr, &dst_addr, sap, -1);

if (putmsg(fd, &ctl, &data, 0) < 0) {
    perror("putmsg");
    exit (-1);
}
```

SEE ALSO

qcc_len(3), qcc_parse(3), qcc_util(3), q93b(7)

"ATM User-Network Interface Specification, V3.0," ATM Forum.

NOTES

The functions in this API include support for Information Elements specific to point-to-multipoint calls. Although point-to-multipoint calls are not supported in the first release of the SunATM software, they will be supported in future releases; thus the necessary parameters were included in the API functions of the first release to avoid changes to the API when point-to-multipoint support is added to the SunATM software package.

This API is an interim solution until the ATM Forum has standardized an API. At that time, Sun will implement that API, and support for the Q.2931 Call Control library may not be continued.

qcc_len(3)

C Library Functions

qcc_len(3)

NAME

qcc_bld_setup_dataalen, qcc_bld_call_proceeding_dataalen,
qcc_bld_connect_dataalen, qcc_bld_connect_ack_dataalen,
qcc_bld_release_dataalen, qcc_bld_release_complete_dataalen,
qcc_bld_status_enquiry_dataalen, qcc_bld_status_dataalen,
qcc_bld_restart_dataalen, qcc_bld_restart_ack_dataalen
qcc_max_bld_dataalen, qcc_ctl_len – get length of Q.2931 messages

SYNOPSIS

```
cc [ flag ... ] file ... -latm [ library ... ]  
  
#include <atm/qcc.h>  
#include <atm/limits.h>  
  
size_t qcc_bld_setup_dataalen();  
  
size_t qcc_bld_call_proceeding_dataalen();  
  
size_t qcc_bld_connect_dataalen();  
  
size_t qcc_bld_connect_ack_dataalen();  
  
size_t qcc_bld_release_dataalen();  
  
size_t qcc_bld_release_complete_dataalen();  
  
size_t qcc_bld_status_enquiry_dataalen();  
  
size_t qcc_bld_status_dataalen();  
  
size_t qcc_bld_restart_dataalen();  
  
size_t qcc_bld_restart_ack_dataalen();  
  
size_t qcc_max_bld_dataalen();  
  
size_t qcc_ctl_len();
```

MT-LEVEL

Safe.

qcc_len(3)

AVAILABILITY

The functionality described in this man page is available in the SUNWatm package included with a SunATM adapter board. The libatm.a library, which is located in /usr/lib, must be included at compile time as indicated in the synopsis.

DESCRIPTION

These functions may be used to determine appropriate buffer sizes for the control and data buffers that are passed into qcc_bld(3) functions. For the data buffer, the qcc_bld*_datalen() functions will return the maximum size of a particular message type. qcc_max_bld_datalen() returns the maximum size of all Q.2931 message types. A buffer allocated for this size will be able to hold any message type. For the control buffer, qcc_ctl_len() will return the required size.

SEE ALSO

qcc_bld(3), qcc_parse(3), q93b(7)

ATM User-Network Interface Specification, V3.0, published by the ATM Forum. ISBN 0-13-225863-3

NAME

qcc_parse, qcc_parse_setup, qcc_parse_call_proceeding, qcc_parse_connect, qcc_parse_release, qcc_parse_release_complete, qcc_parse_status_enquiry, qcc_parse_status, qcc_parse_restart, qcc_parse_restart_ack, qcc_get_hdr
– parse Q.2931 messages

SYNOPSIS

```
cc [ flag ... ] file ... -latm [ library ... ]
```

```
#include <atm/types.h>
```

```
#include <atm/qcc.h>
```

```
int qcc_parse_setup(strbuf_t *datap, int *vcip,  
int *forward_sdusizep, int *backward_sdusizep,  
atm_addr_t *src_addrp, atm_addr_t *dst_addrp,  
int *sapp, int *endpt_refp);
```

```
int qcc_parse_call_proceeding(strbuf_t *datap, int *vcip,  
int *endpt_refp);
```

```
int qcc_parse_connect(strbuf_t *datap, int *vcip,  
int *forward_sdusizep, int *backward_sdusizep,  
int *endpt_refp);
```

```
int qcc_parse_release(strbuf_t *datap, int *causep);
```

```
int qcc_parse_release_complete(strbuf_t *datap,  
int *causep);
```

```
int qcc_parse_status_enquiry(strbuf_t *datap,  
int *endpt_refp);
```

```
int qcc_parse_status(strbuf_t *datap, int *callstatep,  
int *causep, int *endpt_refp, int *endpt_statep);
```

```
int qcc_parse_restart(strbuf_t *datap, int *vcip,  
int *rstallp);
```

qcc_parse(3)

