Lucent Technologies
Bell Labs Innovations

DEFINITY®
Enterprise Communications Server Release 6
CallVisor® PC ASAI Installation and Reference

555-230-227
Comcode 108111303
Issue 3
May 1998

Notice

While reasonable efforts were made to ensure that the information in this document was complete and accurate at the time of printing, Lucent Technologies can assume no responsibility for any errors. Changes and corrections to the information contained in this document may be incorporated into future reissues.

Your Responsibility for Your System's Security

You are responsible for the security of your system. Lucent Technologies does not warrant that this product is immune from or will prevent unauthorized use of common-carrier telecommunication services or facilities accessed through or connected to it. Lucent Technologies will not be responsible for any charges that result from such unauthorized use. Product administration to prevent unauthorized use is your responsibility and your system administrator should read all documents provided with this product to fully understand the features available that may reduce your risk of incurring charges.

Federal Communications Commission Statements

Part 15: Class A Statement. 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 or her own expense.

Part 68: Network Registration Number. This equipment is registered with the FCC in accordance with Part 68 of the FCC Rules. It is identified by FCC registration number AS593M-13283-MF-E.

Part 68: Answer-Supervision Signaling. Allowing this equipment to be operated in a manner that does not provide proper answer-supervision signaling is in violation of Part 68 rules. This equipment returns answer-supervision signals to the public-switched network when:

- Answered by the called station
- Answered by the attendant
- Routed to a recorded announcement that can be administered by the CPE user

This equipment returns answer-supervision signals on all DID calls forwarded back to the public-switched telephone network. Permissible exceptions are:

- A call is unanswered
- A busy tone is received
- A reorder tone is received

Trademarks

CallVisor is a registered trademark of Lucent Technologies.
DEFINITY is a registered trademark of Lucent Technologies in the USA and throughout the world.
CentreVu Computer-Telephony is a registered trademark of Lucent Technologies.
SPARC Solaris is a trademark of Sun Microsystems, Inc. in the USA and other countries.
Solaris is a trademark of Sun Microsystems, Inc. in the USA and other countries.
UNIX is a registered trademark in the USA and other countries, licensed exclusively through X/Open Company Limited.
UnixWare is a registered trademark of the Santa Cruz Operation, Inc. in the USA and other countries.
Windows is a registered trademark of Microsoft Corporation.
Windows NT is a trademark of Microsoft Corporation.

Ordering Information

The ordering number for this document is 555-230-227. To order this document, call the BCS Publications Center at 1 800 457-1235 (international callers use 1 317 322-6791). For more information about Lucent Technologies documents, refer to *Business Communications Systems Publications Catalog,* 555-000-010.

You can be placed on a Standing Order list for this and other BCS documents you may need. Standing Order will enable you to automatically receive updated versions of individual documents or document sets, billed to account information that you provide. For more information on Standing Orders, or to be put on a list to receive future issues of this document, please contact the Lucent Technologies BCS Center.

To learn more about Lucent Technologies products and to order products, contact Lucent Direct, the direct-market organization of Lucent Technologies Business Communications Systems. Access their web site at www.lucentdirect.com, or call 1 800 451-2100. The International fax number is 1 317 322-6699.

Lucent Technologies Fraud Intervention

If you suspect you are being victimized by toll fraud and you need technical support or assistance, call the BCS Technical Service Center Toll Fraud Intervention Hotline at 1 800 643-2353.

Comments

To comment on this document, return the comment card at the end of the document.

Acknowledgment

This document was prepared by the
Product Documentation Development,
Lucent Technologies
Middletown, NJ 07748-9972

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Figures

Figures

Tables

Tables

About This Document

This reference guide is for Lucent Technologies' DEFINITY® Enterprise Communications Server (ECS) CallVisor® PC Adjunct-Switch Application Interface (ASAI) platform for UNIX® and Windows® NT™.

NOTE:
This document can be used with the earlier versions of DEFINITY Communications System products.

The information in this document relates to a specific implementation of an API for the ASAI protocol between an adjunct and the DEFINITY ECS. Emphasis is on the API rather than on the protocol, adjunct, or the DEFINITY ECS. Specific information concerning these other products is readily available in other documents. See the section ''Related Documents'' at the end of this preface, or the documentation provided with your operating system, or computer.

The DEFINITY ECS utilizes the following products made by other software vendors:

- Solaris™ of Sun Microsystems, Inc.
- UNIX is licensed exclusively through X/Open Company Limited
- UnixWare® of the Santa Cruz Operation, Inc.
- Windows® of Microsoft Corporation
- Windows NT of Microsoft Corporation

Reason for Reissue

This document covers all the new features up to Release 6.3.

New features

This section summarizes the new features up to Release 6.3.

Single-Step Conference

The Single-Step Conference allows applications to add a device into an existing call for the purpose of playing announcements or facilitating application-initiated transfers and conferences. This is accomplished with a single ASAI request, without the need for placing anyone on hold or initiating a new call.

Universal Call ID

Universal Call ID (UCID) is a unique tag assigned to each DEFINITY call. It is used by an application to track a call for its life, from origination to disconnect, regardless of where the call may end up and how it gets there (transfer, conference, routing, through a variety of network and the DEFINITY Enterprise Communications Servers, and voice responses, etc.).

The UCID is reported to all ASAI links on the DEFINITY ECS, if so administered. The event reports that contain the UCID are: Call Initiated, Call Offered, Alerting, Connected, Call Transferred, and Call Conferenced. Acknowledgments to Third Party Make Call, Third Party Auto-dial, Third Party Merge, and Third Party Take Control also contain UCID. The UCID is also passed in a Route Request Capability and UCID Query is also available in this release.

Release 5 Major Enhancements

- **Advice of Charge**

  This feature provides an ASAI application with the capability to receive information regarding the cost of an active outgoing call. It applies only in those countries where the network is able to send the cost of the current call to the DEFINITY ECS using the ISDN Advice of Charge feature.

- **Reason Codes**

  ASAI will allow adjuncts to enter a reason code when an agent’s work mode changes to AUX work or when an agent logs out. In addition, the adjunct can also query for an agent’s reason code status. This feature must be optioned and it is mandatory that the AUX Work Reason Codes and the Agent Logout Reason Codes be set to “forced” or “requested.”

- **ASAI Selective Listening**

  This feature allows an ASAI adjunct application to disconnect or reconnect selected listening paths. An application can use this feature to prevent one or more parties on a call from hearing communications among other parties on that call.

- **II-Digits**

  II-Digits provide information about the originating line. For example, these digits will indicate if the call is originating from a prison, a cellular system, a coin machine, or special operator, etc. II-Digits are passed to the DEFINITY ECS by the network on Integrated System Digital Network Primary Rate Interface (ISDN-PRI) trunks and are then passed to the adjunct over an ASAI link. An ASAI application can use the information provided by II-Digits to properly route or provide special treatment for the incoming call. This feature can also be used to prevent fraud.

  II-Digits will be populated in the Call Offered, Alerting, Connected Event Reports and in Route Request.

- **27-Character Display**

  The ASAI-Accessed Integrated Directory query has been modified to return up to 27 characters for names when extensions are retrieved from the Integrated Directory of ECS. Link version 3 (G3V5), must be negotiated between the ECS and the adjunct for this enhancement to work (otherwise, only 15 characters will be passed). International Standards Organization (ISO) certified optrex characters are also included in the ASAI-Accessed Integrated Directory.

- **OPTREX Characters**

  Some newer phones support escape sequences to display certain international characters. CallVisor PC now provides this data in new fields in “wide character” format so that it can be processed by the standard functions that accompany the UNIX operating system.

Intended Audience

This document is intended for system administrators, programmers and testers. It provides step-by-step procedures for installation and administration of CallVisor PC. This document also describes the interaction between the UNIX ASAI adjunct and the DEFINITY ECS. It is intended to assist applications programmers in developing applications for the adjunct. It includes complete information on all the ASAI library functions, capabilities and capability primitives. Integration Test Tool (ITT) is designed to help in testing the library functions. A number of issues and problems that may be encountered during various phases, from installation to regular administration, are also addressed.

It is assumed that the readers are familiar with the UNIX operating system (including advanced topics such as device polling through signals and unwaited I/O) and have at least a basic knowledge of the Integrated Services Digital Network (ISDN). In addition, readers should be thoroughly familiar with the DEFINITY ECS ASAI features and their restrictions.

Organization of This Document

This document is organized as follows:

Chapter 1 provides detailed installation steps.

Chapter 2 provides the manual pages for commands, daemons, and files used in administration.

Chapter 3 is an overview of the Adjunct-Switch Application Interface. It presents the terms and concepts specific to ASAI and includes an overview of the ASAI library: the functions, the application service elements (ASEs) and the capability primitives.

Chapter 4 describes the ASAI library functions and their use and contains a number of coding examples.

Chapter 5 has all the ASAI application service elements, providing details about the capabilities that comprise each ASE.

- Event Notification and Event Reports
- Third Party Call Control Capabilities
- Set Value
- Value Query
- Request Feature
- Adjunct Routing
- Maintenance
- Abort Capabilities

Chapter 6 introduces a new feature — CV/LAN.

Chapter 7 has a detailed list of error returns from the library functions

Chapter 8 is an introduction to ASAI capability primitives.

Chapter 9 contains manual pages for ASAI library functions, ASAI capabilities and primitive types in detail.

Chapter 10 is an overview of Integration Test Tool (ITT) and its functions.

Chapter 11 provides solutions to a number of issues and problems encountered during various phases of installation, administration, application programming and testing.

Appendix A contains ISO certified OPTREX characters.

A **Glossary** and **Index** are also provided for easy access to terms and definitions.

Related Documents

For specific information concerning the DEFINITY ECS Release 6 ASAI, the following documents are available from Lucent Technologies Publications Center (1 800 457-1235) :

- *DEFINITY Enterprise Communications Server Release 6 CallVisor ASAI Technical Reference*, 555 230-220
- *DEFINITY Enterprise Communications Server Release 6 CallVisor ASAI Protocol Reference*, 555 230-221
- *DEFINITY Enterprise Communications Server Release 6 Administration and Feature Description*, 555-230-522
- *DEFINITY Enterprise Communications Server Release 6 System Description Pocket Reference*, 555-230-211
- *DEFINITY ECS CallVisor ASAI DEFINITY LAN Gateway over MAPD Installation, Administration, and Maintenance, 555-230-114*
- *DEFINITY ECS CallVisor ASAI PC LAN over MAPD Installation, Administration, and Maintenance, 555-230-113*
- *CallVisor ASAI CD Document Set*, 585-246-801

This CD ROM contains six CallVisor ASAI release 6 documents. It includes *CallVisor ASAI Technical reference, CallVisor ASAI Protocol Reference, CallVisor ASAI Overview, CallVisor ASAI PC*, *CallVisor ASAI DEFINITY LAN Gateway over MAPD Installation, Administration, and Maintenance*, and *CallVisor ASAI PC LAN over MAPD Installation, Administration, and Maintenance*

Technical Service Center

The CallVisor PC helpline supports CallVisor PC customers who need technical assistance. Technical support is provided for installation, administration and functionality failures (“bugs”). The number of hours of technical support varies with the type of license purchased. The CallVisor PC helpline number is 1 732 957-5725.

The Technical Service Center (TSC) offers a consulting service to help application developers map desired functionality to ASAI capabilities. The TSC can be reached at 1 800 344-9670.

Installation

1

This chapter provides all the information on the DEFINITY ECS administration, system configuration, operation, and limits which is required to install CallVisor PC. This chapter also covers the procedure to install CV/LAN on Windows NT and UNIX, and run the sample application.

Platform Consideration

ASAI supports the following operating system platforms (CallVisor PC server) :

1. SCO UnixWare 2.1.2 (including ptf32801)
2. Solaris 2.4 x86
3. Solaris 2.5 x86

Hardware Platforms

As a general guideline, if one of the above listed operating systems will run on the hardware platform, CallVisor PC will also run. You should first choose a software platform, (UnixWare or Solaris x86), and work from the Hardware Certification list provided with the operating system. If planning a CallVisor PC ISDN-BRI installation, choose a PC model from the list with a clock speed below 90 MHz. A multi-CPU PC is not supported for either a PC-ISDN or LAN Gateway configurations.

The PC ISDN card (IPCI card) has been known to have problems in some PC’s with clock speeds greater than 90 MHz. However, many software vendors have indicated that they are using CallVisor PC ISDN in systems with clock speeds up to 166 MHz. At these speeds, the link will eventually fail.

There are two distinct types of ISDN board failures:

1. Inability to download the ISDN board firmware (failure to “pump”). This is discovered at installation time and can be diagnosed by checking the `/usr/adm/isdn/isdn_log` file for errors during the pumping phase.
2. This is link traffic related. At link message rates, approximately 1/3 to 1/2 the bandwidth of the ISDN link (link bandwidth is 30 messages per second), the ISDN board will experience a hardware timeout, and the link will drop. The only recovery is to re-pump the board (`ipci_OFF` followed by `ipci_ON`). Again, the hardware timeout error can be checked in the `isdn_log` file.

Some PC manufacturers provide the ability to reduce the clock rate via straps on the PC's motherboard. Check your PC's manual, or your hardware vendor to see if this is possible.

The CallVisor PC LAN Gateway interface has no known problems related to PC clock speeds.

The following sequence of steps is necessary to install CallVisor PC ASAI package on a PC. The order of these steps eliminates excessive adjunct computer kernel rebuilds, reboots and powerdowns.

1. Disable COM 2 (only if IRQ3 is to be used)
2. Install the CallVisor PC ISDN software (IPCI) device driver
3. Install CallVisor PC LAN Gateway software
4. Install the CallVisor PC ASAI software
5. Install the PC/ISDN board (hardware)
6. Test the CallVisor PC ASAI link
7. Install the CallVisor PC ITT software
8. Install CV/LAN Server software
9. Install CV/LAN Client software

The next section of this chapter provides detailed instructions to install the CallVisor PC LAN GATEWAY package on an adjunct. The CallVisor PC ISDN package must be installed prior to the LAN GATEWAY package. CallVisor PC ISDN and CallVisor PC LAN GATEWAY may be run concurrently on the same adjunct PC. Instructions to install CV/LAN server software on an adjunct can be found in ‘‘Server Installation on UNIX’’ on page 1-15 of this chapter. CV/LAN server can also be run on the MAPD. For further information, *see DEFINITY ECS CallVisor ASAI PC LAN over MAPD Installation, Administration and Maintenance*. Instructions to install CV/LAN clients on UNIX and Windows NT platforms are described later in this chapter. The adjunct is limited to a total of eight links; any combination of up to four ISDN links and up to four LAN GATEWAY links, is allowed. The links must be installed contiguously; their order may not be intermixed. If ISDN links are to be installed, they must be installed as link numbers 1 through 4. LAN GATEWAY link numbering must start immediately after the last

ISDN link. For example, if three ISDN links and two LAN links are installed, the ISDN links will be links 1, 2, 3 and the LAN links will be links 4 and 5. If no ISDN links are installed, LAN links will be configured starting at link number 1. The installation scripts will perform this numbering automatically. It is not possible to change the number of links (ISDN or LAN) without completely uninstalling and reinstalling the CallVisor PC ISDN, CallVisor PC LAN GATEWAY, and CallVisor PC ASAI packages.

Throughout this document the PC/ISDN board is referred to as the ISDN PC interface (IPCI) board; the DEFINITY LAN Gateway Board (for the optional LAN GATEWAY package) as the MFB, and the Multi Application Platform for the DEFINITY ECS as the MAPD. The MAPD currently supports the DEFINITY LAN Gateway package and the CV/LAN package. The IPCI device driver is part of the CallVisor PC ISDN software that is pumped to the board.

If this is an upgrade installation or the packages are being uninstalled and/or reinstalled to change the number of links, the current configuration parameters should be written down or printed out for future reference. These parameters can be found in the files `/usr/adm/isdn/ipci_parms`, `/usr/adm/asai/asai_parms`, and `/usr/adm/isdn/lan_parms`. Enter the following commands:

cat /usr/adm/isdn/ipci_parms

and

cat /usr/adm/asai/asai_parms

and

cat /usr/adm/isdn/lan_parms

to see the parameters.

NOTE:
Before beginning, the installer must make sure that the Network Support Utilities package has been installed.

1. If no ISDN links are going to be installed, skip to Step 3.

   Decide on the interrupt level (IRQ) number to be used. The IRQ number is the same as the Interrupt Vector Number (IVN) referred to when installing the CallVisor PC ISDN software. Available IRQs are 2 or 3. The CallVisor PC ISDN software package will ask which package is to be used. IRQ 2 is preferred because asynchronous communications port COM2 uses IRQ 3. If IRQ 2 is unavailable, IRQ 3 has to be used. If IRQ 3 must be used, the installer must first find out how to disable COM2 for his or her particular computer before going to the next step.

2. Decide on the shared memory addresses (SCMA) that the IPCI boards will use. The default PC memory space address for one IPCI board is d0000. Use this default address as long as it does not conflict with any other hardware. See Task 1 in Chapter 3, “Configure The Platform Hardware” of *PC/ISDN Platform Installation and Reference* for details on setting the address selection switches to use the desired address; also see Table 3-3, *“Switch Settings for PC/ISDN Interface Card Base Addresses.”*

3. When no one else is using the computer, log in as root. Bring the computer to single-user mode and then run state **1** by entering the command:

   shutdown -iS -y -g120

   Press **ctrl-d** to proceed and enter **1** for the selected run state.

4. Copy `/unix` to `stand/unixold` as a safety precaution in case of disaster. Enter the command:

   cp /unix /stand/unixold

5. Enter the command:

   pkgadd -d diskette1

   Place the disk in the appropriate drive and follow the online instructions.

   The **Pkgadd** displays that the CallVisor PC ISDN package is available to process. Press RETURN to continue the installation.

6. After several minutes the installation procedure asks how many IPCI boards are installed on the machine. Enter the number of IPCI boards that are already installed or the number which will be installed on the machine. The default is 1. If your installation will have only CallVisor PC LAN GATEWAY links, enter 0. Entering 0 will cause the installation to skip to the link version administration step. Enter the correct number (0, 1, 2, 3) or press RETURN for the default**.**

7. The installation procedure asks to enter the IVN number. This is the IRQ/IVN number for the IPCI boards that you decided to use in Step 1. The default is 2. Enter the correct number (2, 3) or press RETURN for the default**.**

8. The installation procedure asks you to enter the 5-digit SCMA address value. This is the shared memory address selected in Step 2. Be sure to use lowercase letters. The defaults are available for four boards and online help displays the available values. Enter the correct numbers or press RETURN for the default.

9. The installation procedure asks you to enter the desired version. The ISDN protocol stack, specifically the QP module, supports link version selection. Version 1 (default) corresponds to G3V3, version 2 corresponds to G3V4, and version 3 corresponds to Release 6.

For Release 6[1], the default also is 3. See the ‘‘DEFINITY ECS Administration’’ on page 1-17 of this chapter for more information how to get the DEFINITY ECS Software Version. Enter the correct number or press RETURN for the default.

NOTE:
Most of the features (except for new Event Reports) will be provided to the applications regardless of whether link version 1, version 2 or version 3 is selected. The TSC may have to activate a number of new ASAI features. ASAI version control is used to allow applications to work even though certain protocol features may not be totally upward compatible. The approach used by the DEFINITY ECS is that new ASAI messages that were unsolicited by the adjunct are under version control, but those messages which were requested by the adjunct are not. For example, the G3V4 Event Reports (Login and Call Originated) would not be sent if the adjunct had negotiated a version 1 (G3V3) link, but the adjunct could request the new Send DTMF Signals feature on G3V4 even under link version 1. This way, the adjunct can still have access to the new functionality without being concerned about receiving unexpected ASAI messages.

In CallVisor, QP_HIGHERVER_OK and QP_LOWERVER_OK have been changed to QP_HIGHERVER and QP_LOWERVER, respectively. The former represents boolean values used to allow or disallow support for other link versions. They now represent a list of versions supported by the adjunct. When necessary, the CallVisor PC administration can limit the versions at which the protocol will run to meet their application’s needs. Three kernel tunables are used to negotiate the ASAI link version:

* QP_DESIRED_VER - the preferred version
* QP_HIGHERVER - the highest allowed version
* QP_LOWERVER - the lowest allowed version

The version will be negotiated to the first acceptable one in the list. First try qp_ver, then qp_high, then qp_high-1.... and finally qp_low. If the ECS or CV/PC does not support version control, it will be negotiated as though the list contained only the lowest version, that is, 1. If the version negotiation fails, there will be no error message and the link will not come up. This can be diagnosed by running isdn_trace and seeing restarts (08 02 00 00 46 79 ...). No restart acks (08 02 80 00 4E 79 ...) will be run.