```
int qcc_parse_restart_ack(strbuf_t *datap, int *vcip,  
int *rstallp);
```

```
qcc_hdr_t *qcc_get_hdr(strbuf_t *ctlp)
```

MT-LEVEL

Safe.

AVAILABILITY

The functionality described in this man page is available in the SUNWatma package included with the SunATM adapter board. The libatm.a library, which is located in /usr/lib, must be included at compile time as indicated in the synopsis.

DESCRIPTION

These functions parse the various messages that make up the Q.2931 protocol which is used for ATM signalling. A full description of the message format and use can be found in the ATM Forum's User Network Interface Specification, V3.0. The functions may be used by processes which are running in user space.

Each function requires a minimum of 1 parameter: datap, which is a pointer to a strbuf_t buffer, or in the case of qcc_get_hdr, ctlp, which is also a pointer to a strbuf_t buffer.

datap is the data portion of a STREAMS message, corresponding to the M_DATA block of the message that is received from downstream. After receiving a message using the getmsg(2) function, the message type may be examined and an appropriate parsing routing called to extract information from the signalling message.

qcc_parse(3)

ctlp is the control portion of a STREAMS message, corresponding to the M_PROTO block of the message that is received from downstream. After receiving a message using the getmsg(2) function, qcc_get_hdr may be used to extract the Q.2931 header structure from the control buffer received from getmsg(2). The Q.2931 header type, qcc_hdr_t, is defined in <atm/types.h>.

Other parameters for each function depend on the type of information that is available in each message type. In all cases, certain IEs are examined in each message, as indicated below. If those IEs exist, the data that is expected from them is retrieved, but no error message is sent if they do not exist; the value of the parameter is set to -1 for any data that was expected from that particular IE. Also, IEs that are not expected are ignored. If the user wishes to ignore any of the parameters of a parse function, passing in a NULL pointer for that parameter is allowed so that space need not be allocated for the unnecessary parameter.

qcc_parse_setup() parses a setup message containing the following Information Elements: AAL parameters, ATM user cell rate, broadband bearer capability, called party number, calling party number, quality of service parameter, connection identifier, broadband higher layer information, and endpoint reference. The endpoint reference IE is only included in setup messages for point-to-multipoint calls, which will be supported starting with the second release of SunATM software. The following table matches the data that is retrieved from the message with the IE from which it is parsed.

DATA RETRIEVED	INFORMATION ELEMENT
vci	connection identifier
forward sdu size	AAL parameters
backward sdu size	AAL parameters
source address	calling party number
destination address	called party number
sap	broadband higher layer
endpoint reference id	endpoint reference

qcc_parse(3)

qcc_parse_call_proceeding() parses a call proceeding message containing a connection identifier IE, which is used to set the value of vci, and an endpoint reference IE, setting the value of endpt_ref. The endpoint reference IE is only included in call proceeding messages for point-to-multipoint calls, which will be supported starting with the second release of SunATM software.

qcc_parse_connect() parses a connect message containing an AAL parameters IE, setting the forward and backward sdu size values, a connection identifier IE, setting the value of vci, and an endpoint reference IE, setting the value of endpt_ref. The endpoint reference IE is only included in connect messages for point-to-multipoint calls, which will be supported starting with the second release of SunATM software.

qcc_parse_release() parses a cause IE, setting the cause value. A listing of the possible values can be found in the <atm/qcc.h> header file. The same is true for qcc_parse_release_complete.

qcc_parse_status_enquiry() parses a status enquiry message containing an endpoint reference IE, setting the value of endpt_ref. The endpoint reference IE is only included when enquiring about a party state in a point-to-multipoint call. Point-to-multipoint calls will be supported starting with the second release of SunATM software.

qcc_parse_status() parses a status message. The IEs that are parsed are call state, cause, endpoint reference, and endpoint state. The call state and cause IEs are used to set the value of the parameters callstate and cause; possible values for both parameters may be found in the <atm/qcc.h> header file. The endpoint reference and endpoint state IEs will be used to set the values of the endpt_ref and endpt_state parameters; they are included if an enquiry is made about a party state in a point-to-multipoint call or to report an error condition in a point-to-multipoint call. Point-to-multipoint calls will be supported starting with the second release of SunATM software.

qcc_parse(3)

`qcc_parse_restart()` parses a restart message containing two possible IEs: connection identifier and restart indicator. The restart indicator IE is used to set the value of `rstall`; this parameter indicates whether a particular `vci` or all `vcis` are to be restarted (`rstall = 1` implies all `vcis`, `rstall = 0` implies a particular `vci`). The connection identifier identifies the particular `vci`. In this case, the value of the parameter `vci` is set to 0 if there is no connection identifier IE in the message. The same format applies to the `qcc_parse_restart_ack()` function.

`qcc_get_hdr()` extracts the Q.2931 header from the control buffer received in `getmsg(2)`. A pointer to this buffer, `ctlp`, is passed in to the function, and a pointer to the header of type `qcc_hdr_t` is returned on success. On failure, a null pointer is returned.

RETURN VALUES

All functions, with the exception of `qcc_get_hdr`, return 0 on success and -1 on error. The return values for `qcc_get_hdr` are described above.

qcc_parse(3)

EXAMPLES

The following code fragment receives and parses a setup message.

```
#include <atm/types.h>
#include <atm/qcc.h>
#include <atm/limits.h>

void
wait_for_setup(int fd);
{
    int        vci;
    int        forward_sdusize;
    int        backward_sdusize;
    int        sap;
    int        flags = 0;
    atm_addr_t src_addr;
    atm_addr_t dst_addr;
    qcc_hdr_t  *hdrp;
    struct strbuf ctl, data;
    char        ctlbuf[QCC_MAX_CTL_LEN];
    char        databuf[QCC_MAX_DATA_LEN];

    ctl.buf = ctlbuf;
    data.buf = databuf;
    ctl.len = data.len = 0;
    ctl.maxlen = QCC_MAX_CTL_LEN;
    data.maxlen = QCC_MAX_DATA_LEN;

    if (getmsg(fd, &ctl, &data, &flags) < 0) {
        perror("getmsg");
        exit (-1);
    }

    hdrp = qcc_get_hdr(&ctl);
```

qcc_parse(3)

```
if ((hdrp) && (hdrp->type == QCC_SETUP)) {
    qcc_parse_setup(&data, &vci, &forward_sdu_size,
        &backward_sdu_size, &src_addr,
        &dst_addr, &sap, NULL);
    printf("parse_setup: vci = 0x%x, sap = 0x%x\n",
        vci, sap);
}
}
```

SEE ALSO

qcc_bld(3), qcc_len(3), qcc_util(3), q93b(7)

"ATM User-Network Interface Specification, V3.0," ATM Forum.

NOTES

The functions in this API include support for Information Elements specific to point-to-multipoint calls. Although point-to-multipoint calls are not supported in the first release of the SunATM software, they will be supported in future releases; thus the necessary parameters were included in the API functions of the first release to avoid changes to the API when point-to-multipoint support is added to the SunATM software package.

This API is an interim solution until the ATM Forum has standardized an API. At that time, Sun will implement that API, and support for the Q.2931 Call Control library may not be continued.

qcc_util(3)

C Library Functions

qcc_util(3)

NAME

q_ioc_bind

SYNOPSIS

```
cc [ flag ... ] file ... -latm [ library ... ]
```

```
#include <atm/qccioctl.h>
```

```
int q_ioc_bind(int fd, int sap);
```

MT-LEVEL

Safe.

AVAILABILITY

The functionality described in this man page is available in the SUNWatm package included with a SunATM adapter board. The libatm.a library, which is located in /usr/lib, must be included at compile time as indicated in the synopsis.

DESCRIPTION

This utility may be used to bind an application program that has opened a stream to the q93b driver. This step is required so that incoming SETUP messages are directed to the correct application by the q93b driver.

Before using this function, a stream must be opened to the q93b driver, using the open(2) system call.

q_ioc_bind() may then be used to bind a service access point, sap, to the opened stream, specified by its file descriptor, fd.

Q.2931 SETUP messages which are to be received by the application program must contain a Broadband Higher Layer Information IE identifying the sap to which the message should be directed.

RETURN VALUES

q_ioc_bind returns -1 on error and 0 on success.

qcc_util(3)**EXAMPLES**

The following example opens a stream to q93b and binds it to sap 0x100.