10. The installation procedure asks to enter the highest allowed version. QP_HIGHERVER is provided to specify the highest version. Enter a version number or press RETURN for the default.
11. The installation procedure asks to enter the lowest allowed version. QP_LOWERVER is provided to specify the lowest version. Enter a version number or press RETURN for the default.

1. The default is 3 which supports Release 5 and 6.

NOTE:
These values are set on a per-adjunct (PC) basis, not on a per-link basis. If the adjunct is to be connected to both a G3V3 link and a G3V4 link it would be advisable to select **1** for QP_LOWERVER and **2** for QP_HIGHERVER**.** Also, **2** should be selected for QP_DESIRED_VER**.** Version 3 corresponds to the DEFINITY ECS Release 6. For Release 6[2], the default also is 3. So for Release 6, QP_HIGHERVER and QP_DESIRED_VER should be set to 3.

12. The installation procedure asks you to remove the diskette from the disk drive and displays shutdown instructions. The message `Installation of <cvisdn> was successful` is also displayed.

    If you are not installing the optional CallVisor PC LAN GATEWAY package, skip to Step 18.

NOTE:
Install CallVisor PC LAN Gateway package before the installation of CallVisor PC LAN. If CallVisor PC LAN Gateway package is not installed first, it will take longer to come up. If it is done hours later, the DLG may hang and will need reboot.

13. Verify that the hostname and IP address of the ASAI MAPD board(s) for each CallVisor LAN link destination are in the `/etc/hosts` file on the adjunct PC. If they are not, then add them now.

NOTE:
If your LAN installation uses a Domain Name Service such as NIS instead of `/etc/hosts` for host name resolution, *The Network Administrator's Guide* manual for your operating system should be consulted for information on how to add hostnames and IP addresses.

14. Place the (optional) CallVisor PC LAN GATEWAY product disk in the appropriate drive and enter the command **pkgadd -d diskette1** and follow the online instructions.
15. The installation script will ask you to enter the number of LAN links you wish to install, enter one number from 1 to 4, the default is 1.
16. For each LAN link entered in the previous step, the installation script will ask you to provide a valid host name for the LAN link destination. This will be the host name(s) of the MAPD(s) added to `/etc/hosts` in Step 13. The default is DEFINITY ECS.

NOTE:
The installation script will attempt to verify the hostname of each LAN destination entered in the previous step and print a warning message if it cannot. Installation will proceed regardless of the outcome.

2. The default is 3 which supports Release 5 and 6.

17. The installation script will verify system requirements and complete the installation. If error messages occur, follow the instructions provided by the messages. Otherwise, the message `Installation of <cvesai> was successful` is displayed.

18. Place the CallVisor PC ASAI product disk in the appropriate drive and follow the online instructions.

    Pkgadd displays that the CallVisor PC ASAI package is available to process. Press RETURN to continue the installation.

19. After several minutes the installation procedure displays the number of ASAI nodes that the package is configuring. If this information is not available, the installation procedure asks you to enter the number of ASAI nodes. This is the combined total number of IPCI boards that are already installed or that are going to be installed and the number of LAN links being installed on the machine. The default is 1. Enter the correct number (1 to 8) or press RETURN for the default.

20. After the installation procedure rebuilds the kernel, the installation program asks you to remove the diskette from the disk drive and shutdown instructions are displayed. The message `Installation of <cvasai> was successful` is also displayed.

    - If this is an upgrade, enter the command:

      shutdown -i6 -y -g0

    and return to Step 19 and proceed.

    - If hardware is to be installed, enter the command:

      shutdown -i0 -y -g0

    When the screen message `reboot the computer now` appears, shut off the power.

21. If no ISDN links were installed, proceed to Step 27.

    With the power shut off, install the IPCI board by completing the Steps 2 and 3 listed in Chapter 3 of *PC/ISDN Platform Installation and Reference manual.*

    Do not do Step 4 or any subsequent steps listed in *PC/ISDN Platform Installation and Reference*.

    In Step 3, you do not have to connect either a headset or a voice terminal (phone). You must use a terminating resistor to connect the LINE connector to the wall jack going to the DEFINITY ECS BRI line.

NOTE:
Most ISDN voice terminals require power to operate. This may involve additional equipment depending on the wire installation and equipment in your building. Consult your DEFINITY ECS administrator or telecommunications consultant for additional information.

Do not turn the power on yet.

If you can see the LED with the cover on the computer, you may put the cover on. Otherwise, leave the cover off until instructed to put it back on after all the software is installed and the LED has confirmed the basic sanity of the IPCI board. If in doubt, leave the cover off.

22. If IRQ 3 was chosen, disable the COM2 port. Most 386/486-class computers have COM1 and COM2 on the motherboard. If this is the case, use the CMOS configuration utility to disable COM2 according to your computer user manual.

    If you do not have COM2 on the motherboard, check to see if you have an expansion board that provides COM2. If you do, disable it according to the manufacturer's documentation.

23. Power up the computer.

    The UNIX system should come up. If it does, proceed to the next step.

    If it does not, reboot again. When the message `Booting the UNIX system` appears, press the space bar. When the system asks `which file to boot from`? Enter the command:

    /stand/unixold

    The system now boots using the kernel that was copied in the earlier step. Call the CallVisor PC helpline number at 1 732 957-5725 for assistance before proceeding.

24. At the login prompt, log in as root.

25. Observe the LED on the IPCI board. This LED will normally begin to flash within 90 seconds after you log in. If it does not flash, call the CallVisor PC helpline number at 1 732 957-5725.

    A flashing LED means the board hardware and software are operating correctly.

26. Be sure that the DEFINITY ECS administration has been completed and the BRI line is connected to the LINE connector on the IPCI board.

27. Enter the command:

    asai_test

    This tests the connection from the adjunct to the DEFINITY ECS and verifies that the administration is correct. All boards and connections (ISDN and LAN) are tested. If the test passes, a success report is displayed.

    The QP module logs link startup failures to both the system consoles and the crash buffer. If the QP module has failed, it is probably because of the DEFINITY ECS and adjunct version incompatibility. If QP messages are seen on the console, correct the parameters as described in the ''Configuring and Reconfiguring Software'' on page 1-19 of this chapter. Use crash (1m) to examine the crash buffer.

    For more link details, see ''asai_test(1)'' in Chapter 2, ''OA&M.''

If the test fails, check the version parameters and consult the DEFINITY ECS administrator. If the test fails for LAN type links, proceed to Step 30.

If the DEFINITY ECS is properly administered and the wiring checks out, call the CallVisor PC helpline number at 1 732 957-5725.

28. Shut down the computer. Enter the command:

    shutdown -i0 -y -g0

    Switch off power and install the cover.

29. Power up the computer.

30. At the login prompt, log in as root and give the IPCI board one minute to come up. Recheck connectivity to the DEFINITY ECS using the **asai_test** command.

    If the test fails, consult the DEFINITY ECS administrator. If the DEFINITY ECS is properly administered and the wiring checks out but the test still fails, call the CallVisor PC helpline number at 1 732 957-5725.

    If the connectivity fails for LAN type links, use the **link_status(1)** command to obtain the current status of LAN link(s).

    If the status is other than Talking, use the **ping** command to verify connectivity to the host; for example, `/usr/sbin/ping hostname` where hostname is the hostname of the MAPD administered in Step 13.

    If the **ping** command returns an affirmative response yet **asai_test** fails, check the `/usr/adm/isdn/lan_parms` file for typographical errors. If there are none, verify that the MAPD administration is correct by following the procedures in the *DEFINITY ECS CallVisor ASAI DEFINITY LAN Gateway over MAPD Installation, Administration, and Maintenance,* for your operating system.

    If the **ping** command returns a negative response, either the ASAI MAPD is not properly configured, not connected to the LAN or there may be other LAN problems (for example, LAN administration or routing). Consult the *DEFINITY ECS CallVisor ASAI DEFINITY LAN Gateway over MAPD Installation, Administration, and Maintenance* and the *Network Administrators' Manual* for these types of problems.

    If the DEFINITY ECS and ASAI MAPD are properly administered and the LAN checks out, call the CallVisor PC helpline number at 1 732 957-5725.

    If the connectivity test succeeds, you have demonstrated the compatibility of the computing platform and the CallVisor PC package.

    Congratulations, you have successfully installed the CallVisor PC package.

NOTE:
The kernel environment variable which is the number of cluster ids (NCLID) sets the amount of memory to be reserved. NCLID effectively limits the number of active ASAI associations that are allowed to run on each BRI board or LAN GATEWAY link. NCLID has a default value of 2048 if only

ISDN links are installed and a default value of 1024 if any LAN GATEWAY links are installed. This should be sufficient for the majority of applications using ASAI.

However, if the adjunct processor has a limited amount of memory or if a running ASAI application is controlling a large number of stations or calls, NCLID may need to be changed. In any case, NCLID should be set to conform to your system's specific operational needs. For more details, see ''Configuration'' on page 1-18 of this chapter.

You can install the CallVisor PC ITT package at any time. Enter the command:

pkgadd -d diskette1

Place the CallVisor PC ITT product disk in the appropriate drive and follow the online instructions.

Pkgadd displays that the CallVisor PC ITT package is available to process. Press RETURN to continue installation**.**

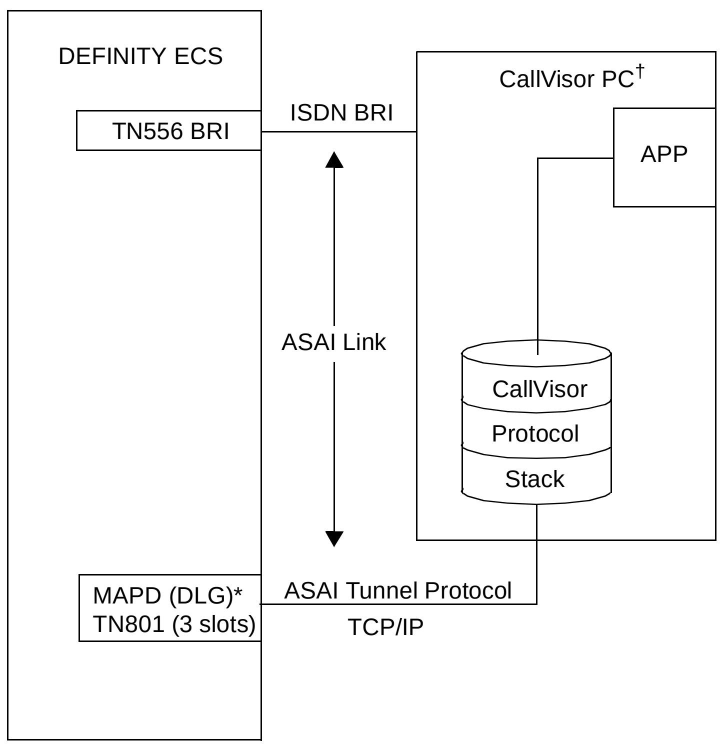
After the installation procedure is complete, remove the diskette from the disk drive. The message `Installation of <cvitt> was successful` is displayed.

CV/LAN Product Description

CV/LAN allows application software running on UnixWare, Solaris x86, SPARC Solaris™, or Windows NT 4.0 to access the DEFINITY ECS CallVisor ASAI features across a TCP/IP LAN in a client-server arrangement.

The server runs on the CallVisor PC or the Multi-Application Platform for DEFINITY (MAPD). The CallVisor PC and CV/LAN applications use the same library and differ slightly in only one API call. For further information see Chapter 6, ‘‘CV/LAN Programming.’’

The following Figure 1-1 shows how the applications run on the CallVisor PC server and connect to the DEFINITY ECS through the DEFINITY LAN Gateway ethernet interface, or through an ISDN-BRI interface.



* MAPD Board Running the DEFINITY LAN Gateway Application
† UnixWare or Solaris X86

Figure 1-1. CallVisor PC Adjunct-Server Configuration

The following Figure 1-2 shows how CallVisor PC is configured as a CV/LAN server. Applications can run on all the connected CV/LAN clients. Connectivity to the DEFINITY ECS is provided through the CallVisor PC server over either a DEFINITY LAN Gateway Interface, or an ISDN-BRI interface. The CV/LAN clients do not communicate directly with the ECS.

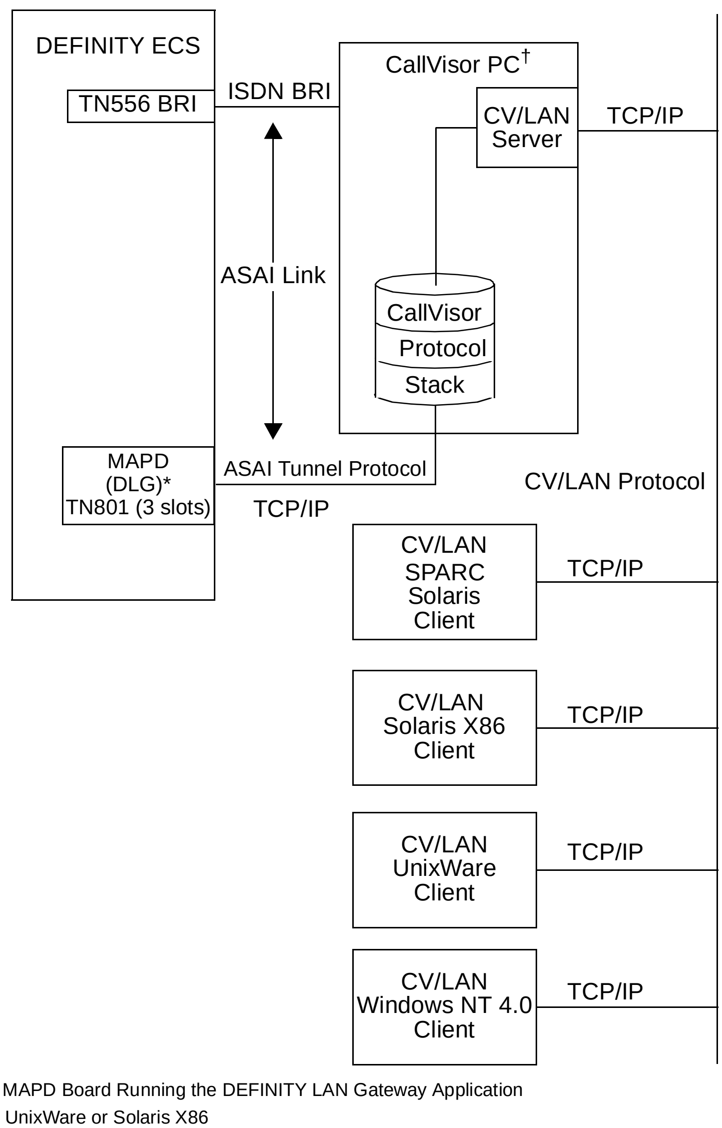


Figure 1-2. CallVisor PC as a CV/LAN server

The following Figure 1-3 shows that ASAI client application communication is done directly through TCP/IP to the MAPD running CV/LAN server software (PEC 1273-CVL). It does not involve any CallVisor PC server.

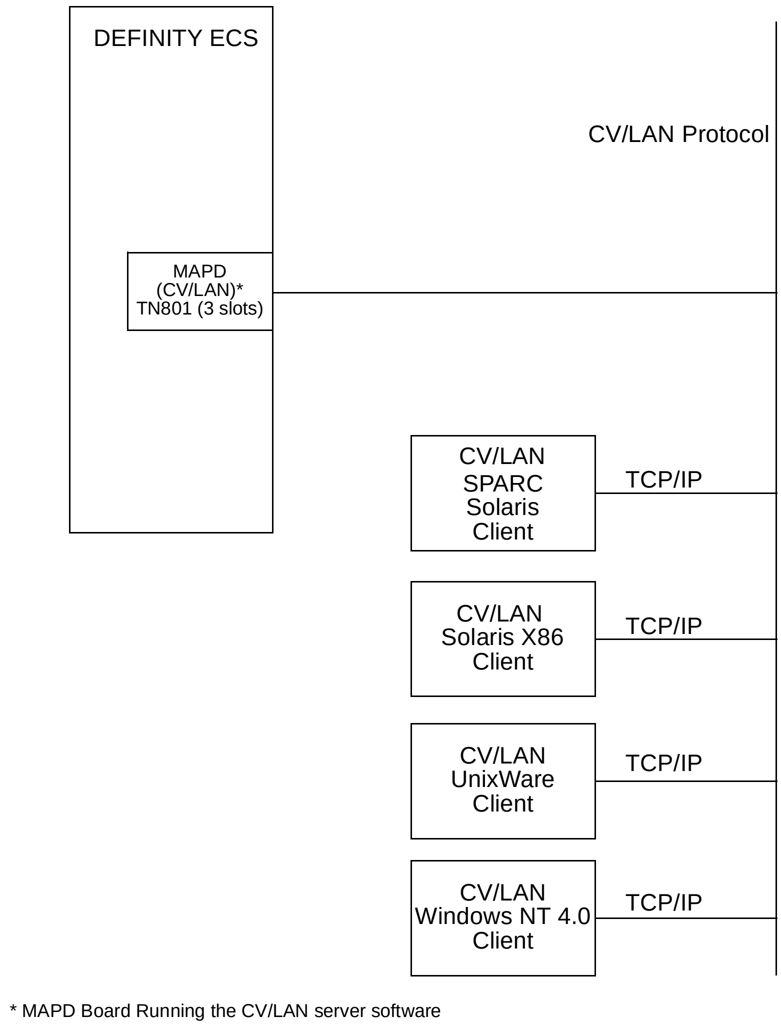


Figure 1-3. CallVisor PC Configuration — CV/LAN to MAPD

CV/LAN client is supported over the following platforms: UnixWare 2.1.2 and later; Solaris x86, SPARC Solaris versions 2.4, 2.5, 2.5.1, and Windows NT 4.0. The CV/LAN server is supported on Unixware 2.1.2 and later, as well as Solaris x86.

System Configuration

Each client communicates with the server through any Ethernet interface card supporting Unixware versions 2.1.2, and later; Solaris x86, SPARC Solaris versions 2.4, 2.5, 2.5.1, as well as Windows NT 4.0. Application software on the clients has access to the full range of ASAI features supported by the CallVisor PC ASAI interface.

The CV/LAN server runs on UnixWare 2.1.2 and later, and Solaris x86 operating systems; it requires the CallVisor PC ASAI library package. The interface card between the server and the DEFINITY ECS may consist of the PC/ISDN-BRI card or the DEFINITY ECS LAN Gateway card. The server communicates with clients over the LAN through any Ethernet interface card supporting Unixware or Solaris x86.

Software Configuration

The CallVisor PC ASAI library must also be installed on the client side. Client administration specifies a "well known address" to access a server. In addition to UNIX, the server must be running the CallVisor PC ASAI library package and CV/LAN server software.

System Operation and Limits

Client applications use the CV/PC ASAI library to establish communication with a server. This process is nearly identical to that used with the CallVisor PC ASAI library (server addresses must be specified by the client; this is not required in the PC ASAI library). Once a connection is established, the applications use library routines `asai_send()` and `asai_rcv()` to send and receive ASAI messages. The server routes the messages to and from the proper DEFINITY ECS. The format of the messages matches exactly those used with the CallVisor PC/ASAI library. Except for minor changes in the communication establishment, `asai_open` expects that the applications developed with the CallVisor PC/ASAI library will run unchanged over the LAN.

Certain configuration limits are inherited from the CallVisor PC ASAI library or the DEFINITY ECS ASAI limits. In particular, only one client can provide routing or maintenance (heartbeat) service, per link for an ECS. ASAI limits for the number of active monitors, etc., supported by a single ECS, are not changed by the CV/LAN implementation.

Server Installation on UNIX

For the CV/LAN, UNIX Platform refers to both Unixware and Solaris x86.

The CV/LAN server can only be installed after CVISDN and CVASAI have been installed.

1. To install the CV/LAN server, type the command:

   pkgadd -d diskette1

2. Place the Lucent Technologies CallVisor PC CV/LAN Server disk in the appropriate drive and follow the online instructions. **Pkgadd** displays that the package is available to process. Press RETURN to continue the installation.

After the installation procedure is complete, the installation program asks you to remove the diskette from the disk drive. Shutdown instructions and the message `Installation of <cvlansrv> was successful`, are also displayed.

Client Installation on UNIX

You can install the PC CV/LAN Client package at any time.

1. Enter the command:

   pkgadd -d diskette1

2. Place the PC CV/LAN Client disk in the appropriate drive and follow the online instructions.
3. **Pkgadd** displays that the PC CV/LAN Client package is available to process. Press RETURN to continue the installation.
4. After the installation procedure is complete, remove the diskette from the disk drive. The message `Installation of <cvlanxcl> was successful`, is displayed.

Client Installation on Windows NT

1. To install the CV/LAN Client on a Windows NT 4.0 system, insert the Client diskette in drive A.
2. Click on the `RUN` File Menu Item under Program Manager or File Manager, type **Setup** in the dialog box, and press **OK**.
3. Follow **Setup** instructions.

NOTE:
After the **Setup** is complete, CV/LAN.DLL, the dynamic link library and CV/LAN.LIB, the import library, will be installed under a subdirectory called `Program`.

Running the Sample Application

To run the application on UNIX, the command is **itt <script-name>.**

For Windows NT 4.0 system, the command **itt <script-name>** can be run from a DOS window.

DEFINITY ECS Administration

The DEFINITY ECS BRI line must be administered to establish ASAI connectivity between the DEFINITY ECS and the adjunct computer. Use the **add station** or **change station** command to administer the BRI line. Use the following table as a guide.

Table 1-1. Field Name and Requirements

Field Name	Required/Optional	Contents
Extension:	R	System dial plan sequence
Type:	R	ASAI
Port:	R	Port connecting to ASAI line
Name:	O	Your name or handle
XID:	R	y
Fixed TEI:	R	y
TEI:	R	3
MIM Support:	R	n
CRV Length:	R	2
Event Minimization	R	n

The Type, XID, Fixed TEI, TEI, MIM Support and CRV Length fields all must have the indicated required contents in order to match the built in administration of the IPCI board and CallVisor PC ASAI software.

The ISDN protocol stack, specifically the QP module, supports version selection. Version 1 (default) corresponds to G3V3, Version 2 corresponds to G3V4 and Version 3 corresponds to the DEFINITY ECS Release 6. For Release 6[3], the default also is 3. To find out which software version is on the DEFINITY ECS, use the **list conf soft** command. The Software Version is displayed on the Memory Resident: line.

3. The default is 3 which supports Release 5 and 6.

Configuration

This section assumes that you have read the installation steps in this document.

NOTE:
The Network Support Utilities (NSU) package must be installed before you can install the CallVisor software packages.

The NSU package contains the streams utilities, header files, libraries and other files that are needed to add streams modules to the UNIX system kernel.

Upgrading Software

The upgrade procedure consists of removing the existing packages and installing the new packages as described in “DEFINITY ECS Administration’’ on page 1-17 of this chapter. Upgrade the system when no one else is using the computer.

You must supply the configuration information as part of the upgrade procedure.

NOTE:
Before beginning the upgrade, copy or print the current configuration parameters for future reference.These parameters can be found in the files `/usr/adm/asai/asai_parms` and `/usr/adm/isdn/ipci_parms`.

1. Log in as root. Bring the computer to single-user mode and then run state **1** by entering the command:

   shutdown -iS -y -g120

2. Press **Ctrl-d** to proceed and enter **1** for the selected run state.
3. Remove the CallVisor PC software by entering the commands:

   pkgrm cvasai and **pkgrm cvisdn**

4. Now install the new version. Configure the software as described in the “Configuring and Reconfiguring Software’’ on page 1-19 of this chapter.

Removing Software

To remove software, enter the command:

pkgrm cvasai

and

pkgrm cvisdn

and

pkgrm cvesai

pkgrm command removes all package-created and installed files, along with directories created when the package was installed.

When you execute **pkgrm cvasai**, all files in the `/usr/adm/asai` directory, along with all other files belonging to the ASAI package, are removed.

Also, for isdn package, when you execute **pkgrm cvasai**, all files in the `/usr/adm/isdn` directory, along with all other files belonging to the ASAI package, are removed.

All ASAI header files installed in `/usr/include/asai` and `ISDN/IPCI` headers for the ISDN package in the `/usr/include/isdn` directory are removed. The directories are removed only if they are empty.

NOTE:
If you have local header files or other files in these directories, they will not be removed and neither will the directory.

The **pkgrm** procedure will build a new UNIX kernel. When this is completed, a message will be displayed requesting to reboot the system. After both packages have been removed, execute the command

shutdown -i6 -y -g0

to reboot the system.

Configuring and Reconfiguring Software

The initial part of the installation and upgrade procedure consists of configuring the software. You must enter the information when prompted by the software. See the ‘‘Configuration’’ on page 1-18 of this chapter for more details. For the ISDN package, you must enter the number of IPCI boards to be installed or those already installed, the IRQ number, the PC memory addresses as well as the matching version parameters. Refer to the table below for default IPCI values:

Table 1-2. Default IPCI Configuration Values

ISDN board #	IRQ/IVN	PC memory address
1	2	d0000
2	2	d4000
3	2	d8000
4	2	dc000

Note that regardless of the number of ISDN boards installed in the computer, the same IRQ/IVN number is used. Matching Versions Requirement for the ISDN protocol stack, specifically for the QP module, must be done correctly or the ASAI link will not come up. Three new tunable parameters are provided to allow selecting the desired version and allowable alternatives. QP_DESIRED is set to the desired version. Version 1 corresponds to G3V3, Version 2 corresponds to G3V4 and Version 3 corresponds to the DEFINITY ECS Release 6. For Release 6[4], the default also is 3. QP_HIGHERVER and QP_LOWERVER are provided to allow higher version and/or lower version operation.

The tunable parameters are located in `/etc/conf/cf.d/stune.` The defaults are:

QP_DESIRED_VER is set to 3 (Release 6)
QP_HIGHERVER is set to 3 to allow higher (later) versions
QP_LOWERVER is set to 1 to allow lower (earlier) versions

If you have to reconfigure your adjunct computer system (to change the version, for example), do so when no one else is using the adjunct.

To change the values, edit the `/etc/conf/cf.d/stune` file. Follow the standard tuning procedures and rebuild the kernel, then reboot the system. Execute the command

shutdown -i6 -y -g0

to reboot the system.

If the version parameters are incompatible with the ECS version, the BRI link will not start up. The QP module will log this error to the system console and the crash buffer.

For the ASAI package, you must enter the number of ASAI nodes/IPCI boards installed: 1, 2, 3, or 4 (if that information is not available at the time of installation). If the ISDN package was installed first, this information is available to the ASAI package.

If only one IPCI board was configured for the ISDN package, the following message is displayed:

```
Configuring <cvasai> for 1 ASAI Nodes.
```

Reconfiguring the packages requires you to reinstall the ISDN and ASAI software.

If you have to reconfigure your adjunct computer system (to add a board, for example), do so when no one else is using the adjunct.

4. The default is 3 which supports Release 5 and 6.

NOTE:
Make sure that you save a copy of the current configuration parameters for future reference.

Memory Considerations

This section introduces a number of concepts that assume practical experience with administration of the ASAI applications package. The material presented here is discussed in detail in Chapter 1, “ASAI and Capability Groups” of *DEFINITY Enterprise Communications Server Release 6 CallVisor ASAI Technical Reference.*

ASAI services are broken down into functional sets called capability groups. Capability groups enable the adjunct to communicate and control the DEFINITY ECS. Central to the idea of capability groups and ASAI in general is the concept of an association.

An association is a channel of communication between the adjunct and the DEFINITY ECS that is used for messaging purposes. An association begins with initiating capabilities, controlling capabilities manipulate messages during an association and terminating capabilities end an association.

An adjunct or more specifically, each BRI board installed on an adjunct, can manage many active associations at one time and each association must be tracked by the CallVisor ASAI software drivers. Tracking consumes adjunct memory that must be reserved for the CallVisor drivers (148 bytes are required to track a single association).

The amount of memory to be reserved is specified by the NCLID (number of cluster ids) environment variable. NCLID effectively limits the number of active ASAI associations that are allowed to run on each BRI board. It is important to note that the NCLID specifies the maximum number of associations that may run on each BRI board, not the maximum number for all boards combined.

The NCLID default is 2048. This should be sufficient for the majority of applications using ASAI. However, if the adjunct processor has a limited amount of memory, or if the ASAI application is controlling a large number of stations or calls, the NCLID will need to be adjusted. In any case, the NCLID value should always be set to conform to your system’s specific operational needs.

The default of 2048 specifies that each BRI board installed be limited to 2048 active associations and that approximately 296K of adjunct memory be reserved on behalf of each board (2048 x 148 bytes per association = 296K). If four boards are installed, a total of approximately 4 x 296K will be reserved for tracking their combined associations.

Note that with four boards installed, if the adjunct has four megabytes of memory or fewer, system performance will suffer greatly. In this case, it is recommended that one or more boards be removed, or that the NCLID be lowered. Lowering the

NCLID reduces both the number of active associations and the amount of adjunct memory reserved for each board.

For instance, setting the NCLID to 1024 reduces the number of active associations per board to 1024 and cuts the amount of reserved memory per board to 148K (1024 x 148). Alternately, setting the NCLID to 512 reduces the number of active associations per board to 512 and cuts the amount of reserved memory per board to 74K (512 x 148). These settings are not likely to impact the performance of most applications.

Certain ASAI applications may require the NCLID to be increased above the default of 2048. If the ASAI application is using one ASAI link to control a large number of calls on a R5r, the NCLID may need to be increased. There is, however, a practical limit to which the NCLID value may be raised. Appendix B “ASAI and Release 6 Requirements” of *DEFINITY ECS Release 6 CallVisor ASAI Technical Reference* discusses the capacity requirements and constraints for the DEFINITY ECS Release 6 in detail.

For example, Table B-1 of that appendix lists all the R5r systemwide ASAI limits for the maximum number of domain-control station associations as 6000. However, setting the NCLID to a value between the default (2048) and the maximum (9600), (for example 5000), will not always guarantee that 5000 active ASAI associations can be supported.

Despite the fact that the *theoretical* maximum number of domain-control station associations is 6000, the *actual* maximum number of associations is limited by the availability of a variety of other system resources.

For this reason, you must consider several factors when setting the NCLID in the hope of maximizing performance. The amount of memory available on the adjunct processor, as well as the number of BRI boards installed, greatly affect system capabilities.

It must also be noted that the capacity requirements and constraints on the DEFINITY ECS Release 6, apply equally to all ASAI links. If an application uses more than one ASAI link or if other ASAI applications are connected to the same DEFINITY ECS, the maximum number of associations per link will be lower than normal. This will also affect the *actual* maximum number of associations that can be realized.

Set NCLID using the **idtune(1M)** command. Follow the procedures outlined in *UNIX System V/386 Release 4 System Administrator’s Guide* for rebuilding the kernel. Any NCLID between 64 and 9600 will work, provided that sufficient adjunct memory is available. Setting the NCLID to a value outside this range is not supported.

Starting Up the System

System start-up for CallVisor PC ASAI is automatic. When you boot the adjunct, the `asai_admin` daemon is started. `asai_admin` completes the protocol stack and begins the process of message writing to the `asai_log`. If there is an existing log, it is moved to `old_asai_log`.

The `asai_log` contains messages about the start-up process. The sample below shows typical log entries for a successful start-up.

```
910318165042:asai_admin: ASAI administration daemon has started.
910318165042:asai_admin: Push of FEL onto /dev/isdn/ipci/signal01 succeeded.
910318165042:asai_admin: Link of /dev/asai/asai file with /dev/isdn/ipci/signal.
910318165042:asai_admin: l01 succeeded.
910318165042:asai_admin: ASAI stack setup was successful.
```

Matching Versions Requirement for the ISDN protocol stack, specifically the QP module, must be done correctly or the ASAI link will not come up. Three tunable parameters are provided to allow selecting the desired version and allowable alternatives. QP_DESIRED_VER is set to the desired version. Version 1 corresponds to G3V3, Version 2 corresponds to G3V4 and Version 3 corresponds to the DEFINITY ECS Release 6[5].

5. The default is 3 which supports Release 5 and 6.

OA&M

2

ASAI operation, administration, and maintenance (OA&M) consists of a few tasks that have to be performed at regular intervals and/or under certain conditions.

The ASAI Log File

The ASAI log file, `asai_log`, is stored in `/usr/adm/asai` and can be examined using **vi** or **cat**.

Each time the UNIX system is started, the existing log file is moved to `old_asai_log` and a new log file is created. This `old_asai_log` file is located in the same directory as `asai_log`. In addition, if `asai_log` becomes "full" (close to `ulimit` in size), it is moved to `old_asai_log`. Note that this overwrites an existing `old_asai_log`.

You can split or copy a log file; however, only `root` has write permission in the `/usr/adm/asai` directory. Therefore, you will have to use a directory in which you have write permission to execute the **split** or **cp** command. (The ASAI Administration menu provides access to UNIX system commands, so you can do this by escaping from the menu, if you prefer.)

The log file contains entries consisting of a date and time stamp followed by a colon, the name of the ASAI process that wrote the message, another colon and the message. The message tells you whether or not the process executed successfully.

```
910318165042:  asai_admin:  ASAI administration daemon has started.
910318165042:  asai_admin:  Push of FEL onto /dev/isdn/ipci/signal01
succeeded.
910318165042:  asai_admin:  Link of /dev/asai/asai with
/dev/isdn/ipci/signal.
910318165042:  asai_admin:  101 succeeded.
910318165042:  asai_admin:  ASAI stack setup was successful.
```

Figure 2-1. An ASAI Log File

In the sample shown above, the first 12 characters represent a date and time stamp. The date is in the form *yymmdd*; the time is in the form *hhmmss*, based on a 24-hour clock. Thus 11:30 p.m. appears as 2330.

Following the date and time stamp, comes the name of an ASAI process, preceded by a colon (`asai_admin` in the sample). Another colon precedes the message. Messages longer than 53 characters are carried over to the next line as shown in lines 2 and 3 in Figure 2-1.

OA&M Manual Pages

This section contains manual pages for the commands and files that comprise ASAI Operations, Administration and Maintenance (OA&M). For ease of reference, commands, and files appear in alphabetical (ASCII) order.

In this section (1) corresponds to commands, (4) refers to the files, and (7) corresponds to the devices and pseudo-devices.

Command name prefixes and suffixes imply the following:

`asai_`	Commands related to the kernel ASAI protocol stack or the ASAI library
`ipci_`	Commands related to the ipci device drivers
`isdn_`	Commands related to the BRI link between the BRI board and the ECS
`.bin`	Commands related to software that is downloaded (“pumped”) to the BRI board
`esai_`	Commands related to the CallVisor LAN GATEWAY interface
`link`	Commands related to the CallVisor LAN GATEWAY interface

```
admin(7)           ipci_off(1)
asai(4)            ipci_on(1)
asai(7)            ipci_stat(1)
asai.Date(4)       ipci_test(1)
asai.Name(4)       ipci_ver(1))
asai_admin(1)      isdn_alarm(1)
asai_cause(1)      isdn.Date(4)
asai_hb(1)         isdn_l1_r(1)
asai_log((4)       isdn_l2_r(1)
asai_test(1)       isdn.Name(4)
asai_trace(1)      isdn_trace(1)
asai_ver(1)        lan_status(1)
boot.bin(4)        log_msgs(4)
cmd(7)             link_alarm(1)
command(7)         link_offline(1)
esai_alarm(1)      link_restart(1)
esai_trace(1)      link_status(1)
ipci_admin(1)      pcisdn.bin(4)
ipci_init(1)       signal(7)
```

admin(7)

Name

`admin` — IPCI and LAN GATEWAY Streams Device Driver Communication File

Description

The admin file allows communication between the `ipci_admin(1)` program, the IPCI streams device driver and the LAN GATEWAY streams module. The IPCI streams device driver communicates with the IPCI board and with software on the board.

This file is meant to be used only by the `ipci_admin(1)` program. Information flows only one way from the IPCI device driver to the `ipci_admin(1)` program.

This file will have the same major device number as the signal(7) and command(7) files. Its minor device number is 0.

Files

```
/dev/isdn/ipci/admin
/dev/esai/admin
```

See Also

```
ipci_admin(1)
command(7)
cmd(7)
signal(7)
```

asai(4)

Name

`asai` — Start ASAI administration

Synopsis

`asai`

Description

`asai` is a shell script that starts the ASAI administration daemon process, `asai_admin(1)`, located in `/etc/idrc.d`.

Operation

The asai program is executed once per machine boot in the multiuser *init* level 2. The name of the log file (`asai_log`) is changed to `old_asai_log`. The administration daemon `asai_admin(1)` is then started. When this is done, the standard error device is defined as `/dev/console` for `asai_admin(1)`.

Files

`/usr/adm/asai/asai_log`

See Also

`asai(7)`
`asai_admin(1)`
`asai_log(4)`

asai(7)

Name

`asai` — ASAI communication file

Description

This file allows communication with the ECS via the ASAI library functions. It is meant to be opened by the ASAI library function `asai_open()`.

This file is created when the software is installed.

Files

`/dev/asai/asai`

See Also

`asai_admin(1)`

asai.Date(4)

Name

`asai.Date` — Date file for UNIX ASAI

Description

This file contains the date and time the software for ASAI product was manufactured.

Files

`/usr/adm/asai/asai.Date`

asai.Name(4)

Name

`asai.Name` — Name file for UNIX ASAI

Description

This file contains the name of the ASAI software package. The content of the file is:

```
Lucent Technologies CallVisor ASAI
```

Files

`/usr/adm/asai/asai.Name`

asai_admin(1)

Name

asai_admin — Administer ASAI file

Synopsis

asai_admin

Description

asai_admin sets up the protocol stack for the ASAI device file and sends necessary parameters to the ASAI provider.

Operation

The asai_admin program is executed once per machine boot. It is started by asai(4) and becomes a daemon process.

This program has two tasks.

a. To perform the necessary streams operations with UNIX kernel module(s) and the signal streams to complete the ASAI streams protocol stack.

b. To condition each protocol stack with the board node ID value and the ECS protocol value.

The protocol stack can be used after asai_admin has built the protocol stack. When asai_admin has built the stack, the /dev/asai/asai file is available for use.

The ASAI stream file is always kept open by asai_admin.

Messages

Messages (status and error information) are written to the asai_log file. Some of these messages may come from the ASAI provider streams module.

Exit Codes

This program, if it exits, exits with a code of 1. Since this program is meant to be run as a daemon process, the exit code is meaningless.

Files

```
/dev/asai/asai
/usr/adm/asai/asai_log
```

See Also

```
asai(4)
asai(7)
asai_admin(1)
asai_log(4)
```

asai_cause(1)

Name

asai_cause — Logs cause values to stdout

Synopsis

asai_cause [-h] [-v] [-t *logtime] [-?] process ids*

Description

When asai_cause is activated for a process (an ASAI application that is running) and the application receives an ASAI message which contains a cause value from the ECS, the library function asai_rcv() sends an Interprocess Communication (IPC) message containing the necessary information to asai_cause. asai_cause reads the IPC queue, formats the cause information and writes it to stdout.

asai_cause is activated by including at least one valid process id. Process ids are not checked for validity. asai_cause may be activated remotely or from any login in the system. It is not necessary to be logged in as root to access asai_cause.

asai_cause prints cause values to stdout. The following information is logged:

- Date and time
- Process id
- ASAI capability name or the hexadecimal operation value (indicates message type)
- sao_id (formerly known as cluster_id) identifies the particular association for which the message is intended
- Contents of the cause information element (IE) that includes the coding standard and the cause value. The cause value is actually the combination of two fields: a class and a value within the class. The DEFINITY ECS value and the ASAI library interpretation of the cause value is also included.

The set of PIDs (process identification) being monitored by asai_cause can be changed by reinvoking the command with a new set of PIDs. However, other parameters cannot be changed in this way. The second invocation of asai_cause simply changes the PIDs and then exits; it does not start a new logging process.

The set of input options includes:

`-h`

Causes SIGHUP to be ignored. Default is to exit `asai_cause` on SIGHUP (hangup).

`-v`

Causes the one-byte hexadecimal operation value from the Facility Information Element (FIE) to reference the ASAI capability rather than include the capability name itself.

`-t log time`

Specifies a decimal digit string indicating the maximum number of seconds that `asai_cause` will run. Default is `-t 0xffffffff`, which is (2**32)-1 seconds.

`-?`

Causes a command usage line to be returned.

`process ids`

Specifies a list of process ids (decimal digit strings) separated by blanks for which cause value logging is to be performed. An empty list turns off cause value logging, but leaves the `asai_cause` process running. An easier way of terminating cause value logging is to kill the `asai_cause` process. A maximum of 160 process ids may be included.

Example

Following is an example of the format used for logging cause values to `stdout` when the `-v` option is not included.

DATE	TIME	PID	CAPABILITY	SAO ID	DEFINITY ECS	ASAI
08/27/92	11:35:33	123456	C_3PMC_CONF	4444	CS3/38	C_NETWORK_OUT_OF_O
08/27/92	11:38:12	123456	C_EN_CAN_CONF	1	CS0/100	C_INVLDIE
08/28/92	07:49:52	55555	C_3PAD_CONF	fffffffd	CS0/17	C_USER_BUSY
08/28/92	09:06:02	55555	C_EN_REP	ffffffff	CS0/16	C_NORMAL

Following is an example of the format used for logging cause values to `stdout` when the `-v` option is specified.