```
#include <atm/qccioctl.h>

setup_q93b()
{
    char    qdriver[] = "/dev/q93b"
    int     qfd;
    int     sap = 0x100;

    if ((qfd = open(qdriver, O_RDWR, 0)) < 0) {
        perror("q93b open");
        exit(-1);
    }

    if (q_ioc_bind(qfd, sap) < 0) {
        perror("q_ioc_bind");
        exit(-1);
    }
}
```

SEE ALSO

qcc_bld(3), qcc_len(3), qcc_parse(3), qcc_bld(9F),
qcc_parse(9F), sa_util(3), q93b(7), sa(7)

D.1.3 Q.93B Kernel Space API

Q.93B kernel space consists of all `qcc_bld` and `qcc_parse` functions under section 9F.

See the `qcc_bld` (9F) and `qcc_parse` (9F) man pages that follow.

qcc_bld(9F)**C Library Functions****qcc_bld(9F)****NAME**

qcc_bld, qcc_bld_setup, qcc_bld_call_proceeding, qcc_bld_connect,
qcc_bld_release, qcc_bld_release_complete, qcc_bld_status,
qcc_bld_status_enquiry, qcc_bld_restart, qcc_bld_restart_ack
– build Q.2931 messages

SYNOPSIS

```
cc -DKERNEL -D_KERNEL [ flag ... ] file ...
```

```
#include <atm/types.h>
```

```
#include <atm/qcc.h>
```

```
char _depends_on[] = "drv/qcc";
```

```
int qcc_parse_setup(mblk_t *mp, int *vcip,  
    int *forward_sdusizep, int *backward_sdusizep,  
    atm_addr_t *src_addrp, atm_addr_t *dst_addrp,  
    int *sapp, int *endpt_refp);
```

```
int qcc_parse_call_proceeding(mblk_t *mp, int *vcip,  
    int *endpt_refp);
```

```
int qcc_parse_connect(mblk_t *mp, int *vcip,  
    int *forward_sdusizep, int *backward_sdusizep,  
    int *endpt_refp);
```

```
int qcc_parse_release(mblk_t *mp, int *causep);
```

```
int qcc_parse_release_complete(mblk_t *mp,  
    int *causep);
```

```
int qcc_parse_status_enquiry(mblk_t *mp,  
    int *endpt_refp);
```

```
int qcc_parse_status(mblk_t *mp, int *callstatep,  
    int *causep, int *endpt_refp, int *endpt_statep);
```

qcc_bld(9F)

```
int qcc_parse_restart(mblk_t *mp, int *vcip,  
int *rstallp);
```

```
int qcc_parse_restart_ack(mblk_t *mp, int *vcip,  
int *rstallp);
```

MT-LEVEL

Safe.

AVAILABILITY

The functionality described in this man page is available in the SUNWatma package included with the SunATM adapter board. The `-DKERNEL` and `-D_KERNEL` flags must be included to indicate that the application should run in kernel space, and the qcc driver must be loaded (this requirement is expressed in the code using the "depends_on" line shown in the synopsis).

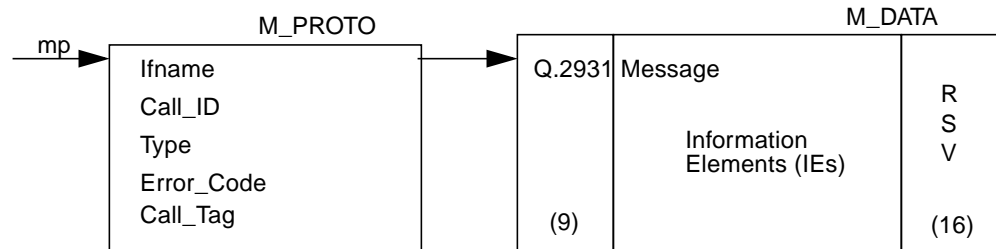
DESCRIPTION

These functions build the various messages that make up the Q.2931 protocol which is used for ATM signalling. A full description of the message format and use can be found in the ATM Forum's User Network Interface Specification, V3.0. The functions may be used by processes which are running in kernel space.

In general, no error checking is performed on the data that is passed in. Whatever data is passed in will be placed in the message that is built without examination. The only exceptions to this are mentioned in the function descriptions.

Two `mblk_t` structures are allocated and linked by each of the functions (their format is shown in the following diagram). The pointer that is returned points to the `M_PROTO` block, and may then be passed downstream with the `putq(9F)` command.

qcc_bld(9F)



The parameters passed in to each function are used to fill in the data portions of these two mblks.

Each function requires a minimum of 2 parameters: `ifname`, which is a string containing the physical interface (such as `sa0`); and an integer, either `calltag` or `callid`, depending on the message type. `calltag` is used in the setup message only; it is a reference number that is assigned by the calling application. `callid` is used in all other messages; it is assigned by the lower layer and will be sent up to the user, with the `calltag`, in the `setup_ack` message.

Other parameters for each function depend on the type of information required for each message type, and are defined in the paragraphs describing each function call.

`qcc_bld_setup()` constructs a setup message containing the following Information Elements: AAL parameters, ATM user cell rate, broadband bearer capability, called party number, calling party number, quality of service parameter, and endpoint reference. The user must pass in the forward and backward sdu sizes for the AAL parameter IE, an ATM address for the destination for the called party number IE, and one for itself for the calling party number IE (`atm_address_t` format is defined in the `<atm/qcc.h>` header file). The value passed in the `sap` parameter is placed in a broadband higher layer IE. The higher layer IE indicates the `sap` to which received messages should be directed. If the user passes in a positive `vci`, a connection identifier IE will be included; if the user passes in a non-negative `endpt_ref` (0 is valid), an endpoint reference IE will be included.

qcc_bld(9F)

The endpoint reference IE indicates that this is a point-to-multipoint call; point-to-multipoint calls will be supported starting with the second release of SunATM software.

`qcc_bld_call_proceeding()` includes a connection identifier IE if a positive `vci` is passed in, and an endpoint reference IE if a non-negative `endpt_ref` is passed in. An endpoint reference IE should only appear if the call is a point-to-multipoint call; point-to-multipoint calls will be supported starting with the second release of SunATM software.

`qcc_bld_connect()` includes an AAL parameters IE, requiring the `forward_` and `backward_sdusize` values, a connection identifier IE if a positive `vci` value is passed in, and an endpoint reference IE if a non-negative `endpt_ref` value is passed in. An endpoint reference IE should only appear if the call is a point-to-multipoint call; point-to-multipoint calls will be supported starting with the second release of SunATM software.

`qcc_bld_release()` includes a cause IE for which the user must pass in a cause value. The possible values can be found in the `<atm/qcc.h>` header file. The same is true for `qcc_bld_release_complete()`.

`qcc_bld_status_enquiry()` includes only an endpoint reference IE if a non-negative `endpt_ref` value is passed in. An endpoint reference IE should only appear if the call is a point-to-multipoint call; point-to-multipoint calls will be supported starting with the second release of SunATM software.

`qcc_bld_status()` includes a call state IE, requiring the user pass in the `callstate` parameter; possible values can be found in the `<atm/qcc.h>` header file. It also includes a cause IE; the cause value must also be passed in. Its possible values may also be found in the `<atm/qcc.h>` header file. Finally, if the call is a point-to-multipoint call, endpoint reference and endpoint state IEs may also be included; they are included if a non-negative `endpt_ref` value is passed in. The `endpt_state` parameter is used in the endpoint state IE; possible party state values may be found in `<atm/qcc.h>`. Point-to-multipoint calls will be supported starting with the second release of SunATM software.

qcc_bld(9F)

qcc_bld_restart() includes a restart indicator IE, which is used to determine whether an individual call or all calls on an interface should be restarted. If rstall is 0, only the call identified by vci should be restarted; in this case, a connection identifier IE will also be included. If rstall is non-zero, all calls will be restarted. The same format applies to the qcc_bld_restart_ack() function.

RETURN VALUES

All functions return a pointer to an mblk_t. If the function is not successful, the pointer will be NULL.

EXAMPLES

The following code fragment builds a setup message and sends it downstream.

```
#include <sys/stream.h>
#include <atm/qcc.h>
#include <atm/limits.h>

char  _depends_on[] = "drv/qcc";

void
wait_for_setup(queue_t *q);
{
    int      vci;
    int      forward_sdusize;
    int      backward_sdusize;
    int      sap;
    atm_addr_t  src_addr;
    atm_addr_t  dst_addr;
    mblk_t      *mp;
    qcc_hdr_t   *hdrp;
```


qcc_bld(9F)

```
do {
    if !(mp = getq(q)) {
        perror("getq");
        exit (-1);
    }
    hdrp = (qcc_hdr_t *)mp;
} while (hdrp->type != QCC_SETUP);

qcc_parse_setup(mp->b_cont, &vci, &forward_sdusize,
               &backward_sdusize, &src_addr,
               &dst_addr, &sap, NULL);
printf("parse_setup: vci = 0x%x, sap = 0x%x0, vci, sap);
}
```

SEE ALSO

qcc_util(3), qcc_parse(9F), q93b(7)

"ATM User-Network Interface Specification, V3.0," ATM Forum.

NOTES

The functions in this API include support for Information Elements specific to point-to-multipoint calls. Although point-to-multipoint calls are not supported in the first release of the SunATM software, they will be supported in future releases; thus the necessary parameters were included in the API functions of the first release to avoid changes to the API when point-to-multipoint support is added to the SunATM software package.

This API is an interim solution until the ATM Forum has standardized an API. At that time, Sun will implement that API, and support for the Q.2931 Call Control library may not be continued.

NAME

qcc_parse, qcc_parse_setup, qcc_parse_call_proceeding, qcc_parse_connect, qcc_parse_release, qcc_parse_release_complete, qcc_parse_status_enquiry, qcc_parse_status, qcc_parse_restart, qcc_parse_restart_ack
– parse Q.2931 messages

SYNOPSIS

```
cc -DKERNEL -D_KERNEL [ flag ... ] file ...
```

```
#include <atm/types.h>
```

```
#include <atm/qcc.h>
```

```
char _depends_on[] = "drv/qcc";
```

```
int qcc_parse_setup(mblk_t *mp, int *vcip,  
    int *forward_sdusizep, int *backward_sdusizep,  
    atm_addr_t *src_addrp, atm_addr_t *dst_addrp,  
    int *sapp, int *endpt_refp);
```

```
int qcc_parse_call_proceeding(mblk_t *mp, int *vcip,  
    int *endpt_refp);
```

```
int qcc_parse_connect(mblk_t *mp, int *vcip,  
    int *forward_sdusizep, int *backward_sdusizep,  
    int *endpt_refp);
```

```
int qcc_parse_release(mblk_t *mp, int *causep);
```

```
int qcc_parse_release_complete(mblk_t *mp,  
    int *causep);
```

```
int qcc_parse_status_enquiry(mblk_t *mp,  
    int *endpt_refp);
```

```
int qcc_parse_status(mblk_t *mp, int *callstatep,  
    int *causep, int *endpt_refp, int *endpt_statep);
```

qcc_parse(9F)

```
int qcc_parse_restart(mblk_t *mp, int *vcip,
                    int *rstallp);

int qcc_parse_restart_ack(mblk_t *mp, int *vcip,
                        int *rstallp);
```

MT-LEVEL

Safe.

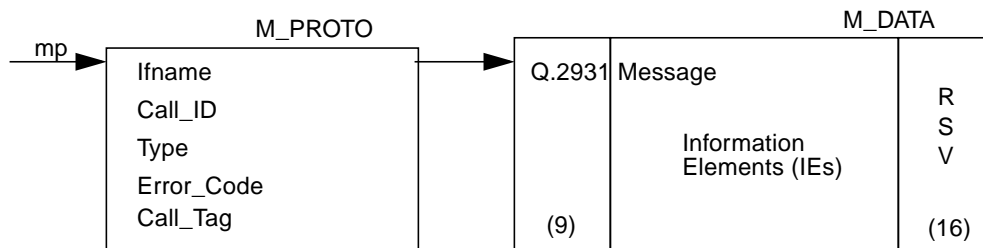
AVAILABILITY

The functionality described in this man page is available in the SUNWatma package included with the SunATM adapter board. The -DKERNEL and -D_KERNEL flags must be included to indicate that the application should run in kernel space, and the qcc driver must be loaded (this requirement is expressed in the code using the "depends_on" line shown in the synopsis).

DESCRIPTION

These functions parse the various messages that make up the Q.2931 protocol which is used for ATM signalling. A full description of the message format and use can be found in the ATM Forum's User Network Interface Specification, V3.0. The functions may be used by processes which are running in kernel space.

Each function requires a minimum of 1 parameter: mp, which is a pointer to a mblk_t structure, and is extracted from the following structure:



qcc_parse(9F)

When a message is received from the q93b driver using the `getq(9F)` function, a pointer to the `M_PROTO` block shown above is returned. However, the q93b message which is parsed is contained in the `M_DATA` block, so the first parameter passed to a `qcc_parse` function must be `mp->b_cont`, where `mp` is the pointer received by `getq()`. The `M_PROTO` block data may be examined to determine the message type, which indicates the parsing function that should be called.

Other parameters for each function depend on the type of information that is available in each message type. In all cases, certain IEs are examined in each message, as indicated below. If those IEs exist, the data that is expected from them is retrieved, but no error message is sent if they do not exist; the value of the parameter is set to -1 for any data that was expected from that particular IE. Also, IEs that are not expected are ignored. If the user wishes to ignore any of the parameters of a parse function, passing in a `NULL` pointer for that parameter is allowed so that space need not be allocated for the unnecessary parameter.

`qcc_parse_setup()` parses a setup message containing the following Information Elements: AAL parameters, ATM user cell rate, broadband bearer capability, called party number, calling party number, quality of service parameter, connection identifier, broadband higher layer information, and endpoint reference. The endpoint reference IE is only included in setup messages for point-to-multipoint calls, which will be supported starting with the second release of SunATM software. The following table matches the data that is retrieved from the message with the IE from which it is parsed.

DATA RETRIEVED	INFORMATION ELEMENT
<code>vci</code>	connection identifier
<code>forward sdu size</code>	AAL parameters
<code>backward sdu size</code>	AAL parameters
<code>source address</code>	calling party number
<code>destination address</code>	called party number
<code>sap</code>	broadband higher layer
<code>endpoint reference id</code>	endpoint reference

qcc_parse(9F)

`qcc_parse_call_proceeding()` parses a call proceeding message containing a connection identifier IE, which is used to set the value of `vci`, and an endpoint reference IE, setting the value of `endpt_ref`. The endpoint reference IE is only included in call proceeding messages for point-to-multipoint calls, which will be supported in the second release of SunATM software.

`qcc_parse_connect()` parses a connect message containing an AAL parameters IE, setting the forward and backward `sdu` size values, a connection identifier IE, setting the value of `vci`, and an endpoint reference IE, setting the value of `endpt_ref`. The endpoint reference IE is only included in connect messages for point-to-multipoint calls, which will be supported starting with the second release of SunATM software.

`qcc_parse_release()` parses a cause IE, setting the cause value. A listing of the possible values can be found in the `<atm/qcc.h>` header file. The same is true for `qcc_parse_release_complete`.

`qcc_parse_status_enquiry()` parses a status enquiry message containing an endpoint reference IE, setting the value of `endpt_ref`. The endpoint reference IE is only included when enquiring about a party state in a point-to-multipoint call. Point-to-multipoint calls will be supported starting with the second release of SunATM software.

`qcc_parse_status()` parses a status message. The IEs that are parsed are call state, cause, endpoint reference, and endpoint state. The call state and cause IEs are used to set the values of the parameters `callstate` and `cause`; possible values for both parameters may be found in the `<atm/qcc.h>` header file. The endpoint reference and endpoint state IEs will be used to set the values of the `endpt_ref` and `endpt_state` parameters; they are included if an enquiry is made about a party state in a point-to-multipoint call or to report an error condition in a point-to-multipoint call. Point-to-multipoint calls will be supported starting with the second release of SunATM software.

qcc_parse(9F)

`qcc_parse_restart()` parses a restart message containing two possible IEs: connection identifier and restart indicator. The restart indicator IE is used to set the value of `rstall`; this parameter indicates whether a particular `vci` or all `vcis` are to be restarted (`rstall = 1` implies all `vcis`, `rstall = 0` implies a particular `vci`). The connection identifier identifies the particular `vci`. In this case, the value of the parameter `vci` is set to 0 if there is no connection identifier IE in the message. The same format applies to the `qcc_parse_restart_ack()` function.

RETURN VALUES

All functions return 0 on success and -1 on error.

EXAMPLES

The following code fragment receives and parses a setup message.