DATE	TIME	PID	CAPABILITY	SAO ID	DEFINITY ECS	ASAI
08/27/92	11:35:33	123456	0x83	4444	CS3/38	C_NETWORK_OUT_OF_O
08/27/92	11:38:12	123456	0x94	1	CS0/100	C_INVLDIE
08/28/92	07:49:52	55555	0xc6	fffffffd	CS0/17	C_USER_BUSY
08/28/92	09:06:02	55555	0x95	ffffffff	CS0/16	C_NORMAL

See Also

`asai_trace(1)`
`asai_trace(1)`

Shared Resources

- Message Queue
- Shared Memory

Exit Codes

If `asai_cause` fails, one of the following exit codes is returned:

1 = `ftok()` failed to determine key

2 = Error in getting shared memory

3 = Error in getting message queue

4 = Error in getting a semaphore

5 = Attempt to ignore SIGHUP failed

6 = `asai_cause` command-line usage problem

7 = Error in attaching to shared memory

8 = Error in setting the value of the semaphore

9 = Error in receiving a message

10 = Error on attempt to log ASAI cause values

asai_hb(1)

Name

asai_hb — Acknowledges heartbeats for a Node

Synopsis

```
asai_hb [-v] [-?] [-m hostname/IPaddress] [1-8] -n
```

Description

When asai_hb is activated for an ASAI Node it registers itself as a maintenance server for that Node and calls the library function asai_rcv(). If the message received is an ASAI heartbeat, an ASAI heartbeat acknowledgment is sent.

If an ASAI heartbeat is not received from the ECS within two minutes, asai_hb sends a heartbeat request to the ECS. If the ECS does not confirm the request, and the -n option has not been specified, the specified Node is brought down using
ipci_off.

-v

Causes asai_hb responses to be printed on the user's terminal (stdout).

-?

Causes command-line usage information to be printed on the user's terminal.

1-8

1-8 is the Node number that corresponds to the board for which heartbeats will be acknowledged. Node number information can be found in the files /usr/adm/asai/asai_parms and /usr/adm/isdn/lan_parms.

The contents of asai_parms for one installed IPCI board are:

```
Node ID for ASAI Node 1 = signal01
```

-m

asai_hb can now be used as a CV/LAN client tool as well as a CV/PC tool. This option has been added to support the use of this tool from a CV/LAN client. The user must specify the IP address or hostname of the CV/LAN server host.

-n

Disables the shutdown ipci_off of the link. This option should be used for all LAN type links. It must be the last option specified on the command line.

NOTE:
-m and -n are mutually exclusive.

Enter the command **asai_hb 1** and the process checks for and acknowledges heartbeats immediately for Node number 1.

Files

```
/usr/bin/asai_hb
/dev/asai/asai
/usr/bin/ipci_off
/usr/adm/asai/asai_parms
/usr/adm/isdn/lan_parms
```

See Also

```
asai_test(1)
asai_trace(1)
```

asai_log(4)

Name

asai_log — Log file for UNIX ASAI

Description

This file contains all status and error messages. It resides in the /usr/adm/asai directory.

The format is:

time stamp: program name: ASCII message newline character

time stamp consists of 12 characters in the format of: *yymmddhhmmss*. Program name is a 12-character field containing the name of the program (left-justified) that contributed the message to the file. The program name is padded on the right with space characters as needed to fill the field. ASCII message consists of 0 through 53 ASCII characters. Every line is ended by a newline character as shown.

This file should not grow quickly. The programmer can monitor the size of this file, if desired. If this file does get close to the ulimit value, it is renamed old_asai_log and subsequent messages are written to a new asai_log.

When the system is booted, the old version of this file is renamed old_asai_log. A new asai_log file of 0 length is then created. Subsequent messages are written to this new file. Any previous copy of old_asai_log is removed by this operation.

Files

/usr/adm/asai/asai_log

See Also

asai(4)
asai_admin(1)
ulimit(1)

asai_test(1)

Name

`asai_test` — Test connection(s) with the ECS

Synopsis

```
asai_test [-?] [-m hostname/IPaddress] [link]
```

Description

For all ASAI nodes, the connection to the corresponding ECS is tested.

-m<machine>

This option has been added to support the use of this tool as a CV/LAN client. The user must specify the IP address or hostname of the CV/LAN server host.

`link`

The link that must be tested. All links are tested if this argument is omitted.

Operation

For each ASAI node, an ASAI heartbeat request is sent to the corresponding ECS. The response from the ECS is tested for correctness.

Return Value

For success, 0 is returned. If an error is detected, 1 is returned.

Output

If the response is correct, a `success` message is printed on `stdout`. If there is an error, the corresponding error message is printed on `stderr`.

NOTE:
The list of all links is in `config_data`.

Files

```
/dev/asai/asai
/usr/adm/asai/config_data
```

asai_trace(1)

Name

asai_trace — Decodes an ASAI message stream

Synopsis

```
asai_trace [-a] [-c] [-E] [-e] [-f] [-h] [-i] [-m] [-n] [-o]
[-q] [-r] [-s] [-v] [-?]
```

Description

asai_trace decodes an ASAI message stream and can be called with command-line options or options can be set using an online help facility.

-a

Use short mnemonics (abbreviated mode)

-c

Coding standard information is included

-E

Expect LAN GATEWAY format messages on stdin; for example,
esai_trace | nawk -f /usr/adm/isdn/bin/xlat.k | asai_trace -E

-e

Causes input to be taken from the command esai_trace

-f

Creates from 1 to 6 raw hex files (in addition to the trace information). The first file created is rawtrace.out.0. If it overflows, output goes to a second file rawtrace.out.1. If this file also overflows, rawtrace.out.2 is created and so on, up to a maximum of six files. If the sixth file overflows, the entire process begins again and rawtrace.out.0 is overwritten.

-h

Displays the help menu

-i

Causes input to be taken from the board using the command isdn_trace

`-m`

Message type information is included

`-n`

Public or private network and type of user information is included

`-o`

Displays the options help menu

`-q`

Q932 Messages are included

`-r`

Hex Code is added to the messages (raw mode)

`-s`

Messages are displayed in split-screen format. Messages from the adjunct appear on one side of the screen while messages to the adjunct appear on the opposite side. Abbreviated mode is forced.

`-v`

Single line per message mode (highly abbreviated)

`-?`

Displays the terse help menu

Examples

`asai_trace -i`

Trace input directly from the board using `isdn_trace`

`cat trace.in | asai_trace -r`

Decode `trace.in`, using raw mode, when a heartbeat is sent to the ECS by `asai_test`. The `trace.in` file is created by the command `isdn_trace > trace.in &`. Output similar to the following is generated:

```
<Heartbeat>       ADJ=>sw *0*crv=0004 sec=75.37
rcv cc 0 07538 90 11 00 08 02 00 04 64 96 1c 09 91 a1 06 02 01 03 02 01 b3 00 00 00 00
```

`asai_trace -h`

The verbose help menu is displayed.

Files

```
/usr/adm/isdn/bin/asai_trace
/usr/adm/isdn/bin/isdn_trace
/usr/adm/isdn/bin/esai_trace
```

See Also

```
asai_trace(1)
isdn_trace(1)
```

asai_ver(1)

Name

asai_ver — Print ASAI product and software version information

Synopsis

asai_ver

Description

ASAI product information is printed to stdout. This includes the name, date and version of the ASAI product.

Exit Codes

If there are no errors, the exit code is 0. Otherwise, the exit code is set to 1 to indicate an error.

Files

/usr/adm/asai/asai.Name
/usr/adm/asai/asai.Date

See Also

asai.Date(4)
asai.Name(4)

boot.bin(4)

Name

`boot.bin` — IPCI Board Bootstrap Loader Program File

Description

This data file, containing the IPCI Bootstrap Loader Program, is downloaded to the IPCI board by `ipci_on`. When this program is started, it is used to load the ISDN software onto the board.

Files

`/usr/adm/isdn/boot.bin`

See Also

`ipci_on(1)`
`pcisdn.bin(4)`

cmd(7)

Name

`cmd(7)` — LAN GATEWAY Streams Module Communication File

Description

The `cmd` file allows communication between certain OA&M commands and the LAN GATEWAY streams module.

This device is meant to be used by OA&M delivered commands and not by customer commands.

NOTE:
This file has the same major device number as the LAN GATEWAY admin(7) and the signal(7) files.

Files

`/dev/esai/cmd`

See Also

`admin(7)`
`signal(7)`

command(7)

Name

`command` — IPCI Streams Device Driver Communication File

Description

The command file allows communication between certain OA&M commands and the IPCI streams device driver. The IPCI streams device driver communicates with the IPCI board and with software on the board.

This device is meant to be used by OA&M delivered commands and not by customer commands.

Notes

This file has the same major device number as admin(7) and the signal(7) files.

Files

`/dev/isdn/ipci/command`

See Also

`admin(7)`
`signal(7)`

esai_alarm(1)

Name

esai_alarm — Turn on or off isdn_bri maintenance alarming

Synopsis

```
esai_alarm -a|d [-b1...-b8] [-t0...-tn] [-e] [-?]
```

Description

esai_alarm

Uses the suspend and resume routines to turn on or off the ASAI BRI port alarms associated with the link specified.

-a

Activates alarming

-d

Deactivates alarming

-b<linkNo>

Specifies the link, where <linkNo>=1...8, has sent the message

-t<numsec>

Sets the time-out to n seconds, where <numsec>=0...n. Default is -t3

-e

Displays the exit codes list shown below

-?

Displays the terse help message

Exit Codes

If esai_alarm fails, one of the following exit codes is returned:

0 = The alarm condition was changed per request

-1 = Failure due to inadequate memory

-2 = Open stream failure

-3 = Failure to send message to link

-4 = Failure to receive messages from link

-5 = time-out while waiting to receive a message

-6 = Layer 1 is down (could not attempt reset)

-7 = Layer 2 failed to recover from reset

-8 = State machine transitioning, try again

esai_trace(1)

Name

esai_trace — Trace LAN Gateway (ESAI) ASAI messages

Synopsis

esai_trace

Description

esai_trace is a general-purpose diagnostic tool. Its output is the actual ASAI message traffic to and from the ASAI-E LAN Gateway board. The message trace has the following general format:

```
<time stamp> <driver> <function> <slash> <sbn> <diagnostic> <link> <hexs>
```

The following table explains the format.

Table 2-1. Format of esai_trace-Generated ESAI Device Driver Messages

Mnemonic	Description	Example
<time stamp>	message time stamp number	15273
<driver>	name of module generating the entry	ESAI QP Klog
<function>	name of function handling the message	uplink
<slash>		/
<sbn>	starting byte number	0 12
<diagnostic>	message traffic has the value MSG_TRACE in this field. Only message traffic is useful.	MSG_TRACE
<link>	link number	1
<hexs>	GVI message contents in hex	96 1c 09 91

Received messages (those coming from the ECS) usually have uplink in the function field.

Transmitted messages (those going to the ASAI-E board and then the ECS) usually have a mnemonic function, for example, downlink.

The function name is separated from the starting byte number by a slash (`/`).

The starting-byte number indicates the byte number of the beginning (left most) byte of the hex field. The first line of a message has a byte number of 0. The second line has a number greater than zero (usually 12) and so forth for each additional line of the same message. The next message will have a byte number of 0 for the first line.

The diagnostic field indicates the purpose of the entry. Only trace entries with a diagnostic field value of MSG_TRACE are useful.

The driver QP frequently writes diagnostic messages. These messages are normal and can be ignored. Periodically, the kernel-reporting mechanism `klog` also writes entries. These can also be ignored.

Examples

`>esai_trace > trace.in &`

Turn on trace and send the output to the file *trace.in*.

`tail trace.in`

Display the end of the `trace.in` file. A heartbeat was sent to the ECS using the command `asai_test`. Output similar to the following is generated:

```
15273 ESAI downlink /0 MSG_TRACE[1]: 08 02 00 02 64 96 1c 09 91
15273 ESAI downlink /12 MSG_TRACE: a1 06 02 01 03 02 01 b3 00 00 00 00
15273 ESAI uplink /0 MSG_TRACE: 08 02 00 02 64 96 1c 09 91
15273 ESAI uplink /12 MSG_TRACE: a1 06 02 01 03 02 01 b3 00 00 00 00
15273 QP qp_rsrv/515 freeing unrecognized message of size 0x2
15273 QP qp_rsrv/516 msg[0]=0x8, msg[1]=0x1
```

Bugs

The kernel mechanism used by the device driver does not guarantee message order or integrity. Therefore, a sequence number is placed on each message by the ESAI device driver to allow detection of missing or out-of-order messages.

Turning on the message trace facility causes an additional system load. Sometimes the load is significant.

Files

```
/usr/adm/isdn/bin/esai_trace
/usr/adm/isdn/bin/asai_trace
```

See Also

`asai_trace(1)`

ipci(4)

Name

ipci — Pump and start administration for all IPCI boards

Synopsis

ipci

Description

ipci is a shell script that starts the ipci administration daemon process, ipci_admin (1) located in /etc/idrc.d.

Operation

The ASAI program is executed once per machine boot in the *multiuser init* level of 2. The name of the log file (asai_log) is changed to old_asai_log. The administration daemon [asai_admin(1)] is then started. When this is done, the standard error device is defined as /dev/console for ipci_admin (1).

Exit Codes

If pumping the board(s) is successful, 0 is returned as the exit code. Otherwise, 1 is returned.

Files

```
/usr/adm/asai/asai_log
/dev/isdn/ipci/admin
/dev/asai/asai
/dev/isdn/ipci/command
/dev/isdn/ipci/signal01
/dev/isdn/ipci/signal02
/dev/isdn/ipci/signal03
/dev/isdn/ipci/signal04
```

See Also

```
admin(7)
asai(7)
asai_admin(1)
asai_log(4)
command(7)
ipci_admin(1)
ipci_on(1)
signal(7)
```

ipci_admin(1)

Name

ipci_admin — Administer admin and signal files

Synopsis

ipci_admin

Description

ipci_admin sets up the protocol stack for signal device files and logs messages received from the admin(7) stream.

Operation

The ipci_admin program is executed once per machine boot. It is started by the ipci(4) script and becomes a daemon process.

This program has two tasks.

- To add the necessary module(s) into the signal(7) streams that carry ISDN D channel signaling or control information. In the case of CallVisor PC LAN GATEWAY (ESAI) links, ipci_admin adds the necessary modules to send and receive the same information over TCP/IP to and from the LAN GATEWAY circuit pack (MAPD) in the DEFINITY ECS.

  In the case of CallVisor LAN GATEWAY links, the ipci_admin daemon has the responsibility of initiating the connection from the CallVisor PC adjunct to the MAPD over the LAN. The connection is attempted immediately on start-up and if not successful, the daemon will try four times at 10-second intervals and every two minutes after that to connect to the MAPD.

- To receive status information from both the IPCI streams device driver and the ESAI driver module via the admin(7) stream. This information is logged in the log file [asai_log(4)]. (The device driver sends three types of information to ipci_admin. One type contains a status/error number that is converted to a corresponding ASCII message and placed into the asai_log file. The log_msgs(4) file is used to convert this number to the ASCII message. The sent type is ASCII information, of many different formats and byte lengths. The third type indicates boards that have been successfully pumped. This information is meant to be used for software debugging.)

- To receive and send requests and replies (via semaphores and message queues) from the various LAN GATEWAY OA&M commands.

If a nonrecoverable error condition occurs within the process, a message is produced and this program exits. Otherwise, this program waits for data from the admin stream. After processing the received data, this program then waits for more data from the admin stream.

The admin and signal stream files are always kept open by `ipci_admin`.

Messages

All of the following messages in the `asai_log` file are identified by a time stamp and the phrase `ipci_admin`:

- `Status Messages` — Two status messages are written to the `asai_log` file `/usr/adm/asai/asai_log(4)`by `ipci_admin`. One message indicates the beginning of the `ipci_admin` program. The other message indicates the successful setup of each signal stream.
- `Error Messages` — If `ipci_admin` encounters a nonrecoverable error situation, an error message is sent to the console if it cannot be logged, otherwise, the message is sent to the log file.
- `PCI Driver Status Messages` — For every status message received from the IPCI driver, a corresponding formatted message is written to the `asai_log` file.

CAUTION:
The ipci_admin daemon should never be killed with a kill -9; a kill (-15) should be used to allow the daemon to clean up the semaphores and message queues prior to exiting. If the semaphores or message queues become locked or hung, they can be freed using the ipcs(1) UNIX utility.

Exit Codes

This program, if it exits, will exit with a code of 1. Since this program is meant to be run as a daemon process, the exit code probably does not carry any great significance.

Files

```
/dev/isdn/ipci/admin
/dev/isdn/ipci/signal
/usr/adm/asai/asai_log
/usr/adm/asai/log_msgs
```

See Also

```
admin(7)
asai_log(4)
ipci(4)
log_msgs(4)
signal(7)
link_restart(1)
link_status(1)
```

ipci_off(1)

Name

`ipci_off` — Put specified IPCI board offline

Synopsis

```
ipci_off [n]
```

Description

This command is used to place board *n* offline. (The value of *n* can be 1, 2, 3 or 4.) When this is done, no new instances of communication between the UNIX ASAI adjunct and the ECS can be started. After executing this command, the board cannot be used for any signaling requests. To place it back online, it must be pumped using `ipci_on`.

Exit Codes

If an error is encountered or if the board is already offline, 1 is returned as the exit code. Otherwise, 0 is returned.

Notes

When a board has reached the offline state, it cannot be used for any signaling requests. To place it back online, it must be pumped.

Files

`/dev/isdn/ipci/command`

See Also

`command(7)`
`ipci_on(1)`
`ipci_stat(1)`

ipci_on(1)

Name

ipci_on— Pump Software to IPCI Board(s)

Synopsis

```
ipci_on [n]
```

Description

ISDN software is placed on the IPCI board(s) and that software is initialized. The board is then considered to be *online*.

If a value of n is specified, only that board is addressed. If no value of *n* is specified, the software is sent to all IPCI boards and all IPCI boards are placed into the *online* state.

If n is used, its value will be 1, 2, 3 or 4. Board numbers are assigned to the physical boards when the system administrator installs the software.

Operation

First, /usr/adm/isdn/boot.bin (the bootstrap loader) is copied to the stream file /dev/isdn/ipci/ command. When this has been done successfully, the IPCI streams device driver starts running this program on the board(s). Then /usr/adm/isdn/pcisdn.bin, the ISDN program, is copied. When this has been completed successfully, the pcisdn.bin program is started on the board(s) by the IPCI streams device driver. Then, the software on the boards is initialized.

Messages

All of the following messages written by ipci_on are identified by a date and time stamp followed by the program name ipci_on:

- Error Messages — If an error is encountered, a message is written to stderr and a copy of the same message is written to the log /usr/adm/asai/asai_log(4)file.
- Status Messages — Two status messages are written to the asai_log file. One message indicates the beginning of the pumping. The other message indicates the ending of the pumping.

Exit Codes

For success, 0 is returned. If an error is detected, 1 is returned.

Notes

This program is executed at boot time for all IPCI boards. It can be executed any other time for a board(s) that has successfully been placed offline. `ipci_off` should be used to take the board offline.

Files

```
/usr/adm/asai/asai_log
/usr/adm/isdn/boot.bin
/usr/adm/isdn/pcisdn.bin
/dev/isdn/ipci/command
```

See Also

```
asai_log(4)
boot.bin(4)
command(7)
ipci(4)
ipci_off(1)
pcisdn.bin(4)
```

ipci_stat(1)

Name

`ipci_stat` — Print Status of IPCI Board(s)

Synopsis

```
ipci_stat [n]
```

Description

If *n* is specified, a status report is printed for only that board. Otherwise, a status report is given for all boards. The report is printed to `stdout`.

For each board, four items are printed:

`line_control`

Normally, the adjunct computer (UNIX ASAI adjunct) should be in control of the interface between the ISDN phone and the ECS. When this is the case, `pc` is printed. If the adjunct software is not in control of this interface, `phone` is printed.

Line control is determined by the status of relays on the IPCI board. When they are *energized,* the phone and the ECS are connected by the `pc` software. When the relays are not *energized*, the phone and the ECS are directly connected by a metallic connection.

`switch_connection`

The connection to the ECS is either `active` or not *`active`.*

`switch_tei`

The board LAPD Terminal Endpoint Identifier (TEI) is printed as a decimal number and will not be greater than 127.

`link_status`

Indicates the status, up or down, of both levels 1 and 2.

Files

`/dev/isdn/ipci/command`

See Also

`command(7)`

ipci_test(1)

Name

`ipci_test` — Test an IPCI Board

Synopsis

```
ipci_test[n]
```

Description

IPCI board `n` is tested.

Operation

If the board is online, nondestructive (loop back) online tests are done. If the board is offline, board memory tests are done.

Return Value

For success, 0 is returned. If an error is detected, 1 is returned.

Output

The online/offline status is printed on `stdout` followed by the results of the test. Errors are printed on `stderr`.