```
#include <sys/stream.h>
#include <atm/qcc.h>
#include <atm/limits.h>

char  _depends_on[] = "drv/qcc";

void
wait_for_setup(queue_t *q);
{
    int      vci;
    int      forward_sdusize;
    int      backward_sdusize;
    int      sap;
    atm_addr_t  src_addr;
    atm_addr_t  dst_addr;
    mblk_t      *mp;
    qcc_hdr_t    *hdrp;
```

qcc_parse(9F)

```
do {
    if !(mp = getq(q)) {
        perror("getq");
        exit (-1);
    }
    hdrp = (qcc_hdr_t *)mp;
} while (hdrp->type != QCC_SETUP);

qcc_parse_setup(mp->b_cont, &vci, &forward_sdusize,
               &backward_sdusize, &src_addr,
               &dst_addr, &sap, NULL);
printf("parse_setup: vci = 0x%x, sap = 0x%x0, vci, sap);
}
```

SEE ALSO

qcc_util(3), qcc_bld(9F), q93b(7)

"ATM User-Network Interface Specification, V3.0," ATM Forum.

NOTES

The functions in this API include support for Information Elements specific to point-to-multipoint calls. Although point-to-multipoint calls are not supported in the first release of the SunATM software, they will be supported in future releases; thus the necessary parameters were included in the API functions of the first release to avoid changes to the API when point-to-multipoint support is added to the SunATM software package.

This API is an interim solution until the ATM Forum has standardized an API. At that time, Sun will implement that API, and support for the Q.2931 Call Control library may not be continued.

D.2 Driver API

The driver supports the ATM-specific ioctls described below. Definitions for the ioctl commands and structures can be found in <atm/saioc.h>.

sa(7) **Special Files** **sa(7)**

NAME

sa – Sun ATM device driver

SYNOPSIS

```
#include <sys/stropts.h>
#include <atm/sa.h>
#include <atm/saioc.h>
```

DESCRIPTION

The sa driver is a Solaris 2.x DDI/DKI compliant MT safe STREAMS device driver. It presents a DLPI interface to the upper layers and supports M_DATA fastpath and M_DATA raw. The hardware interface supports the SunATM-155 Fiber and UTP products.

The two modes of operation that should be used by application programs are raw mode and dlpi mode. The mode is specified by the choice of encapsulation indicated with the A_ADDVC ioctl call. NULL encapsulation indicates raw mode, while LLC encapsulation indicates dlpi mode. The mode chosen defines the format in which data should be sent to the driver.

Raw mode implies that only a single mblock will be sent to the driver, containing a four-byte vpci followed by the data. When a message is received on a vpci running in raw mode, it will be directed to upper layers based on the vpci. The four-byte vpci will be sent up with the data if an ioctl call setting DLIOCRAW has been made; if DLIOCRAW has not been set, the vpci will be stripped and only the data will be sent up.

DLPI mode implies that two mblocks will be sent to the driver. The first, of type M_PROTO, contains the dlpi message type, which is dl_unitdata_req for transmit and dl_unitdata_ind for receive. The vpci is included in this mblock as well; the format for the mblock is defined in the header file <sys/dlpi.h>. The second mblock is of type M_DATA and contains the message. When the driver gets the two mblocks from the upper layer, it will remove the first mblock, add a LLC header containing the sap which has

sa(7)

been bound to this stream (by passing down a `dl_bind_req` message) to the `M_DATA` mblock, and transmit it. On receive, the LLC header is stripped, the `M_PROTO` mblock is added, and the two-mblock structure is sent up the stream indicated by the `sap` in the LLC header.

The driver supports several of the DLPI message types defined in the `<sys/dlpi.h>` header file. Specifically, users of the `sa` driver may use the `DL_ATTACH_REQ`, `DL_DETACH_REQ`, `DL_BIND_REQ`, `DL_UNBIND_REQ`, `DL_UNITDATA_IND`, and `DL_UNITDATA_REQ`. In addition, a Sun-specific `dlpi` ioctl is supported, `DLIOCRAW`. There is no data structure associated with the `DLIOCRAW` ioctl; simply a `strioc` struct with `ic_cmd` set to `DLIOCRAW` may be used to set a stream to raw mode.

The driver also supports the ATM-specific ioctls described below. Definitions for the ioctl commands and structures may be found in `<atm/saioc`.h>.

IOCTLS

The driver supports a set of ioctl functions which are called using the `I_STR` ioctl and `strioc` structure as the argument. See the `streamio(7)` man page and the `<sys/stropts.h>` header file for more information on this type of ioctl call.

The commands supported in the `ic_cmd` field of the `strioc` structure are described in the following paragraphs. The structures that the `ic_dp` field should point to are also described for each command.

sa(7)

A_ALLOCBW	Allocate bandwidth for this stream. <code>ic_dp</code> should point to an <code>a_allocbw_t</code> structure, which is defined as: <pre>typedef struct { int bw; } a_allocbw_t;</pre>
A_RELSEBW	Release bandwidth that was previously allocated for this stream. <code>ic_dp</code> should point to an <code>a_allocbw_t</code> structure.
A_ADDVC	Add a <code>vpci</code> to those serviced by this stream, and specify the encapsulation type. The encapsulation type defines the format in which data will be sent to the driver: raw mode, indicated by <code>NULL_ENCAP</code> , implies a single mblock with only the four-byte <code>vpci</code> followed immediately by the data. <code>dlpi</code> mode, indicated by <code>LLC_ENCAP</code> , implies a two-mblock message, consisting of a <code>M_PROTO</code> mblock followed by a <code>M_DATA</code> mblock containing the data. The <code>M_PROTO</code> mblock will contain a <code>dlpi</code> message type (<code>dl_unitdata_req</code> or <code>dl_unitdata_ind</code>) and the <code>vpci</code> ; the format may be found in <code><sys/dlpi.h></code> . For the <code>A_ADDVC</code> ioctl call, <code>ic_dp</code> points to an <code>a_addvc_t</code> structure, which is defined as:

sa(7)

```

typedef struct {
    u_long vp_vc;    /* vpci to be added */
    int aal_type;   /* null -> 0,          */
                    /* AAL5 -> 5          */
    int encap;      /* encapsulation; see */
                    /* <atm/saiocctl.h> for */
                    /* possible values   */
    int buf_type;   /* if AAL5:           */
                    /* 0 -> small buf (9 k) */
                    /* 1 -> big buf (9 k)  */
                    /* 2 -> huge buf (64 k) */
                    /* if null AAL         */
                    /* -> # of cells      */
} a_addVC_t;

```

A_DELVC

Remove a vpci from those serviced by this stream.
ic_dp points to an a_delVC_t structure:

```

typedef struct {
    u_long vp_vc;
} a_delVC_t;

```

A_ALLOCBW_VC

Allocate bandwidth for a specific vpci on this stream.
ic_dp point to an a_allocbw_vc_t structure:

```

typedef struct {
    int bw; /* Mbits/sec */
    int vc; /* vpci */
} a_allocbw_vc_t;

```

A_RELSEBW_VC

Releases bandwidth that has been allocated for a specific vpci. The structure passed in ic_dp should be an a_relsebw_vc_t structure, which is typedef'ed as an a_allocbw_vc_t structure:

```

typedef a_allocbw_vc_t a_relsebw_vc_t;

```

sa(7)**EXAMPLES**

The following code fragment demonstrates opening an sa device and allocating 20 Mbits/sec of bandwidth for that stream.

```
#include <atm/saioc.h>

char      dev[0x20] = "/dev/sa0";
int       fd;
struct strioc_t  strioc;
a_allocbw_t  ap;

if ((fd = open(dev, O_RDWR)) < 0) {
    exit(-1);
}

ap.bw = 20;

strioc.ic_cmd = A_ALLOCBW;
strioc.ic_timeout = -1;
strioc.ic_len = sizeof (a_allocbw_t);
strioc.ic_dp = (caddr_t) &ap;

if (ioc(fd, I_STR, &strioc) < 0) {
    exit(-1);
}
```

SEE ALSO

sa_util(3), dlpi(7), streamio(7)

sa_util(3)

C Library Functions

sa_util(3)

NAME

sa_util, sa_open, sa_close, sa_attach, sa_detach, sa_bind, sa_unbind,
sa_setraw, sa_add_vpci, sa_delete_vpci, sa_allocate_bw, sa_release_bw
– Sun ATM driver utilities

SYNOPSIS