Files

```
/dev/isdn/ipci/command
/usr/adm/isdn/boot.bin
```

See Also

```
boot.bin(4)
command(7)
ipci_off(1)
ipci_on(1)
ipci_stat(1)
```

ipci_ver(1)

Name

`ipci_ver` — Print ipci product and software version information

Synopsis

```
ipci_ver
```

Description

Ipci product information is printed to `stdout`. This includes the name, date and version of the pumpware, the ipci product and the device driver as well as the board identification number.

Exit Codes

If there are no errors, the exit code is 0. Otherwise, the exit code is set to 1 to indicate an error.

isdn_alarm(1)

Name

isdn_alarm — Turn on or off isdn_bri maintenance alarming

Synopsis

isdn_alarm — a|d [-b1...-b4] [-t0...-tn] [-e] [-?]

Description

isdn_alarm uses the suspend and resume routines to turn on or off the isdn_bri alarms that are sent from the ECS.

-a

Activates alarming

-d

Deactivates alarming

-b<numboard>

Specifies the board, where <numboard>=1...4, is sent the message

-t<numsec>

Sets the time-out to *n* seconds, where <numsec>=0...*n* ; the default is -t3

-e

Displays the exit codes list shown below

-?

Displays the terse help message

Exit Codes

If `isdn_alarm` fails, one of the following exit codes is returned:

0 = The alarm condition was changed per request

-1 = Failure due to inadequate memory

-2 = Open stream failure

-3 = Failure to send message to board

-4 = Failure to receive messages from board

-5 = Time-out while waiting to receive a message

-6 = Layer 1 is down (could not attempt reset)

-7 = Layer 2 failed to recover from reset

-8 = State machine transitioning, try again

isdn.Date(4)

Name

`isdn.Date` — Date file for UNIX ISDN

Description

This file contains the date and time the software for ISDN product was manufactured.

Files

`/usr/adm/isdn/isdn.Date`

isdn_11_r(1)

Name

`isdn_11_r` — Restart layer 1 and layer 2

Synopsis

`isdn_11_r`

Description

`isdn_11_r` will cause layer 1 and layer 2 to be restarted.

Operation

`isdn_11_r` is invoked as a single command with no options; the user is prompted for a board number from 1 to 4.

`isdn_11_r` is a script file that invokes `ipci_on` and `ipci_off` for the specified board. The diagnostics for `isdn_11_r` are the same as those for `ipci_off` and `ipci_on` and the exits codes are those of `ipci_on`.

See Also

`ipci_on(1)`
`ipci_off(1)`
`asai_hb(1)`

isdn_12_r(1)

Name

`isdn_12_r` — Restart layer 2

Synopsis

```
isdn_12_r [-b1...-b4] [-e] [-?]
```

Description

`isdn_12_r` sends a layer 2 reset to the board specified by the `-b` parameter. If layer 1 and layer 2 are both up, layer 2 will be taken down then brought back up. If layer 2 is already down, an attempt will be made to bring it back up.

`-b<numboard>`

Restarts the level 2 link for the specified board, where <numboard>=1...4, the default is `-b1`.

`-e`

Displays the exit codes list shown below

`-?`

Displays the terse help menu

Exit Codes

If `isdn_12_r` fails, one of the following exit codes is returned:

0 = The reset occurred per request

-1 = Failure due to inadequate memory

-2 = Open stream failure

-3 = Failure to send message(s) to board

-4 = Failure to receive message(s) from board.

-5 = Time-out while waiting to receive a message

isdn.Name(4)

Name

`isdn.Name` — Name file for UNIX ISDN

Description

This file contains the name of the ISDN software package. The content of the file is:

```
Lucent Technologies CallVisor ISDN
```

Files

`/usr/adm/isdn/isdn.Name`

isdn_trace(1)

Name

`isdn_trace` — Trace IPCI device driver messages

Synopsis

`isdn_trace`

Description

`isdn_trace` writes the kernel protocol stack messages to `stdout`. This is a general-purpose diagnostic tool. Only one of its outputs is the actual message traffic to and from the PC/ISDN board. The message trace has the following general format:

```
<time stamp> <driver> <function> <slash> <sbn> <diagnostic> <link> <hexs>
```

The following table explains the format.

Table 2-2. Format of `isdn_trace`-Generated IPCI Device Driver Messages

Mnemonic	Description	Example
<time stamp>	message time stamp number	15273
<driver>	name of module generating the entry	IPCI QP Klog
<function>	name of function handling the message	rcv cc gviSendSignal
<slash>		/
<sbn>	starting byte number	0 12
<diagnostic>	message traffic has the value MSG_TRACE in this field. Only message traffic is useful.	MSG_TRACE
<link.	link number	1
<hexs>	GVI message contents in hex	96 1c 09 91

Received messages (those coming from the ECS) usually have *rcv cc* in the function field. *cc* refers to the courtesy copy buffer which is a general-purpose board to the driver-transfer buffer.

Transmitted messages (those going to the IPCI board and then the ECS) usually have a mnemonic function, for example, `gviSendSignal`.

The function name is separated from the starting byte number by a slash (`/`).

The starting byte number indicates the byte number of the beginning (left most) byte of the hex field. The first line of a message has a byte number of 0. The second line has a number greater than zero (usually 12) and so forth for each additional line of the same message. The next message will have a byte number of 0 for the first line.

The diagnostic field indicates the purpose of the entry. Only trace entries with a diagnostic field value of MSG_TRACE are useful.

The driver QP frequently writes diagnostic messages. These messages are normal and can be ignored. Periodically, the kernel-reporting mechanism `klog` also writes entries. These can also be ignored.

Examples

```
>isdn_trace > trace.in &
```

Turn on trace and send the output to the file `trace.in`.

```
tail trace.in
```

Display the end of the `trace.in` file. When a heartbeat was sent to the ECS using the command `asai_test`, output similar to the following is generated:

```
15273 IPCI gviSendSignal/0 MSG_TRACE_LINK: 95 11 00 08 02 00 02 64 96 1c 09 91
15273 IPCI gviSendSignal/12 MSG_TRACE: a1 06 02 01 03 02 01 b3 00 00 00 00
15273 IPCI rcv cc/0 MSG_TRACE: 90 11 00 08 02 00 02 64 96 1c 09 91
15273 IPCI rcv cc/12 MSG_TRACE: a1 06 02 01 03 02 01 b3 00 00 00 00
15273 QP qp_rsrv/515 freeing unrecognized message of size 0x2
15273 QP qp_rsrv/516 msg[0]=0x8, msg[1]=0x1
```

Bugs

The kernel mechanism used by the device driver does not guarantee message order or integrity. Therefore, a sequence number is placed on each message by the IPCI device driver to allow detection of missing or out-of-order messages.

Turning on the message trace facility causes an additional system load. Sometimes the load is significant.

Files

```
/usr/adm/isdn/bin/isdn_trace
/usr/adm/isdn/bin/asai_trace
```

See Also

```
asai_trace(1)
```

lan_stat(1)

Name

`/usr/adm/isdn/bin/lan_stat` — Print Status of LAN Gateway links

Synopsis

```
lan_stat [-n] linkNo | [-a]
```

Description

If `n linkNo` is specified, a status report is printed only for that link. Otherwise, a status report is given for all LAN gateway links. The report is printed to `stdout`. `Lan_stat` with no arguments specified will print the status for only those LAN Gateway links that are administered.

If the `-a` option is specified, a long form of status is reported for all possible LAN Gateway links. The long form contains the following information:

Table 2-3. Long Form Status Values

Link	The LAN Gateway link number. Possible values are 1 to 8.
Dest	The Hostname of the ASAI-E board with which the link will connect.
Type	The type of link (LAN or ISDN) should always be LAN.
Status	The status of the link (numeric value in braces).
Num Retry	The number of times the LAN Gateway link has attempted to connect to the ASAI-E board.
time to Retry	The number of seconds until the next connect attempt will be made.
Link Fd	The file descriptor of the socket connection for the LAN link.
MUX Fd	The file descriptor of the multiplexor for the LAN link.
TLI Fd	The file descriptor of the TLI to TCP connection.

The short form (default) prints only Link, Dest, and Status.

Table 2-4. Common Status Values

Talking	This indicates "Layer 2" connectivity to the ECS.
TLiFailure	This indicates that the PC is unable to talk to the ASAI MAPD. It could be a LAN problem or the MAPD could be down or disconnected from the LAN.
BRiDown	This indicates that the CallVisor PC can connect to the MAPD via the LAN, but the MAPD is not communicating with the DEFINITY ECS.
InvalidClient	This status indicates that the MAPD board has not been administered to communicate with this CallVisor PC LAN link.
offline	This status indicates that this LAN link was put offline via the `link_offline(1)` OA&M command.

See Also

```
ipci_stat(1)
link_status(1)
```

link_alarm(1)

Name

/usr/bin/link_alarm — Suspend or resume alarms for a LAN link

Synopsis

esai_alarm(1)

Description

Link_alarm is a shell script that verifies that the specified link is a LAN link before invoking esai_alarm(1) to suspend or resume alarms.

See the manual page for esai_alarm(1)) for command line options.

See Also

esai_alarm(1)
isdn_alarm(1)

link_change(1)

Name

/usr/bin/link_change — Change the destination of LAN GATEWAY links

Synopsis

```
link_change -n <link_number> -h <hostname> -i <IP address>
```

Description

link_change is a shell script which is used to change the host destination of a LAN GATEWAY link.

-n <link_number> and either *-a* <hostname> or *-i* <IP address> must be specified.

link_change will verify the PC's ability to communicate over the LAN with the specified host or IP address (via ping) and print a warning if it is unable to do so, but will make the change anyway. After the verification of the host or IP address, link_change sends a message to the administration daemon to restart the specified LAN link and attempts to connect to the new destination. The user is prompted if communication with the administration daemon is not possible. link_change updates the LAN parameters file with new destination and exits.

link_change can only be invoked by the super-user.

link_change cannot be used to add or delete LAN GATEWAY link; the link to be changed must exist in the current configuration. Links may only be added or deleted during the ISDN/LAN GATEWAY installation process.

See Also

lan_change(1)
lan_stat(1)

link_offline(1)

Name

`/usr/bin/link_offline` — Take down LAN GATEWAY connection

Synopsis

```
link_offline &lt;link_number&gt;-1
```

Description

The `link_offline` command sends a message to the LAN GATEWAY administration daemon requesting the specified link(s) be put offline. Both TLI and TCP connectivity are broken and the retry strategy will be disabled. Any LAN link put offline will not be put back online until a `lan_restart` request has been made or the PC is rebooted.

The `-1` options makes the request for all LAN GATEWAY links.

See Also

`link_restart(1)`
`link_status(1)`

link_restart(1)

Name

`/usr/bin/link_restart` — Take down the LAN GATEWAY connection and restart it

Synopsis

`link_restart` `<link_number> -1`

Description

The *`link_restart`* command sends a message to the LAN GATEWAY administration daemon requesting the specified link(s) be taken offline and then immediately reestablished. Both TLI and TCP connectivity are broken.

The `-1` option makes the request for all LAN GATEWAY links.

See Also

`link_offline(7)`
`link_status(1)`
`lan_stat(1)`

link_status(1)

Name

/usr/bin/link_status — Print Status of IPCI Link(s) and LAN GATEWAY links

Synopsis

`link_status[-n] LinkNo [-a]`

Description

If `-n link No` is specified, a status report is printed for only that link. Otherwise, a status report is given for all administered links. The report is printed to `stdout`. `link_status` is a shell script that determines if a specified link belongs to an IPCI group or a LAN GATEWAY group and (via the appropriate OA&M call) prints the status of the link(s).

If `-a` is specified, a long version of the status for LAN GATEWAY links is generated.

See the manual page for `lan_stat(1)` for more details on the LAN GATEWAY link status.

See Also

`isdn_trace(1)`
`ipci_stat(1)`

log_msgs(4)

Name

`log_msgs` — UNIX ASAI adjunct Log Message File

Description

This file contains an explanation of certain status or error indications received by `ipci_admin (1)` from the IPCI device driver as it monitors the admin stream. Where applicable, it also contains remedial action information for the system administrator. Each status or error number is unique.

The information in this file is used by `ipci_admin(1)`. All characters in this file are printable ASCII characters. This file resides in the `/usr/adm/asai` directory.

There are two types of lines in this file: 1) comment lines and 2) message lines.

A comment line has “#” as its first character. It is followed by zero or more ASCII characters that are followed by a new-line character. A comment line may contain a maximum of 61 characters. This limit includes the “#” of character and the new-line character. “Blank” comment lines can be used to provide relief between successive messages.

A message for a particular status or error number may consist of one or more message lines. The format of a message line is:

<number><space><ASCII message><newline character>

“number” is the status or error number. It is a field of six characters with the number right justified. The number consists of digits only and is padded to the left with either blanks or zeroes.

<ASCII message> is from 0 to 53 ASCII characters.

Subsequent message lines of the same message contain the same status/error number also. The messages in the file must be in ascending status/error number order.

Files

`/usr/adm/isdn/log_msgs`

See Also

`ipci_admin(1)`
`asai_log(4)`

pcisdn.bin(4)

Name

pcisdn.bin — IPCI Board ISDN Program File

Description

This program file is downloaded to the IPCI board by ipci_on(1).

Files

/usr/adm/isdn/pcisdn.bin

See Also

ipci_on(1)
boot.bin(4)

signal(7)

Name

`signal` — IPCI and LAN GATEWAY Device Driver Communication Files

Description

The signal devices allow direct communication between a Lucent Technologies application program and the ECS.

There is a one-to-one correspondence between each signal stream and the IPCI board it serves or, in the case of LAN GATEWAY, between the signal stream and the MAPD. Hence, signal01 corresponds to board 1, etc.

Note the path difference for LAN GATEWAY links. LAN GATEWAY links reside in the directory `/dev/esai/signal0-8`.

Notes

These files will have the same major device number as the admin(7) and command(7) files. Their minor device number will be 1, 2, 3 or 4 depending on the board number.

The names of these files are shown below.

Files

```
/dev/isdn/ipci/signal01
/dev/isdn/ipci/signal02
/dev/isdn/ipci/signal03
/dev/isdn/ipci/signal04
/dev/esai/signal0[1-8]
```

See Also

```
ipci_admin(1)
admin(7)
command(7)
cmd(7)
```

CV/LAN Administration

OA&M changes for client and server

It is mandatory for the client and server both to connect to the same TCP/IP port. In order to meet this requirement, for the UNIX platforms in `/etc/services` file and for Windows NT platform in `C:\ntdir`[1]`\system32\drivers\etc\services` file, the following line must be administered:

CV/LAN XXXX/tcp # Callvisor LAN service

9999 is an unused/unreserved TCP port name.

1. ntdir is the Windows NT System directory where NT has been installed, for example, winnt 4.0.

Introduction to ASAI

3

This chapter is an overview of ASAI — what it does and how it works. Its purpose is to provide enough general information to design an application. Specific information, in the form of manual pages for the ASAI functions and capabilities, can be found in Chapter 8, ‘‘ASAI Capability Primitives’’ and Chapter 9, ‘‘Programming Manual Pages.’’ Coding examples appear throughout Chapter 4, ‘‘ASAI Library Functions.’’

The Adjunct-Switch Application Interface (ASAI) is an open standard interface that permits an application running on an adjunct computer to access telephony services provided by the ECS. Lucent Technologies’ CallVisor Switch-Adjunct Application Interface Platform for the UNIX system provides connectivity between an adjunct and the DEFINITY ECS via the ISDN personal computer interface (IPCI).

The ASAI library, a dynamic C library of functions and capabilities, provides an application programmer with easy access to the IPCI so that his or her application program can utilize the DEFINITY ECS services or features. The ASAI library also allows the ECS to access services provided by the adjunct, such as Routing and Heartbeat.

The ASAI library provides a set of standard capabilities for performing the following tasks:

- Managing active calls to which the adjunct is not a party (called “third party calls”); for example, transferring a call or putting a call on hold
- Querying or setting values of various indicators maintained by the ECS — for example, setting on the message-waiting indicator
- Activating or deactivating a number of the ECS features — for example, call forwarding or send all calls
- Routing incoming and outgoing calls

- Testing a communication path to ensure that the ASAI link is operating
- Terminating (aborting) calls when necessary
- Reading event reports to provide a complete audit trail of all of these activities

Each group of capabilities is called an Application Service Element or ASE.

Capabilities are the building blocks of ASAI. They are sent from the adjunct to the ECS or received by the adjunct from the ECS. The exchange of capabilities is managed by the ASAI library functions `asai_send()` and `asai_rcv()`.

Library functions perform another very important task: controlling the communication path across which capabilities are exchanged. The functions `asai_open()`, `asai_set_env()`, `asai_get_env()`, and `asai_close()` are used to establish and terminate a communication path.

In addition, `asai_errval()` provides error information to be used in error-handling routines.

Terms and Concepts

ASAI is an interface between an adjunct and the ECS; however, a number of terms can be used instead of or in addition to “adjunct” and “ECS.” In this book, the following terms are used with the meanings shown below:

adjunct	A PC, used to communicate with a PBX switch, in this case the DEFINITY ECS.
client	A program that sends service requests to another program (generally on a different processor) called the server. Usually the ECS is the server and CV/PC is the client. However, this is not the case with adjunct routing and heartbeat requests where the roles interchange; for example, heartbeat requests and confirmation can come from both the sides at any time.
server	A server is a program or machine that responds to a particular class of request. In this document, “server” refers to one of the following: 1. A program that responds to requests from another program (generally on a different processor) called the client. Usually the ECS is the server and CV/PC is the client except for adjunct routing and heartbeat requests where the roles interchange; for example, heartbeat requests and confirmation can come from both the sides at any time. 2. CV/LAN server is a program that allows CallVisor PC applications to run remotely.
library	The ASAI library, provider of ASAI services.
program	The application program that uses the services of the ASAI library, whether as a client, a server, or both.

The term “application” is used only in the phrase “application program” or “application programmer.”

To the ECS, “application” means an adjunct entity that requests and receives ASAI services or capabilities. Although more than one application can reside on a single adjunct, the ECS is unable to distinguish between these applications. As a result, the adjunct and all resident applications are treated as a single application and these terms are used interchangeably.

This ambiguity is avoided by restricting the use of the term “application” throughout this document.

The building blocks of the ASAI library are “capabilities.” Requests and indications are capabilities. For example, Third Party Make Call is a request for setting up a call; Third Party Make Call is an ASAI capability. An Event Report is an indication that an event has occurred; Event Report is a capability.

ASAI capabilities can be further broken down into capability “primitives” that perform a single task (requesting a service, responding to a request, confirming a request or indicating that a request has been fulfilled). A primitive is an atomic message passed by the library. For example, the Heartbeat capability consists of two primitives: C_HB_REQ and C_HB_CONF. For more details on Capability primitives see ‘‘ASAI Capability Manual pages’’ on page 9-21 in Chapter 9, ‘‘Programming Manual Pages.’’

Groups of capabilities—for example, all the Third Party capabilities—are “Application Service Elements” (ASEs) or “capability groups.”

The term “association” is critical to understand ASAI. An association is a single invocation of an ASE between an adjunct and the ECS. An association is represented by a unique CRV/link combination. (The Glossary at the end of this document contains definitions of additional terms and acronyms that relate to ASAI.) The terms ASE instance and Single Application Object (SAO) are synonymous with association. An active association is one that applies to an existing call on the ECS or to an extension on the call.

Associations can affect a ‘‘call” or a ‘‘party” to a call (or both). A call usually has two or more parties (that can be identified by their ‘‘extensions” as well as by party_id). An extension can be three, four or five numbers that internally identify a station or voice/data terminal. Extension can also mean a 10-digit number that includes area code and a full 7-digit telephone number.

Important Note about Heartbeat

It is the responsibility of the CallVisor PC adjunct to respond to the ASAI heartbeat messages that are periodically sent from the ECS. This is the case for every CallVisor PC adjunct and for each ASAI link that is active.

The ECS sends a register message (with the operation value set to heartbeat), to the adjunct every two minutes. In order to respond to this request, the application must have previously called the function asai_set_env with server_type set to C_MAINT_SER for the application to be considered a maintenance server. As a maintenance server, the application must respond to every C_HB_REQ capability it receives from the ECS with a C_HB_CONF. If the adjunct fails to respond to three consecutive heartbeat requests from the ECS, the ECS takes down Layer 1 for five seconds and all active associations are aborted. If the application program itself has to perform the maintenance server role, great care must be taken to ensure that the application does not block waiting on other events which would cause it to miss a heartbeat request. A separate process should be used to perform the maintenance server role.

Beginning with Release 2.2 of the CallVisor ASAI product, an OA&M process, asai_hb, is available to accomplish this maintenance task.