```
cc [ flag ... ] file ... -latm [ library ... ]  
#include <atm/sa.h>  
  
int sa_open(register char *interface);  
int sa_close(int fd);  
int sa_attach(int fd, u_long ppa, int timeout);  
int sa_detach(int fd, int timeout);  
int sa_bind(int fd, u_long sap, int timeout);  
int sa_unbind(int fd, int timeout);  
int sa_setraw(int fd);  
int sa_add_vpci(int fd, vci_t vpci, int encap, int buf_type);  
int sa_delete_vpci(int fd, vci_t vpci);  
int sa_allocate_bw(int fd, int bw);  
int sa_release_bw(int fd);
```

MT-LEVEL

Safe.

AVAILABILITY

The functionality described in this man page is available in the SUNWatma package included with a SunATM adapter board. The libatm.a library, which is located in /usr/lib, must be included at compile time as indicated in the synopsis.

sa_util(3)

DESCRIPTION

These utilities perform various operations on the SunATM device driver, sa. They may be used by application programs that need to transmit and receive data over an ATM connection to set up a data stream to the ATM driver.

Data may be transmitted over a vc connection in one of two modes: raw mode, or dlpi mode. The mode is specified by the choice of encapsulation indicated in the call to sa_add_vpci(). NULL encapsulation indicates raw mode, while LLC encapsulation indicates dlpi mode. The mode chosen defines the format in which data should be sent to the driver.

Raw mode implies that only a single mblock will be sent to the driver, containing a four-byte vpci followed by the data. When a message is received on a vpci running in raw mode, it will be directed to upper layers based on the vpci. The four-byte vpci will be sent up with the data if sa_setraw() has been called; if sa_setraw() has not been called, the vpci will be stripped and only the data will be sent up.

DLPI mode implies that two mblocks will be sent to the driver. The first, of type M_PROTO, contains the dlpi message type, which is dl_unitdata_req for transmit and dl_unitdata_ind for receive. The vpci is included in this mblock as well; its format is defined in the header file <sys/dlpi.h>. The second mblock is of type M_DATA and contains the message. When the driver gets a message of this type from the upper layer, it will remove the first mblock, add a LLC header containing the sap which has been bound to this stream using sa_bind() to the message mblock, and transmit it. On receive, the LLC header is stripped, the M_PROTO mblock is added, and the two-mblock structure is sent up the stream indicated by the sap in the LLC header.

Note – If the application is running in user space rather than kernel space, the M_PROTO and M_DATA mblocks correspond to the ctl and data buffers, respectively, which are passed into putmsg(2) or received from getmsg(2).

sa_util(3)

`sa_open()` opens a stream to the physical interface (i.e. `sa0`, `sa1`, etc.) passed in as a null-terminated string in `interface`. On success, the file descriptor (`> 0`) is returned.

`sa_close()` closes the stream specified by its file descriptor, `fd`.

`sa_attach()` associates a physical point of attachment, `ppa`, with an opened `sa` device specified by its file descriptor, `fd`. The `ppa` is usually defined as the physical interface number (0 for `sa0`, 1 for `sa1`, etc.). `timeout` may optionally be used to specify an amount of time in milliseconds to wait for the function to complete. The function will fail if it does not complete in the specified amount of time. Possible values for `timeout` are -1, which blocks until completion, 0, which returns immediately, or a number greater than 0 which specifies a number of milliseconds to wait. This value will be rounded up to an implementation-dependent minimum value, which is currently at approximately 100 ms.

`sa_detach()` detaches the stream specified by its file descriptor `fd` from its `ppa`. Values of `timeout` apply as described in `sa_attach()`.

`sa_allocate_bw()` specifies a bandwidth amount in megabits per second (in the SunATM-155 products, the user may allocate up to 135 Mbps; 2 Mbps is reserved by the ATM software stack, and 20 Mbps is lost to cell header and physical layer overhead) passed in as `bw`, that will be allocated for transmitting data from the stream identified by the file descriptor `fd`. All unallocated bandwidth is assigned to IP and `dlpi` mode traffic. This step is not necessary if a stream is only to be used to receive data; and SHALL NOT be called if a stream is using `dlpi` mode.

`sa_release_bw()` releases all bandwidth that has been previously allocated to the stream identified by `fd`.

`sa_add_vpci()` adds the given virtual path connection identifier, `vpci`, to those recognized on the specified stream (identified by its file descriptor, `fd`). The type of encapsulation that is being used on this connection must also be specified in `encap`; the possible values are `NULL_ENCAP`, `LLC_ENCAP`, and

sa_util(3)

NLPID_ENCAP, as defined in <atm/saioclt.h>. Finally, the buffer type must be specified in `buf_type`; definitions may also be found in <atm/saioclt.h> for the possible types `SMALL_BUF_TYPE`, `BIG_BUF_TYPE`, and `HUGE_BUF_TYPE`.

`sa_delete_vpci()` deletes given virtual path connection identifier, `vpci`, from the specified stream (identified by its file descriptor, `fd`).

`sa_bind()` binds a service access point, `sap`, to an opened stream, specified by its file descriptor, `fd`. `sap` values of `0x800` and `0x806` are reserved for IP and ARP traffic, respectively; the user shall not use these values. The `sap` is used by the driver to direct traffic to upper layers if LLC encapsulation is used. This function also has a timeout parameter; the values of timeout described in `sa_attach()` apply in `sa_bind()` as well.

`sa_unbind()` disassociates a stream-to-`sap` binding. The stream is specified by its file descriptor, `fd`. Values of timeout apply as described in `sa_attach()`.

`sa_setraw()` indicates to the driver that the stream specified by the file descriptor `fd` will be transmitting and receiving raw data which will be interpreted directly by the application at the stream head. The only header information included in messages passed down the stream will be the 4-byte virtual path connection identifier. When a message is received, the `vpci` will be used to direct the message to upper layers.

The ordering of the `sa` utility function calls is important. After calling `sa_open()`, the order must be `sa_attach()`, followed by `sa_allocate_bw()` if required, and `sa_add_vpci()`. Next, depending on the type of encapsulation used on this stream, should be either `sa_bind()` for LLC encapsulation or `sa_setraw()` for null encapsulation. All functions must be called only once per interface, with the exception of `sa_add_vpci()`, which may be called multiple times to support multiple `vpcis`.

RETURN VALUES

All functions return `-1` on error. With the exception of `sa_open`, which returns the file descriptor on success, all functions return `0` on success.

sa_util(3)

EXAMPLES

The following example opens a stream to sa0 and sets up that stream to communicate over vpci 0x100 at 10 Mbits/sec in raw mode.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stropts.h>
#include <sys/errno.h>
#include <atm/sa.h>

main()
{
    char    interface[] = "sa0";
    int     fd;
    int     ppa;
    int     bw = 10;
    int     vpci = 0x100;
    char    ctlbuf[256];
    char    databuf[256];
    struct strbuf  ctl, data;

    ctl.buf = ctlbuf;
    data.buf = databuf;
    ctl.maxlen = data.maxlen = 256;

    ppa = atoi(&interface[strlen (interface) - 1]);
    if ((fd = sa_open(interface)) < 0) {
        perror("open");
        exit(-1);
    }
    sa_attach(fd, ppa);

    if (sa_allocate_bw(fd, bw) < 0) {
        perror("sa_allocate_bw");
        exit(-1);
    }
}
```

sa_util(3)

```
if (sa_add_vpci(fd, vpci, NULL_ENCAP, BIG_BUF_TYPE) < 0) {
    perror("sa_add_vpci");
    exit(-1);
}
if (sa_setraw(fd) < 0) {
    perror("sa_setraw");
    exit(-1);
}

<construct a message to pass down in ctlbuf and databuf>

if (putmsg(fd, &ctl, &data, 0) < 0) {
    perror("putmsg");
    exit(-1);
}

}
```

The following example opens a stream to sa0 and sets up that stream to communicate over vpci 0x100, using sap 0x100, in dlpi mode.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stropts.h>
#include <sys/errno.h>
#include <sys/dlpi.h>
#include <atm/sa.h>

main()
{
    char    interface[] = "sa0";
    int     fd;
    int     ppa;
    int     vpci = 0x100;
    int     *vpcip;
    int     sap = 0x100;
    char    ctlbuf[256];
```

sa_util(3)

```
char  databuf[256];
struct strbuf  ctl, data;
dl_unitdata_req_t *dludp;

ctl.buf = ctlbuf;
data.buf = databuf;
ctl.maxlen = data.maxlen = 256;

ppa = atoi(&interface[strlen (interface) - 1]);
if ((fd = sa_open(interface)) < 0) {
    exit(-1);
}
sa_attach(fd, ppa);

if (sa_add_vpci(fd, vpci, LLC_ENCAP, BIG_BUF_TYPE) < 0) {
    perror("sa_add_vpci");
    exit(-1);
}
sa_bind(fd, sap);