The ASAI Library Functions

The functions provided by the ASAI library are:

```
asai_open()
asai_close()
asai_set_env()
asai_get_env()
asai_send()
asai_rcv()
asai_errval()
```

These functions afford access to the services defined by the full set of ASAI capabilities. The basic purpose of the functions is twofold: to manage the communication path over which ASAI capabilities are requested and acknowledged and to manage the exchange of information over an established communication path.

Table 3-1 gives a brief description of the ASAI library functions that manage the communication path:

Table 3-1. ASAI Library Functions for Managing the Communication Path

Function	Description
`asai_open`	Opens a communication path to the capability provider, using either a blocking or nonblocking mode of operation. The communication path is identified by a file descriptor.
`asai_close`	Terminates the communication path from the application process to the capability provider.
`asai_set_env`	Allows the application process to set a characteristic of an opened communication path. The first characteristic to be set must be the destination node for service requests. Other characteristics that can be set are the type of service and node on which requests for this service are to be received, called the service/destination pair. (This tells the ASAI library to place indications of the specified ASE from the specified node on this communication path.)
`asai_get_env`	Requests information from the ASAI library concerning a characteristic of an opened communication path. Characteristics that can be queried are the following:
	Destination node to which service requests will be sent.
	Version of the ASAI library implementation. (This tells the application whether the features of the current ASAI library are sufficient for the application.)
	Type of service and node that will send initiating indications. (This tells the application that initiating indications for the specified type of service and for the specified node will be received from ECS.)

The other ASAI library functions are described briefly in Tables 3-2 and 3-3 below.

Table 3-2. ASAI Library Functions for Exchanging Information

Function	Description
`asai_send`	Initiates, continues or terminates an association by transmitting request or confirmation primitives that identify specific capabilities.
`asai_rcv`	Receives primitives that identify specific capabilities to initiate, continue, or terminate an association.

Table 3-3. ASAI Library Function for Handling Errors

Function	Description
`asai_errval`	Allows the user to specify an error message, followed by the exiting error return value, written to `stderr`.

In general, ASAI library functions return 0 when successful and -1 on failure. Exceptions are noted in the manual pages in Chapter 8, ‘‘ASAI Capability Primitives’’ and Chapter 9, ‘‘Programming Manual Pages.’’

Application Service Elements

The ASAI Application Service Elements (ASEs) support ASAI functions. ASEs, also referred to as “capability groups,” are the high-level building blocks from which specific applications are constructed. Each ASE defines a series of relationship interactions between a client and a server: third party call control, event notification, value query and so on.

Each capability provides a specific type of interaction between the client and the server: third party make call, third party reconnect, third party selective hold, third party selective listening, etc.

The client and server issue a sequence of calls to the **send** and **receive** functions in which capability primitives are exchanged. The capability primitives request, acknowledge and terminate an interaction.

Table 3-4 lists the ASAI ASEs and the capability primitives in each group.

Table 3-4. ASAI Capabilities by Capability Group (ASE)

ASE	Capability	Description
Third Party Call Control	C_3PAD	Auto Dial
	C_3PAD_CONF	Confirm Auto Dial
	C_3PANS	Answer
	C_3PANS_CONF	Confirm Answer
	C_3PCC	Clear Call
	C_3PCC_CONF	Confirm Clear Call
	C_3PCE	Call Ended
	C_3PDC_CONF	Confirm Domain Control
	C_3PDC_REQ	Domain Control
	C_3PDCE	Domain Control End
	C_3PM	Merge
	C_3PM_CONF	Confirm Merge
	C_3PMC_REQ	Make Call
	C_3PMC_CONF	Confirm Make Call
	C_3PR	Reconnect
	C_3PR_CONF	Confirm Reconnect

Table 3-4. ASAI Capabilities by Capability Group (ASE) — *Continued*

ASE	Capability	Description
Third Party Call Control (continued)	C_3PRC	Relinquish Control
	C_3PRC_CONF	Confirm Relinquish Control
	C_3PREDIR	Redirect Call
	C_3PREDIR_ACK	Confirm redirect Call
	C_3PSD	Selective Drop
	C_3PSD_CONF	Confirm Selective Drop
	C_3PSDS	Send DTMF Signals
	C_3PSDS_CONF	Confirm Send DTMF Signals
	C_3PSH	Selective Hold
	C_3PSH_CONF	Confirm Selective Hold
	C_3PSL_DISC	Listen Disconnect
	C_3PSL_DISC_ACK	Confirm Listen Disconnect
	C_3PSL_RECONN	Listen Reconnect
	C_3PSL_RECONN_ACK	Confirm Listen Reconnect
	C_3PSSC_CONF	Confirm Conference Call
	C_3PSSC_REQ	Make Conference Call
	C_3PTC_REQ	Take Control
	C_3PTC_CONF	Confirm Take Control
Event Notification	C_EN_CAN	Cancel EN
	C_EN_CAN_CONF	Confirm Cancel
	C_EN_CONF	Confirm EN
	C_EN_END	End EN
	C_EN_REP	EN Report
	C_EN_REQ	Request EN
	C_EN_SCN	Stop EN
	C_EN_SCN_CONF	Confirm Stop EN
Maintenance (Heartbeat)	C_HB_CONF	Confirm HB
	C_HB_REQ	Request HB
Request Function	C_RF_CONF	Confirm RF
	C_RF_REQ	Request RF
Routing	C_RT_END	End RT
	C_RT_REQ	Request RT
	C_RT_SEL	Select RT
Set Value	C_SV_CONF	Confirm SV
	C_SV_REQ	Request SV

Table 3-4. ASAI Capabilities by Capability Group (ASE) — *Continued*

ASE	Capability	Description
Value Query	C_VQ_CONF	Confirm VQ
	C_VQ_REQ	Request VQ
	C_VQ_RESP	VQ Response

In addition, the ASAI library provides two types of `C_ABORT` capabilities, as shown in Table 3-5. The two ASAI abort services are referred to as `user aborts` and `provider aborts`.

A user abort is a program-initiated abort and a provider abort is a library-initiated abort. The `C_ABORT` capability is applicable to all ASEs. The application can send and receive an abort at any time. An abort is always terminating.

Table 3-5. ASAI Abort Capabilities

ASE	Capability	Description
all	C_ABORT(P_ABORT)	library-initiated (provider) abort
all	C_ABORT(U_ABORT)	ECS-initiated (user) abort

ASAI Capabilities

The library functions `asai_send()` and `asai_rcv()` are used to exchange capabilities. Each ASAI capability is specific, performing a single task. These tasks are classified as initiating, continuing or terminating. The capabilities are also classified as acknowledged or unacknowledged.

Table 3-6 lists each capability primitive and shows how it is classified. Note that a number of the confirmation primitives are classified both as ‘‘unacknowledged - continuing’’ and ‘‘unacknowledged - terminating.’’ In such cases, the primitive is continuing when confirmation is positive and terminating when confirmation is negative.

Table 3-6. Classification of ASAI Capabilities

	Acknowledged			Unacknowledged		
	Init.	**Cont.**	**Term.**	**Init.**	**Cont.**	**Term.**
C_3PAD		x			x	
C_3PAD_CONF					x	
C_3PANS		x				
C_3PANS_CONF					x	
C_3PCC		x				
C_3PCC_CONF					x	x
C_3PCE						x
C_3PDC_CONF					x	x
C_3PDC_REQ	x					
C_3PDCE						x
C_3PM		x				
C_3PM_CONF					x	
C_3PMC_CONF					x	x
C_3PMC_REQ	x					
C_3PR		x				
C_3PR_CONF					x	
C_3PRC			x			
C_3PRC_CONF					x	x
C_3PREDIR	x				x	

Table 3-6. Classification of ASAI Capabilities — *Continued*

	Acknowledged			Unacknowledged		
	Init.	**Cont.**	**Term.**	**Init.**	**Cont.**	**Term.**
C_3PREDIR_ACK	x				x	
C_3PSD		x				
C_3PSD_CONF					x	
C_3PSDS		x				
C_3PSDS_CONF					x	
C_3PSH		x				
C_3PSH_CONF					x	
C_3PSL_DISC		x				
C_3PSL_DISC_ACK					x	
C_3PSL_RECONN		x				
C_3PSL_RECONN_ACK					x	
C_3PSSC_CONF					x	
C-PSSC_REQ		x				
C_3PTC_CONF					x	x
C_3PTC_REQ	x					
C_ABORT						x
C_EN_CAN		x				
C_EN_CAN_CONF					x	x
C_EN_CONF					x	x
C_EN_END						x
C_EN_REP					x	
C_EN_REQ	x					
C_EN_SCN		x				
C_EN_SCN_CONF					x	
C_HB_CONF						x
C_HB_REQ	x					
C_RF_CONF						x
C_RF_REQ	x					

Table 3-6. Classification of ASAI Capabilities — *Continued*

	Acknowledged			Unacknowledged		
	Init.	**Cont.**	**Term.**	**Init.**	**Cont.**	**Term.**
C_RT_END						x
C_RT_REQ	x					
C_RT_SEL					x	
C_SV_CONF						x
C_SV_REQ	x					
C_VQ_CONF						x
C_VQ_REQ	x					
C_VQ_RESP					x	

The programmer is not restricted to a specific number of capabilities that can be supported on a given communication path; however, he or she is restricted to a single node for all capabilities on a given path.

ASAI Library Functions

4

The ASAI library functions provide the application program with an easy method of sending and receiving information (in the form of ASAI capabilities) to and from the ECS. Before you can send or receive information, however, you must establish a communication path. This is a two-step procedure, requiring you to first open a path, using `asai_open()`, and then to identify the ECS link over which your program will communicate. The ECS is identified by a node identifier, set with `asai_set_env()`.

`asai_set_env()` identifies applications as being clients and/ or servers. A client application initiates a request for service and a server application responds to a request for service.

The ASAI library does not allow an application to assume an inappropriate role. For those capabilities for which it is registered as a client, an application can only send messages defined for clients. It cannot send server-type messages. Conversely, when an application is registered as a server, it can only send messages defined for servers. It cannot send client-type messages.

This chapter describes each function briefly and shows how it works. The final section of this chapter is a sample program. The functions are presented in logical, rather than alphabetical order; thus, `asai_open()` comes first and `asai_close()` comes last, as indicated in Table 4-1.

Table 4-1. ASAI Library Functions: A Typical Application

Function	Purpose
`asai_open`	Open communication path
`asai_errval`	Write error message if function call fails
`asai_set_env`	Bind communication path to a node ID or set server for more links
`asai_get_env`	Check library version (optional)
`asai_send`	Send a capability
`asai_rcv`	Receive a capability
`asai_close`	Close the communication path (optional)

It is possible to process more than one ASE on a communication path; however, each communication path can be used to service only a single node. Thus, if an application wishes to use a communication path to send service requests, all the requests must be made to the same destination node as that of the initiating capability. If an application wishes to use a communication path to receive capabilities, requests must come via the port used to receive the initiating capability.

In general, the ASAI library functions are not reentrant; therefore, you should avoid using them in signal handlers, or use `sig_hold()` for the pertinent signals before calling any ASAI function.

All seven of the library functions are declared as `long`. To prevent unnecessary messages from `lint`, you may want to use `0L` to test for successful completion rather than just `0`. In some cases, you may also need to use a cast to convert expressions to type `long`. There are examples of casting in some of the code fragments later in this chapter.

Detailed information on each function can be found in the manual pages in Chapter 9, “Programming Manual Pages.”

asai_open( )

The `asai_open()` function is the first of the library functions that your program calls. This function opens a communication path (stream) and returns file descriptor `fd` if the call is successful; the function returns `-1` if the call fails. `asai_open()` requires two arguments; `pathname` and `ndelay_flag`. The `pathname` is `/dev/asai/asai` or a machine name for CV/LAN.

The `ndelay_flag` determines whether you will be operating in synchronous mode with blocking and no delay, or asynchronous mode with delay and no blocking. For synchronous mode, set `ndelay_flag` to 0. For asynchronous mode, set `ndelay_flag` to `O_NDELAY`.

Code Example 4-1 below is a typical **open** routine.

```
open_routine()
{
    int fd;
    if ((fd = asai_open("pathname",0)) < 0
    {
          asai_errval("error message");
          exit(-asai_errno);
    }
    .
    .
    .
}
```

Figure 4-1. Opening a Communication Path

Note that the routine above calls `asai_errval()` if the value returned is less than 0.

See the sample code called `asai_open()` on line 52 in Figure 4-9 on page 4-15.

asai_errval ( )

When an error occurs, an error value is made available in the variable `asai_errno` of data type `long`. Once `asai_errno` is set, its value is not reset by successful function calls. If an error is returned by an ASAI library function, there is state transition, either for the communication path or for the instance of communication. `asai_errval()` lets you specify an error message that is written to the standard error device along with the system-supplied ASAI error message.

The `asai_errval()` function should be used after every failed function call in your program.

Your error message is specified in a null-terminated string, `mes_buf`, of maximum size `C_MESIZE` (the length set for `C_MESIZE` in the `samp.hdrs` file supplied with the system is 80 characters).

You can specify a message directly, as in the code example in Figure 4-2, or you can specify it indirectly, as in the code example in Figure 4-3 on page 4-5.

```
.
.
.
if (fd < 0)
{
     asai_errval("asai_open(/dev/asai/asai) failed:");
     exit(i);
}
.
.
.
```

Figure 4-2. Using asai_errval (), Example 1

NOTE:
The variable, `asai_errno`, can no longer be defined with `extern`. It must be included using <`asai/asai_err.`h>.

```
.
.
char pathname[];
char buf[C_MESIZE];
char *p;
if ( asai_function(args) < 0 )
{
      "ASAI function failed\n"
      asai_errval “ASAI function failed”;
}
.
.
```

Figure 4-3. Using asai_errval (), Example 2

Note that the colon (:) included as part of the message in Figure 4-2 on page 4-4 is not necessary. The message written to standard error consists of your specified message, a colon, and then the ASAI message. For a full listing of the ASAI error messages, see Chapter 7, ‘‘Error Messages.’’

`asai_errval` returns `0` on success and `-1` on failure. The sample code at the end of this chapter uses a conditional call to `asai_errval()` after every function call (lines 54, 59, 69, 77, 105, 149 and 154). If the previous function call fails, `asai_errval()` is called.

asai_set_env( )

After opening a communication path, the next step is to establish an environment that will support your program. The `asai_set_env()` function enables you to set one characteristic at a time. You *must* set the node identifier using the `C_NODE_ID` characteristic, if you are going to establish new association.

`asai_set_env()` has two mandatory and one optional arguments. The file descriptor and the characteristic to be set are mandatory. Depending on the characteristic you are setting, you also may have to specify a value.

The characteristic `C_NODE_ID` establishes the name of the node from which your application will initiate service request. This must be done first. If an application needs services from more than one node, it must open the library once for each node it needs services from. This means that multiple file descriptors must be managed.

You must use the characteristic `C_SERVER` to specify the name(s) of the node(s) for which your application will provide services; that is, it will respond to the ECS initiated requests. If your application does not handle the ECS requests, you do not need to do this. Note that each node will allow only one server for each kind of service.

If your application does not handle the ECS requests, it can provide service to one or more nodes for each file descriptor.

The code example in Figure 4-4 on page 4-6 uses the `C_NODE_ID` characteristic to supply the node identifier. Note that the alternative compares the return value from `asai_set_env()` to a `long 0`.

```
if (asai_set_env(fd,C_NODE_ID,node_id) < 0 )
{
    asai_errval ("asai_set_env (C_NODE_ID) failed");
    exit (-asai_errno);
}
    /*  an alternative  */
if (asai_set_env(fd,C_NODE_ID,argv[1]) < 0L)
```

Figure 4-4. Setting the Library Environment: Node ID

You may also need to specify that the application is to function as a server for an ECS routing and maintenance requests. A single application can function as a server for more than one node on a single file descriptor. (A node can be equated to a link or port on an ASAI connection.) Note that each node will allow only one server for each kind of service. To do this, set the `num_node` field of the `server_type_t` structure to the number of nodes the application is to serve. Set the `*buf` field to point to an array of structures of the same length. Set the name of the node for example, signal01, in the `node_id` field or set all the names together of the server types desired in the `server_type` field.

The code example in Figure 4-5 illustrates the use of `asai_set_env()` to set the server type. Note that each call to `asai_set_env` overwrites the previous setting on a per-characteristic basis.

```
.
.
.
setarg.numnode = 4;
setarg.buf = srvrs;
strcpy (srvrs[0].nodeid, “signal01”);
srvrs[0].server_type = C_RT_SER | C_MAINT_SER;
.
.
.
if (asai_set_env (fd, C_SERVER, &setarg) < 0)
{
    asai_errval ("asai_set_env (C_SERVER) failed:");
    return asai_errno;
}
.
.
.
```

Figure 4-5. Setting the Environment: Establishing a Server

When you are specifying servers, be sure to include maintenance (`C_MAINT_SER`) in the list or invoke the heartbeat OA&M process prior to running your application. If you do not, your program will be unable to respond when the ECS sends a heartbeat. If this happens, the ECS tears down the link.

You should also include the routing (`C_RT_SER`) server in order to receive route requests.

The sample code at the end of this chapter calls `asai_set_env ()` on line 57, in Figure 4-9.

NOTE:
Be aware that each call to `asai_set_env` overwrites the previous value(s) on a per-characteristic basis.

asai_get_env( )

The `asai_get_env` function allows you to check the version of the ASAI library, or retrieve your node id and server settings.

The version number of the ASAI library includes three values: a major value, a minor value and a delta value. A change to a major value indicates significant functional change to the ASAI library; for example, addition of new capabilities. A change to a minor value indicates change within the existing capabilities; that is, an enhancement. A change to a delta value indicates a correction and does not represent any increased functionality. The parameters `major_ver, minor_ver` and `delta_ver`, respectively, represent these values.

`asai_get_env` also lets you check which node is handling requests from you and which services you are offering to the ECS.

asai_send( )

asai_send() and asai_rcv() together are the central functions in the library. These two functions convey all ASAI capabilities and data across an association.

asai_send() has only three parameters: file descriptor fd, a pointer to the buffer that contains the message to be sent and the length of the buffer. It is your responsibility to map the message to the appropriate structure (one of the structures contained in the union asai_info_t); otherwise, the message cannot be interpreted correctly on the receiving side.

For example, if you are sending a third party make call request, you map the message to the structure a3pmc_info_t (shown in part below).

```
typedef struct {
        asai_common_t   asai_common;
        char            *calling_num;
        char            *called_num;
           .
           .
           .
}a3pmc_info_t;
```

(All of the structures contained in asai_info_t start with asai_common_t, referred to as the header.) asai_common_t consists of four long data types as follows:

```
typedef struct {
        capability_t    capability;
        primitive_t     primitive_type;
        long            sao_id;
        long            reserved;
}asai_common_t;
```

In the case of the third party make call request, the capability is C_3PMC_REQ and the primitive_type is C_REQUEST.

With asai_send (), the library encodes the structure you have filled and forwards the result of the encoding to the ECS. With asai_rcv (), when the library receives a message from the ECS, it decodes the message, fills the necessary data into the appropriate structure and passes the structure to the application.

In addition to this mapping responsibility, you must also ensure that the length of your buffer is sufficient to contain the entire message. (If the buffer is too small, results are unpredictable.)

The maximum useful size of the user buffer is size of (asai_info_t), but you can select a size appropriate for your application. The example below uses size of (asai_common_t). If you are not sure, use size of (asai_info_t).

```
if ( asai_send(fd, &mybuf, sizeof(asai_common_t)) < 0 )
{
    asai_errval("asai_send");
    exit (-asai errno);
}
```

Figure 4-6. asai_send () Function

asai_send() returns the size of the message on success and -1 on failure. Technically, a return code of 0 is not an error; however, it does indicate that only the control portion of a message has been sent. This is probably not what you intended.

The sample code at the end of this chapter calls asai_send() on line 70 and on line 147, in Figure 4-9 on page 4-15.

asai_rcv( )

This function reads messages from the communication path specified by `fd`. Like `asai_send()`, it expects a buffer and the length of the buffer to be passed to it.

```
asai_rcv_return_value = asai_rcv (fd, input_buf, BUFSIZ );
if ( asai_rcv_return_value < 0L )
{
    p = "receive data from switch for node %s failed\n";
    sprintf ( buf, p, node_id );
    fflush ( stdout );
    fflush ( stderr );
    asai_errval ( buf) ;
    return ( asai_errno ) ;
}
```

Figure 4-7. asai_rcv () Function

`asai_rcv()` returns the size of the buffer on success and `-1` on failure. A return code of `0` does not indicate an error; it merely indicates that the received message was intended for the library and no program data is available at this time.

In the event that a received message will not fit in the buffer supplied by the programmer, the ASAI library returns a `-1` (error) and sets `asai_errno` to `-14` (`C_BADLNG`). If a program ignores this error and simply calls the library again, an infinite loop is created. All application programs must check for the presence of a `C_BADLNG` error and provide a larger buffer, if it occurs. To avoid length problem, you can use `asai_info_t`.

The sample code in Figure 4-9 calls `asai_rcv()` on lines 75, 103 and 152.

asai_close( )

You may never have to use `asai_close()`. However, the UNIX system imposes a limit on the number of files that can be open at one time. If your application opens a large number of files, you may have to call `asai_close()` to close specific communication paths. `asai_close()` closes the communication path identified by `fd`. It returns `0` on success and `-1` on failure.

You may also want to write a function to close your application under certain conditions, for example to stop the system from hanging on an `asai_rcv()` if the ECS fails to return the expected response. Figure 4-8 shows `asai_close()`.

```
void appy_close ()
{
    extern int fd;
    extern char pathname[];
    char buf[C_MESIZE];
    char *p;
    if ( asai_close(fd) < 0 )
    {
          p = "close of ASAI communication path %s failed\n";
          sprintf ( buf, p, pathname );
          asai_errval (buf );
          fflush ( stdout );
          fflush ( stderr );

    }
    return;
```

Figure 4-8. Closing the Communication Path

Sample Code

The sample code reproduced here tests the Event Notification capabilities.

First the program determines whether the domain type is an ACD split or a call vector. Then it requests appropriate event reports.

The C_EN_REQ capability is sent on lines 65-74 in Figure 4-9. If an abort capability is received (lines 75 to 91) the program exits. Similarly, if C_EN_CONF is received with a negative acknowledgement (lines 92 to 98), as shown in Figure 4-9, the program exits. Otherwise, an event report is received. The request is cancelled by sending the C_EN_CAN capability.

```
/*
 * :set tabstop=4 shiftwidth=4
 *
 * usage: entest <node> acd|vec <num> [<# reports>]
 *     entest signal01 acd 9999
 *               # receive reports from acd domain
 *     entest signal01 vec 9999 10
 *               # receive 10 reports from vector domain
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/stropts.h>
#include <asai/asai_def.h>
#include <asai/asai_str.h>
#include <asai/asai_err.h>

extern char *caps[];
extern char *typs[];
extern char *causes[];
extern char *modes[];
extern char *names[];

main(argc, argv)
   char *argv[];
{
   int fd, cnt;
   en_buf_t en;
   asai_info_t rsp;

   if (argc < 4)
   {
   usage:
      fprintf(stderr, "usage: %s <node> acd|vec <num> \
             [<# reports>]\n",argv[0]);
      exit(1);
   }
   if (strcmp(argv[2], "acd") == 0)
   {
      en.acd_grp_info.domain_type = C_ACD_GROUP;
      en.acd_grp_info.domain_ext = argv[3];
   }
   else if (strcmp(argv[2], "vec") == 0)
   {
      en.cv_info.domain_type = C_CALL_VECTOR;
```

Figure 4-9. Sample Code — Testing the Event Notification Capabilities

```
      en.cv_info.domain_ext = argv[3];
   }
   else
   {
      goto usage;
   }
   if ((fd = asai_open("/dev/asai/asai", 0)) < 0)
   {
      asai_errval("asai_open(dev/asai/asai) failed:");
      exit(-asai_errno);
   }
   if (asai_set_env(fd, C_NODE_ID, argv[1]) < 0)
   {
      asai_errval("asai_set_env(C_NODE_ID) failed:");
      exit(-asai_errno);
   }


   cnt = argc == 5 ? strtol(argv[4], 0, 0) : 1;
   en.en_common.asai_common.capability = C_EN_REQ;
   en.en_common.asai_common.primitive_type = C_REQUEST;
   en.en_common.asai_common.sao_id = 0;

   printf("Sent: C_EN_REQ (C_REQUEST) on clid 0\n");
   if (asai_send(fd, &en, sizeof(en))<0)
   {
      asai_errval("asai_send");
      exit(-asai_errno);
   }
   if (asai_rcv(fd, &rsp, sizeof(rsp))<0)
   {
      asai_errval("asai_rcv");
      exit(-asai_errno);
   }
   printf("Received: %s (%s) on clid %d",
      caps[rsp.asai_common.capability],
      typs[rsp.asai_common.primitive_type],
      rsp.asai_common.sao_id);
   if (rsp.asai_common.capability==C_ABORT)
   {
      printf("\tabort type:%s cause:%s\n",
         rsp.abort_info.abort_type == C_PROV_ABT
            ? "C_PROV_ABT" : "C_USER_ABT",
         causes[rsp.abort_info.cause_value]);
      exit(1);
```

Figure 4-9. Sample Code — Testing the Event Notification Capabilities — *Continued*

```
}
else if (rsp.asai_common.capability==C_EN_CONF &&
    rsp.asai_common.primitive_type==C_NEG_ACK)
{
   printf("\tcause:%s\n", \
      causes[rsp.en_rsp.en_nak.cause_value]);
   exit(2);
}
printf("\n");

for (;cnt--;)
{
   if (asai_rcv(fd, &rsp, sizeof(rsp))<0)
   {
      asai_errval("asai_rcv");
      exit(-asai_errno);
   }
   printf("Received: %s (%s) on clid %d",
      caps[rsp.asai_common.capability],
      typs[rsp.asai_common.primitive_type],
      rsp.asai_common.sao_id);
   switch (rsp.asai_common.capability)
   {
   case   C_ABORT:
      printf("\tabort type:%s cause:%s\n",
         rsp.abort_info.abort_type == \
               C_PROV_ABT
             ? "C_PROV_ABT" : "C_USER_ABT",
         causes[rsp.abort_info.cause_value]);
      exit(3);
   case   C_EN_END:
      if (rsp.ene_info.cause_value != C_NUSE_LONG)
      {
         printf("\tcause:%s\n", \
                        causes[rsp.ene_info.cause_value]);
      }
      else
      {
         printf("\tcause not returned\n");
      }
      exit(0);
   case   C_EN_REP:
      printf("\teven: %s\n", \
          names[rsp.evr_buf.erep_common.event_name]);
```

Figure 4-9. Sample Code — Testing the Event Notification Capabilities — ***Continued***

```
            break;
         default:
            printf("\tUnexpected capability \
                           causes abort!\n");
            exit(4);
         }
      }

      en.en_common.asai_common.capability = C_EN_CAN;

      printf("Sent: C_EN_CAN (C_REQUEST) on clid 0\n");
      if (asai_send(fd, &en, sizeof(asai_common_t))<0)
      {
         asai_errval("asai_send");
         exit(-asai_errno);
      }
      if (asai_rcv(fd, &rsp, sizeof(rsp))<0)
      {
         asai_errval("asai_rcv");
         exit(-asai_errno);
      }
      printf("Received: %s (%s) on clid %d",
         caps[rsp.asai_common.capability],
         typs[rsp.asai_common.primitive_type],
         rsp.asai_common.sao_id);
      if (rsp.asai_common.capability==C_ABORT)
      {
         printf("\tabort type:%s cause:%s",
            rsp.abort_info.abort_type == C_PROV_ABT
               ? "C_PROV_ABT" : "C_USER_ABT",
            causes[rsp.abort_info.cause_value]);
      }
      else if (rsp.asai_common.capability==C_EN_CAN_CONF &&
          rsp.asai_common.primitive_type==C_NEG_ACK)
      {
         printf("\tcause:%s", \
                      causes[rsp.enc_rsp.enc_nak.cause_value]);
      }
      printf("\n");
      exit(0);
   }
```

Figure 4-9. Sample Code — Testing the Event Notification Capabilities — *Continued*

ASAI Capabilities

5

This chapter contains all the ASAI application service elements and provides the details of all the capabilities that comprise each ASE. These application service elements are:

- Event Notification and Event Reports
- Third Party Call Control Capabilities
- Set Value
- Value Query
- Request Feature
- Adjunct Routing
- Maintenance
- Abort Capabilities

Event Notification

The Event Notification capability group provides capabilities for a client to obtain information related to specific events that are monitored by the server. For example, an application can monitor the progress of calls arriving at an ACD split. Based on the event information received, the client can take various actions by invoking other ASEs, such as *Third Party Take Control*.The Event Notification ASE provides capabilities for the following:

- Initiating event notification
  - **Event Notification Request**
- Terminating event notification
  - **Event Notification Cancel**
  - **Event Notification End**
  - **Event Notification Stop Call Notification**
- Event Reports

In addition, the two *Abort* capabilities can be considered part of the Event Notification ASE; *Abort* capabilities are described in ''Abort Capabilities'' on page 5-51. The server only accepts requests for event reports on domains that are directly associated with it. The following domains are supported:

- ACD split
- VDN
- TAC

Event Notification Request

Description

The *Event Notification Request* capability allows the client to request the server to begin reporting certain events. After the server acknowledges the request, the client receives event reports each time an event occurs. The client does not have to be a party to any of these events. A client can have several instances of Event Notification active at the same time and a single event can be reported for more than one ASE instance. In such cases, the client receives separate event reports.

The client invokes this capability by issuing a `C_EN_REQ` to which the server responds with a `C_EN_CONF`. If the acknowledgement is positive, the client starts to receive event reports and continues to receive them until one of the following terminates the association:

- The client sends **Event Notification Cancel**
- The server sends **Event Notification End**
- The client receives an **Abort capability**

Request/Indication Parameters

Domain — The domain over which notifications are requested. Domain consists of:

- **Domain Type** — ACD split, VDN, or TAC
- **Domain Value** — The specific domain, within domain type, that is to be the subject of reports

Notes

The client can request notification only for domains directly associated with the server. Only one domain can be included per request.

Event Notification End

Description

The server sends the *Event Notification End* capability if it can no longer provide the requested service. The client receives no further event reports on the specified association identifier. The ASE is terminated.

The server invokes this capability by sending a `C_EN_END`.

Notes

Notification must have been initiated through the *Event Notification Request* capability.

Event Notification Cancel

Description

The *Event Notification Cancel* capability allows the client to request cancellation of event reports for a specified domain.

The client invokes this capability by sending a `C_EN_CAN`, to which the server responds with a `C_EN_CAN_CONF`. If the confirmation is positive, the ASE instance is terminated. If the confirmation is negative, the client continues to receive event reports and can reissue the *Event Notification Cancel* capability.

Notes

Event notification must have been initiated by an *Event Notification Request*.

Event Notification Stop Call Notification

Description

The *Event Notification Stop Call Notification* capability allows the client to request the server to stop sending event reports for a specified call. The server continues to send event reports for other calls with the specified association identifier.

The client invokes this capability by sending a `C_EN_SCN` to which the server responds with a `C_EN_SCN_CONF`. If the server is able to terminate event reporting for the specified call, the client no longer receives the associated event reports. If the server cannot terminate event reporting for the call, or if the call does not exist, the server issues a negative acknowledgement, and the client continues to receive any event reports associated with the call (if it exists).

Request/Indication Parameter

- **Call Identifier** — The call for which no further event reports are desired.

Notes

Only one call identifier can be specified in each Stop Call Notification request.

Event Report

Description

The *Event Report* capability is used by the server to report call-related events. An event applies to one of the following:

- A party on a call
- Multiple parties on a call
- A call independent of parties
- A particular ECS entity (ACD agents) independent of any call

The list below shows which events can apply to each of the groups shown.

Party	Multi. Parties	Call	Sw. Entity
Alerting	Call conf'd	Call ended	Logout
Answered	Call transf'd	Call offered	Login
Busy		to domain	
Connected		Queued	
Charge			
Cut-through		Reorder/Denial	
Drop		Call initiated	
Hold		Call redirected	
Reconnected		Call Originated	
Trunk Seized			

Events are related to states, more specifically to state transitions. A state is associated with each call and with each party to that call. A party to several calls can be in different states on each call and different parties to the same call can be in different states. The states are:

- null
- seized
- alerting
- active
- held

Event reports are issued for state transitions; that is when a party or a call for which reports have been requested moves from one state to another.

Table 5-1. Valid Event Item Combinations

Event	Items Reported						
	Calling Party No.	**Called Party No.**	**Connected Party No.**	**Call Id**	**New Call Id**	**Party Id**	**Party_Id List**
Alerting	I	I	I	A		A	
Answered		I	I	A		A	
Busy/Unavailable		I		A		I	
Connected		I	I	A		A	
Cut-Through/Progress				A		A	
Disconnect/Drop			I	A		I	
Hold			I	A		A	
Reconnected			I	A		A	
Trunk Seized		I		A		A	
Call Conferenced	I	I		I	A	I	A
Call Transferred	I	I		I	A	I	A
Call Redirected				A			
Call Ended				A			
Call Offered to Domain	I	I		A			
Call Initiated				A		I	
Call Originated	A	A		A		A	
Queued		I		A			
Reorder/Denial		I		I			
Logout							
Login							

Key:
A: item always provided in event report.
I: item may be provided in event report.

Table 5-1. Valid Event-Item Combinations — *Continued*

Event	Items Reported					
	Calls In Queue	**Cause**	**User Entered Info**	**Lookahead Interflow Info**	**Domain**	**Redirection Number**
Alerting					I	I
Answered		I				
Busy/Unavailable		I				
Connected						I
Cut-Through/Progress						
Disconnect/Drop		I				
Hold						
Reconnected						
Trunk Seized						
Call Conferenced						
Call Transferred						
Call Redirected					I	I
Call Ended		A				
Call Offered to Domain			I	I	I	I
Call Initiated						
Call Originated						
Queued	I					
Reorder/Denial		I				
Logout						
Login						

Key:
A: item always provided in event report.
I: item may be provided in event report.

Table 5-1. Valid Event-Item Combinations — *Continued*

Event	Items Reported							
	Extension	Extension List	Trunk Group Number	Work Flow	Progress Indicator	UUI	oli	ucid
Alerting			I			I	I	I
Answered								
Busy/Unavailable								
Alerting								
Answered								
Busy/Unavailable								
Charge	A		A					
Connected							I	I
Cut-Through/Progress					I			
Disconnect/Drop								
Hold								
Reconnected								
Trunk Seized			I					
Call Conferenced		I						I
Call Transferred		I						I
Call Redirected								
Call Ended								
Call Offered to Domain			I			I	I	I
Call Initiated								I
Queued								
Reorder/Denial								
Logout	A							

Key:
A: item always provided in event report.
I: item may be provided in event report.

Third Party Call Control Capabilities

The Third Party Call Control capability group includes a number of third party subgroups and two types of the abort capabilities. In addition, there are numerous Event Report interactions. Event reports and abort capabilities are described in this chapter.

Third Party Call Control allows a client to establish, control, terminate, and monitor calls to which the client is not a party. When invoking the Third Party capabilities, the client controls a call on behalf of a party to the call. For example, a client can request that a call be set up between Party A and Party B. At a later time, Party C can be conferenced into the call. The server handles the third party conference request as if it had been made by Party A or Party B.

Third Party Call Control that interacts with switch-classified calls is unique. In general, if the ECS receives any RELease COMplete message for a Third Party Call Control association (started by `3P_Make_Call` or `3P_Take_Control`), the ECS continues to process the call normally.

The exception to this occurs when the ECS receives any RELease COMplete message for an ECS call that is in the classification state (has not yet been classified). In this case, on receipt of the RELease COMplete message, the ECS will deny a Relinquish Control request and tear down the call.

Third Party Call Control provides for the following:

- Third Party Answer Call
- Third Party Automatic Dialing
- Third Party Clear Call
- Third Party End Call
- Third Party Domain Control
- Third Party Domain Control End
- Third Party Listen Disconnect
- Third Party Listen Reconnect
- Third Party Make Call
- Third Party Merge
- Third Party Reconnect
- Third Party Redirect Call
- Third Party Relinquish Control
- Third Party Selective Drop
- Third Party Selective Hold
- Third Party Send DTMF Signals

- Third Party Single-Step Conference
- Third Party Take Control

There are certain restrictions common to most third party calls:

- The calling number is not permitted the service. For example, the calling number may not be permitted to make out-of-state calls.
- The called number is not permitted the service. For example, the called number is not permitted to accept out-of-state calls.
- The client is not permitted to request the service.

Third Party Answer Call

Description

The *Third Party Answer Call* capability allows the client to request to “answer” a call on behalf of a station user. The call can be a ringing, bridged, or held call that is present at a station.

Answering a ringing, bridged or held call indicates to connect a call by forcing the station off-hook (if the user is on-hook) or cutting through the call to the head or handset (if the user is off-hook). The effect is as if the station user selected the call appearance of the alerting, bridged, or held call and then went off-hook.

The client invokes this capability by issuing a `C_3PANS` to which the server responds with a `C_3PANS_CONF` (*Third Party Answer Call* confirmation). After the acknowledgement, the client continues to receive event reports for the answered call.

The *Third Party Answer Call* request is acknowledged by the server if it is able to connect the specified call by either forcing the station off-hook (turning the speakerphone on) or waiting up to five seconds for the user to become off-hook.

The server sends a `C_NEG_ACK` if the parameters of the request are invalid or if it cannot attempt the answer operation.

Request/Confirmation Parameters

- **Call Identifier** — Indicates the alerting, bridged, or held call to be connected at the controlled station.

Notes

The Third Party Answer capability can be used to answer a call present at any station type (that is, analog, DCP, hybrid, and BRI), as long as the station is domain controlled. A call which is already connected when the Third Party Answer request is made will result in a positive acknowledgement.

Interactions with Event Reports

This capability can be issued after the domain-controlled station is in the alerting or held state on the call for which this capability is requested. After this capability is successfully acknowledged, the station party moves to the active state.

Third Party Auto Dial

Description

The *Third Party Auto Dial* capability allows a client to request the server to establish a call on behalf of the controlled extension specified in a *Third Party Domain Control* (extension) request. The effect is that the extension places a call to the specified called number automatically when that extension is off-hook. The capability also allows the server to confirm the request.

The client issues a `C_3PAD` to request that the call be made and monitored. The server responds with a `C_3PAD_CONF` if the client has requested acknowledgement. The *Third Party Auto Dial* confirmation indicates only that the request has been received and understood and that the server will try to establish the requested call. If the server cannot establish the call, it issues a call ended event report to the client. The server issues event reports only after a positive acknowledgement.

A negative acknowledgement is issued only if the client is not subscribed to the service or if the server is unable to accept the request.

Request/Confirmation Parameters

- **Destination Address** — The called endpoint.
- **Return Acknowledgement** — Indicating that the server must return an acknowledgement to the client.

In addition, the following parameters are optional:

- **Facility Type** — Used only if the Destination Address is a trunk group.
- **Facility Access Code** — Required only if Facility is specified.
- **Priority Calling** — Provides a special ring and places call ahead of all other calls for that party.
- **User to-User-Data** — A substructure containing user data. The structure contains the following fields:
  - `Leng` field — An integer value that indicates the number of octets of user data included in the request. This field is set to 0 if no user-user data is present. Currently, the ECS will accept up to 32 bytes of data (`leng` = 32). If more than 32 bytes is specified, all data is discarded.
  - `Protocol` field — Indicates the type of information and is restricted to `C_UU_USER` which indicates a user-specific protocol or `C_UU_IA5` which indicates IA5 or ASCII characters.
  - `Info` field — A pointer to an ASCII string of no more that 32 characters in length.

Notes

There are no restrictions on this capability.

Related Event Reports

This capability does not affect the state(s) of the parties to the call.

Third Party Call Ended

Description

The *Third Party Call Ended* capability is issued by the server to inform the client that a client-controlled call has been disconnected and the association has been cleared. The client can no longer control the call or use its association identifier.

The server issues a `C_3PCE` when the endpoints to a call have been disconnected or when some internal condition causes it to terminate processing for the call.

Request/Confirmation Parameters

Call Identifier — The call that has been terminated.

Notes

This capability may indicate that an existing call has been merged with another existing call. It may also be used if a call initiated by a `C_3PMC_REQ` cannot be routed by the server, thus terminating call processing.

Interactions with Event Reports

Reports that a call, identified by its call identifier, has ended and that the last party to the call was dropped or disconnected. A *drop* event for the last party added to the call is not issued in these instances; thus, this capability replaces the last *drop* event on a call.

Third Party Clear Call

Description

The *Third Party Clear Call* capability allows a client to request that an existing call be disconnected and related resources be freed. All parties to the call are disconnected.

The client invokes this capability by issuing a `C_3PCC`. The server responds with a `C_3PCC_CONF` as soon as the call has been completely cleared. Until the call is completely cleared and all resources freed, the server continues to issue event reports about the call.

Notes

Control of the call to be cleared must have been initiated by a *Third Party Make* Call or a *Third Party Take Control* capability.

The client can issue this capability at any time, regardless of pending or outstanding acknowledgements for previously requested capabilities.

Interactions with Event Reports

After this capability is successfully acknowledged, all parties on the call return to the *null* state.

Third Party Domain Control

Description

The *Third Party Domain Control* capability allows a client to request the server to begin reporting events for calls that originate and terminate at the specified domain. The client can also request third party call control capabilities for those calls. After the server accepts the request, the client receives an *Event Report* each time an event occurs. Clients can have several simultaneously active instances of *Third Party Domain Control* and a single event can be reported to more than one association. When this happens, the client receives separate event reports.