<construct the message in databuf>

ctlLen = sizeof (dl_unitdata_req_t) + 4;
memset(ctlbuf, 0, ctlLen);
dludp = (dl_unitdata_req_t *) ctlbuf;
dludp->dlprimitive = DL_UNITDATA_REQ;
dludp->dl_dest_addr_length = 4;
dludp->dl_dest_addr_offset = sizeof (dl_unitdata_req_t);
vpcip = (int *) &ctlbuf[sizeof (dl_unitdata_req_t)];
*vpcip = vpci;

if (putmsg(fd, &ctl, &data, 0) < 0) {
    perror("putmsg");
    exit(-1);
}
}
```

sa_util(3)

SEE ALSO

dlpi(7), sa(7)

Advanced Configurations



Networks are rarely homogeneous. For interoperability purposes, there may be cases when a network must be configured:

- with different characteristics than the defaults that are built into the SunATM adapter
- with unusual addressing schemes that require more than the basic flags described in Section 3.2.1.2, “Editing the `/etc/aarconfig` File”

Edit the `/etc/aarconfig` file using the flags in this section to alter the defaults and/or change the behavior of the interface.

E.1 Flags That Specify Additional Entry Types

- b* Specifies the VCI to use for back-to-back SVC connections between two ARP clients. This entry is required in addition to the *t* entry because the VCI is normally provided by the switch. *VCI* is required. See Table E-1.
- B* Specifies the VCI to use for a back-to-back SVC connection between an ARP client and a server. This entry is required in addition to the *s* or *t* entry. *VCI* is required. See Table E-1.

Note – The *b* and *B* options are useful for testing. For normal operation, the *t* flag may be used with a VCI and no ATM address.

Note – When using SVCs over back-to-back connections, the two systems should use different VCI values. Also, if both a *b* and a *B* entry are used, they should each have different VCI values.

- c Indicates an alternate client address for ARP traffic only in the server's `aarconfig` file. There may be configurations that require an ATM ARP client to have different ATM addresses or PVCs for ARP connections and for regular data connections. In this case, a distinction must be made in the server's `/etc/aarconfig` file between the two address entries; the *c* flag specifies the ARP address, while the *t* flag identifies the data address. If both a *t* and *c* entry are provided in a server's `aarconfig` file, any of the VCIs or ATM addresses in those entries may be used by the client to contact the server. Either *ATM Address* or *VCI* is required. See Table E-1.

As an example, consider this situation: a server has a client that uses a different selector byte to identify the ARP connection. A requirement is that the ATM address end with 00 for data connections, and end with 05 for ARP connections. To represent this client, the following two entries are required in the server `/etc/aarconfig` file:

Interface	Host	ATM Address	VCI	Flag
sa0	client1	45:00:00:00:00:00:00:00:0f:00:00:00:00:00:08:00:20:13:00:10:00	-	t
sa0	client1	45:00:00:00:00:00:00:00:0f:00:00:00:00:00:08:00:20:13:00:10:05	-	c

- A Specifies on the server an alternate local ATM address for ARP traffic only. Similar to situations where a client has different addresses for data and ARP connections (see above example), it may be that the server also has different addresses. In this case, it is still sufficient to have only an *s* entry in the client `/etc/aarconfig` file since the client will be able to send ARP requests to the server for the server's data address. However, the server must be aware of the two different addresses. This is accomplished by using the *A* flag to identify the ARP address, while the *L* flag identifies the data address. *ATM Address* is required. See Table E-1.

As an example, consider a situation where the server uses an address with selector 00 for data connections and an address with selector 01 for ARP connections:

Interface	Host	ATM Address	VCI	Flag
sa0	-	45:00:00:00:00:00:00:00:0f:00:00:00:00::08:00:20:13:00:10:00	-	L
sa0	-	45:00:00:00:00:00:00:00:0f:00:00:00:00::08:00:20:13:00:10:01	64	A

Table E-1 /etc/aarconfig Advanced Configuration Flags

Interface	Host	ATM Address	VCI	Flags
required	illegal	illegal	required	b
required	illegal	illegal	required	B
required	optional	or ¹	or ¹	c
required	illegal	required	illegal	A
required	illegal	illegal	illegal	P
required	illegal	illegal	illegal	I

¹or - Means one or the other required, and both are also legal.

E.2 Flags That Change the Behavior of the Interface

- P Enables the function that sends a Call_Proceeding message when setting up a connection (this message type is optional according to the UNI 3.0 Specification). Some switches may not be designed to handle this message type since it is not required, so default behavior of the SunATM signaling is to *not send* the message. Use of the Call_Proceeding message is desirable if some amount of delay is likely in the processing of setup messages because it essentially prolongs the length of time a caller will wait to receive a connect back before giving up on the connection. If your switch supports the Call_Proceeding message, and the feature is desirable, you can turn the function on for a particular interface using the *P* flag.

- I Provides additional security. If your ARP server is capable of handling inverse ARP requests (the SunATM implementation has this capability), you may choose to have a client who receives a setup request from a peer do address verification with the server rather than the calling party. The default behavior is to send the inverse ARP for address verification to the calling party. If the *I* flag is set in the `/etc/aarconfig` file for a particular interface, the inverse ARP will be sent to the server instead. This allows the access list for the network to be specified in the server `/etc/aarconfig` file. Hosts that do not appear in this file will not be verified by the inverse ARP sent to the server, so the call will not be accepted.

The default behavior for all interfaces, rather than just one, may be set by adding one or both of the following lines to the `/etc/system` file:

```
set aar:aar_can_use_call_proc = 1

set aar:aar_invarp_to_server = 1
```

These are equivalent to providing a *P* or *I* flag entry for every interface in `/etc/aarconfig`.

Revision History

Revision	Dash	Date	Comments
801-6772-10	Rev A	March 1995	Early Access Release
801-6572-11	Rev A	May 1995	

Reader Comments

We welcome your comments and suggestions to help improve this manual. Please let us know what you think about the *SunATM-155 SBus Cards Manual*, part number 801-6572-11.

- The procedures were well documented.

Strongly
Agree

Agree

Disagree

Strongly
Disagree

Not
Applicable

Comments _____

- The tasks were easy to follow.

Strongly
Agree

Agree

Disagree

Strongly
Disagree

Not
Applicable

Comments _____

- The illustrations were clear.

Strongly
Agree

Agree

Disagree

Strongly
Disagree

Not
Applicable

Comments _____

- The information was complete and easy to find.

Strongly
Agree

Agree

Disagree

Strongly
Disagree

Not
Applicable

Comments _____

- Do you have additional comments about the *SunATM-155 SBus Cards Manual*?

Name: _____

Title: _____

Company: _____

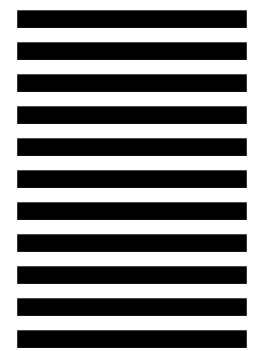
Address: _____

Telephone: _____

Email address: _____



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL
FIRST CLASS MAIL PERMIT NO. 1 MOUNTAIN VIEW, CA

POSTAGE WILL BE PAID BY ADDRESSEE



SUN MICROSYSTEMS, INC.
Attn: Manager, Hardware Publications
MS MPK 14-101
2550 Garcia Avenue
Mt. View, CA 94043-9850