The client invokes this capability by issuing a `C_3PDC_REQ` to which the server responds with event reports. The server continues to issue event reports until one of the following occurs:

- The client invokes the **Third Party Relinquish Control** capability
- The server invokes the **Third Party Domain Control Ended** capability
- The server receives a **P_ABORT** capability
- The server receives or invokes a **U_ABORT** capability

Request/Confirmation Parameter

Domain — The domain over which event reports are to be provided and third party call control capabilities can be requested. The domain consists of:

- **Domain Value** (station or ACD split/skill)

Notes

The client can request control only for domains directly associated with the server.

Interactions with Event Reports

As soon as the server accepts this request, it begins to issue event reports to the client.

Third Party Domain Control End

Description

The server issues the *Third Party Domain Control Ended* capability when it can no longer provide event reports or allow third party capabilities for the specified domain.

The server issues a `C_3PDCE`, after which the client no longer receives event reports for the domain. The server will not accept additional third party capabilities and the association is terminated.

Notes

Control must have been initiated with a *Third Party Domain Control* request.

Interactions with Event Reports

This capability does not affect the state(s) of the parties to the call.

Third Party Listen-Disconnect

Description

Third Party Listen-Disconnect capability allows the client to request the server to temporarily disconnect a specified party (the listener) from listening to communication from another party or parties [the talker(s)] on an active call. The client continues to receive feedback regarding the call and may continue to control the call with further Third Party capabilities.

The server sends a `C_3PSL_DISC_ACK` with primitive type `C_POS_ACK` when it has disconnected the listener party from listening.

The server sends a `C_3PSL_DISC_ACK` with primitive type `C_NEG_ACK` if the parameters of the request are invalid or if it cannot disconnect the listener party from listening.

Request/Confirmation Parameters

Depending on the conditions, the request must specify the following parameters:

- **Listener Party Identifier** — The party on the call to be disconnected from listening.

The request may also specify:

- **Talker Party Identifier** — This parameter is specified to identify a specific party who will no longer be heard by the listener. If this parameter is omitted, the listener will be disconnected from listening to *all* other parties (that is, the listener will not hear any other party on the call).

Notes

The listener party must be connected on the call.

Although any voice path originating from a station or trunk on a call is blocked for the party which is disconnected from listening, it is possible that a disconnected listener will still hear faint tones or speech. This can happen in a multiparty call where the primary voice path is disconnected but alternate paths remain (for example, through a third party on the call whose voice path to listener is not disconnected).

Interactions with Event Reports

This capability does not affect the state of any party on a call.

Third Party Listen-Reconnect

Description

The *Third Party Listen-Reconnect* capability allows the client to request the server to reconnect a specified party (the listener) who was previously blocked from listening to one or more parties on a call via the Third Party Listen-Disconnect capability.

The server sends a `C_3PSL_RECONN_ACK` in response to this request. Primitive type `C_POS_ACK` indicates success; `C_NEG_ACK` is used if the parameters of the request are invalid or if it cannot reconnect to listen.

Request/Confirmation Parameters

Depending on the conditions, the request must specify the following parameters:

- **Listener Party Identifier** — The party on the call to be listen-reconnected to listening.

The request may also specify:

- **Talker Party Identifier** — This parameter is specified to identify a specific party who will be listen-reconnected. If this parameter is omitted, the listener will be listen-reconnected to *all* other parties.

Notes

The listener party must be connected on the call and should have been previously listen disconnected.

Interactions with Event Reports

The capability does not affect the state of any party on a call.

Third Party Make Call

Description

The *Third Party Make Call* capability allows a client to request the server to establish a call on behalf of two other parties, one of which must be designated as the calling party and the other as the called party. The capability also allows the server to confirm the request.

The client invokes this capability by issuing a `C_3PMC_REQ` to which the server responds with a `C_3PMC_CONF` (*Third Party Make Call* confirmation) if the client has requested acknowledgement. The *Third Party Make Call* request initiates an Event Report (`C_EN_REP`).

The client retains control of the call until one of the following conditions is met:

- The client issues a **Third Party Clear Call** request for the call.
- The client issues a **Third Party Relinquish Control** request for the call.
- The client receives a **Third Party Call Ended** capability for the call.
- The client invokes or receives a **U_ABORT** capability for the call.
- The client receives a **P_ABORT** capability for the association.

Note that this is not an instantaneous process; an acknowledgement from the server means only that the server has received and understood the request. The server then must attempt to fulfill the request, perhaps try several alternative routes. If the first route fails, the server may not be able to attempt a second route and call processing terminates. In this case, if the `ack_flag` is set on, the client receives an acknowledgement from the ECS. If the `ack_flag` is not set on, the client receives no response, regardless of whether the call can be made or not (see ‘‘C_3PMC_CONF (3ASAI)’’ on page 9-49 in Chapter 9, ‘‘Programming Manual Pages’’).

Request/Confirmation Parameters

- **Calling Number** — The party, specified by extension number, on whose behalf the call is being made
- **Called Number** — The extension number of the third party who is being called
- **Acknowledgement Requested** — A flag that must be set on or off, indicating whether the server must return an acknowledgement as soon as it accepts the `C_3PMC` request

In addition, the programmer can specify any or all of the following:

- **Type of Facility** — Trunk group, ACD split or neither (default = neither)

- **Facility Access Code** — Access code of the trunk or ACD split, if one is specified
- **Priority Call** — A flag indicating that the call is a priority call (default = no priority)
- **Waiting Time** — The number of rings (after the call has reached the alerting state) to wait for the call to be answered (default is ECS-dependent)
- **Service Circuit** — A flag indicating that a call classifier is to be added to the call (default = no call classifier)
- **Direct Agent Call** — A flag indicating that the call is directed to a specific ACD agent (default = no agent)
- **Supervisory Assist** — A flag indicating that the call has been directed by an ACD agent who needs assistance to an ACD supervisor (default = no assist)
- **Alternate Destination** — A flag indicating that the called number is to be alerted first, before the calling number (default = calling number first)
- **User-to-User Data** — A substructure containing user data. The structure contains the following fields:
  - `Leng` field — An integer value that indicates the number of octets of user data included in the request. This field is set to 0 if no user_user data is present. Currently, the ECS will accept up to 32 bytes of data (`leng` = 32). If more than 32 bytes is specified, all data is discarded.
  - `Protocol` field — Indicates the type of information and is restricted to `C_UU_USER` which indicates a user-specific protocol or `C_UU_IA5` which indicates IA5 or ASCII characters.
  - `Info` field — A pointer to an ASCII string of no more that 32 characters in length.
- **ans_mach_treat** —If `ans_mach_treat` is `CO_AM_DISC`, the call is disconnected upon answering machine detection. If the treatment is `C_CO_AM_SWITCH`, it follows the setting in the ECS administration. If the treatment is `C_CO_AM_CONNECT`, the call is connected.This parameter must be coded -1 if unused.

Notes

If the Service Circuit parameter is used to add a call classifier to the call and a call classifier is not available, the server rejects the *Third Party Make Call* request, terminates the call and sends the *Third Party Call Ended* capability to inform the client.

If the Direct Agent parameter is selected, the Type of Facility must be set to ACD and the Facility Access Code must be supplied.

Interactions with Event Reports

Without the Alert Originator First option, this capability places the originating party in the *active* state.

Third Party Merge

Description

The *Third Party Merge* capability allows a client to request the server to merge two existing, client-controlled calls that have a common party. The capability also allows the server to confirm the request. The effect of the merge is similar to a conference or transfer operation, depending on the value of the conference flag. If the conference flag is on, the common party remains connected to the call along with all the other parties (as in conferencing). If the conference flag is off, the common party is dropped from the merged call (as in transferring).

The client invokes this capability by issuing a `C_3PM` request to which the server responds with a `C_3PM_CONF`. The server sends positive acknowledgement when it has validated the parameters and merged the calls. After the acknowledgement, the client receives a *Third Party Call Ended* (`C_3PCE`) and continues to receive event reports for the merged call. The client can control the merged call using the association identifier that remains active.

Negative acknowledgement means that the request parameters were invalid or that the server cannot merge the two calls. In this case, the client can continue to control both calls.

Request/Confirmation Parameters

- **Call Identifier 1** — One of the calls to be merged.
- **Call Identifier 2** — The other call. This identifier is inferred from the association identifier for the call.
- **Common Party Identifier** — The party common to both calls.
- **Conference Flag** — The flag that indicates whether the common party is to be retained on the merged call or disconnected.
- **Party List** — Pointer to a structure that indicates the ECS-assigned party identifiers after two calls are merged.
- **Old Party Identifier** — Pointer to a structure that indicates the old party identifiers before the call is merged.

Notes

The ECS chooses which call is to be *Call Identifier 1* and which is to be *Call Identifier 2*. Consult your DEFINITY ECS system administrator for specific information.

Interactions with Event Reports

This capability can be issued after the common party is in the *held* state on the call for which the capability is requested and in the *active* state on the second call. After this capability is successfully acknowledged, the common party moves to the *active* state or to the *null* state, depending on the conference flag. All other parties to the call remain in the state they were in before the merge request.

Third Party Reconnect

Description

The *Third Party Reconnect* capability allows the client to request the server to reconnect a specified party.

The client invokes this capability by issuing a `C_3PR` to which the server responds with a `C_3PR_CONF`. The client must use feedback to determine the state of the call with respect to each endpoint. If the server is unable to attempt the reconnection or if the parameters specified by the client are invalid, the association identifier remains valid for further control of the call and the party that was to have been reconnected remains in the state it was in, prior to the request.

Request/Confirmation Parameters

The following parameters are mandatory under the conditions noted; otherwise, they are optional:

- **Party Identifier** — The party to be reconnected. This parameter is mandatory if the association is started with a *Third Party Make Call* or *Third Party Take Control* capability. It can be inferred (and thus is optional) if the association is started with a *Third Party Domain* Control capability.
- **Call Identifier** — The call that is to be reconnected. This parameter is mandatory if the association is started with a *Third Party Domain Control* capability; if the association is started with a *Third Party Make Call* or *Third Party Take Control* capability, it can be inferred.

Notes

The party to be reconnected must be in the *held* state. Some servers may have to be directly connected to the party to be reconnected.

Interactions with Event Reports

This capability can be requested for a directly connected party when the party is in the *held* state. After this capability is successfully acknowledged, the party goes from the *alerting* state to the *active* state.

Third Party Redirect Call

Description

The *Third Party Redirect Call* capability allows a client to request that an alerting call be redirected to another number.

The client invokes this capability by issuing a `C_3PREDIR` request to which the server will respond with an acknowledgment. The server sends positive acknowledgment when it has validated the parameters.

The server sends a `C_NEG_ACK` if the parameters of the request are invalid or if it cannot apply the tones to the call.

Request/Confirmation Parameters

Depending on conditions, the request must specify the following parameters:

- **ASAI Common** — The capability should be set to `C_PREDIR` and `primitive_type` to `C_REQUEST`. The `sao_id` (formerly known as `cluster_id)` should also be set to identify the association that controls the alerting call.
- **Call Identifier** — This parameter is required for domain control association, and is ignored with call control associations. Two fields, `id_length` and `id_ptr` are in a call id. If the `id_length` is zero, the field is omitted. If the `id_length` is not zero the `id_ptr` contains a binary representation for the call id.
- **Redirect Number —** This required parameter identifies the destination of the redirected call. Two fields, `plan_type` and `s`, are in a number id. The field `s` is a null-terminated string containing the destination number. The field plan_type is described in ‘‘Identifiers’’ on page 8-5 in Chapter 8, ‘‘ASAI Capability Primitives’’.
- **Party Identifier** — This is an optional parameter. It specifies the `redirected-from` party on the call. The length of the party identifier is `id_length`. The character array pointed to by `id_ptr` contains the party_id. If omitted, the last added party is used.

Notes

For a call to be redirected, it must be alerting at a station.

Interactions with Event Reports

A redirected call’s event reports will be the same as to call coverage.

Third Party Relinquish Control

Description

The *Third Party Relinquish Control* capability allows a client to yield control of an existing call. All parties remain connected to the call. Once acknowledgement is received from the server, no further feedback is provided and the client cannot exert any further control over the call or domain within that association.

The client invokes this capability by issuing a `C_3PRC_REQ` to which the server responds with a `C_3PRC_CONF` (*Third Party Relinquish Control* confirmation). Once positive acknowledgement has been received by the client, the association identifier used by the `C_3PTC_REQ` is terminated.

If the server sends a negative acknowledgement, the association identifier used by the `C_3PTC_REQ` is still active.

Notes

A client can issue this capability at any time, regardless of any pending or outstanding acknowledgements for any previously requested capability.

Interactions with Event Reports

This capability does not affect the state(s) of the parties to the call.

Third Party Selective Drop

Description

The *Third Party Selective Drop* capability allows a client to request the server to disconnect a specified party. The client continues to receive feedback regarding the call and may continue to control the call with further Third Party capabilities.

The server sends a `C_3PSD_POS_ACK` when it has disconnected the party. The client must use feedback to determine the state of the call with respect to each endpoint.

The server sends a `C_3PSD_NEG_ACK` if the parameters of the request are invalid or if it cannot disconnect the party.

Request/Confirmation Parameters

Depending on conditions, the request must specify the following parameters:

- **Party Identifier** — The party on the call to be disconnected. This parameter must be specified if the association is started with a *Third Party Make Call* capability; it can be inferred if the association is started with a *Third Party Domain Control* capability.
- **Call Identifier** — The call on which this capability operates. This parameter must be specified if the capability is requested over an association started with a *Third Party Domain Control* capability; otherwise, it must not be used.
- **Use- to-User Data** — A substructure containing user data. The structure contains the following fields:
  - `Leng` field — An integer value that indicates the number of octets of user data included in the request. This field is set to 0 if no user-user data is present. Currently, the ECS will accept up to 32 bytes of data (`leng` = 32). If more than 32 bytes is specified, all data is discarded.
  - `Protocol` field — Indicates the type of information and is restricted to `C_UU_USER` which indicates a user-specific protocol or `C_UU_IA5` which indicates IA5 or ASCII characters.
  - `Info` field — A pointer to an ASCII string of no more that 32 characters in length.

Notes

There are no restrictions on this capability.

Interactions with Event Reports

This capability can be requested for a party when the party is in any state except *null*. After the capability is successfully acknowledged, the party goes to the *null* state. This capability is the only one allowed for a party that is in the *seized* state.

Third Party Selective Hold

Description

The *Third Party Selective Hold* capability allows the client to request the server to invoke this service on behalf of a specified party. The effect is the same as if the identified party directly invoked the hold service. The client continues to receive feedback regarding the call and may continue to control the call with further Third Party capabilities.

The server sends a `C_3PSH_POS_ACK` when it has placed the party on hold. The client must use feedback to determine the state of the call with respect to each endpoint.

The server sends a `C_3PSH_NEG_ACK` if the parameters of the request are invalid or if it cannot attempt the hold operation.

Request/Confirmation Parameters

Depending on conditions, the request must specify the following parameters:

- **Party Identifier** — The party on the call to be placed on hold. This parameter must be specified if the association is started with a *Third Party Make Call* or a *Third Party Take Control* capability; it can be inferred if the association is started with a *Third Party Domain Control* capability.
- **Call Identifier** — The call on which this capability operates. This parameter must be specified if the capability is requested over an association started with a *Third Party Domain Control* capability; otherwise, it must not be used.

Notes

Some third party servers may have to be directly connected to the specified party. The party must be in the active state on the call.

Interactions with Event Reports

This capability can be requested for a directly connected party when that party is in the *active* state. After the capability is successfully acknowledged, the party goes from the *active* state to the *held* state.

Third Party Send DTMF Signals

Description

The *Third Party Send DTMF Signals* capability allows a client to request the server to issue a sequence of DTMF tones on behalf of a party on the call. The DTMF sequence to be generated may contain any digit as well as the # and * characters. A maximum string of 32 characters may be sent from the client to the server in a single service request message.

The client invokes this capability by issuing a `C_3PSDS` request to which the server will respond with a `C_3PSDS_CONF.` The server sends positive acknowledgment when it has validated the parameters.

The server sends a `C_NEG_ACK` if the parameters of the request are invalid or if it cannot apply the tones to the call.

Request/Confirmation Parameters

Depending on conditions, the request must specify the following parameters:

- **Call Identifier** — The call on which this capability operates. This parameter must be specified if the capability is requested over an association started with a *Third Party Domain Control* capability; otherwise, it must not be used.
- **Party Identifier** — The party on the call for whose benefit the tones will be sent. This parameter must be specified if the association is started with a *Third Party Make Call* capability; it can be inferred if the association is started with a *Third Party Domain Control* capability.
- **User Data** — A substructure containing the user digit data. The structure contains the following usable fields:
  - `Leng` field — An integer value that indicates the number of octets (digits) of user data included in the request. Currently, the ECS will accept up to 32 bytes of data (`leng` = 32).
  - `type` field — Indicates the type of information and is restricted to `C_UU_IA5`, which indicates IA5 or ASCII characters.
  - `digits` field — A pointer to an ASCII string of no more that 32 characters in length.

Third Party Single-Step Conference

Description

The *Third Party Single-Step Conference* capability allows the client to add an IDLE station into an existing call with a single ASAI operation. It is not necessary to place the call on hold or to use the Third Party Merge capability. The party being added must either have a speaker phone, or go off-hook on an idle call appearance within five seconds, or the request will fail. The positive response to the request provides a list of all the parties existing on the call.

Request/Confirmation Parameters

- Alert Destination — Indicates if the station being added should be alerted for the conference. See the notes below.
- Visibility — Indicates whether or not the talk path of the station being added should be connected to the conference call. In the future, visibility will also effect the display.
- Number of Conferenced Extensions — Indicates the number of parties on the call.
- Party list — Contains information about the parties on the call including the party identifier, extension number, and the numbering plan information for each party on the call.
- Universal Call Identifier — Indicates the unique Universal Call ID assigned by the ECS for the call if this feature is available in the ECS.

Notes

This capability may be specified on a Call Control association where the extension to be added to the existing call is specified by the Station Extension parameter. When used on a Domain Control association, the Single-Step request must be provided on the Domain Control association and the Call Identifier of the existing call is specified in this request.

Single-Step Conference also allows control of the voice path of the call being added via the visibility parameter. It allows the station to be added in a listen-only connection or with both talk and listen paths connected.

In the initial release of Single-Step Conference, the ECS will always treat the Alert Destination parameter as being set to **OFF**, regardless of how the CallVisor PC application sets it. See the DEFINITY ECS Release 6 CallVisor ASAI Technical Reference, 555-230-220, for complete details of this capability.

Third Party Take Control

Description

The *Third Party Take Control* capability allows the client to request the server to allow it to control a call in progress to which the client is not a party. Once the request for *Third Party Take Control* is acknowledged, the client can use other third party capabilities (for example, *Third Party Call Merge*) to control the call.

The client invokes this capability by issuing `C_3PTC_REQ`, to which the server responds with a `C_3PTC_CONF`. The client retains control of the call until one of the following conditions is met:

- The client invokes the **Third Party Clear Call** capability for the call.
- The client invokes the **Third Party Relinquish Control** capability for the call.
- The client invokes or receives a **U_ABORT** capability for the call.
- The client receives a **P_ABORT** capability for the association.

Request/Confirmation Parameters

- **Call Identifier** — The identifier of the call that the client wants to control.

In addition, the programmer can specify any or all of the following:

- **Party Identifier** — The identifier for a party to the call. This parameter is mandatory if the extension address (below) is specified.
- **Address** — The extension address (number) of each party to the call.

Note that the party identifier and the address can be repeated.

Notes

The client must have received an event report (from a different association) that included the call identifier. The client can request an event report, receive it and then request the *Third Party Take Control* capability.

Interactions with Event Reports

This capability does not affect the state(s) of the parties to the call.

Notes

There are no restrictions on this capability.

Set Value Capabilities

The Set Value capability group consists of the *Set Value* capability and the *Abort* capability which is described later in this chapter.

Set Value allows a client to request a server to set the Message Waiting Indicator (MWI) or to activate the Flexible Billing Feature. Both the feature to be set and its values are specified as parameters to the capability.

Description

Set Value allows a client to request that an item controlled by the server (for example, `C_SV_MWI, C_SV_FLEX,SAC,` and `SCFS`) be set to a specified value. It is a user-initiated, acknowledged, end-to-end service. The client invokes this capability by issuing a `C_SV_REQ` with the items to be set, additional parameters when required and the desired value.

The server responds by issuing a `C_SV_CONF`. If the request is not honored, the cause parameter is set.

Request/Indication Parameters

- **Item** — Message Waiting Indicator. To set the MWI, these elements must be specified:
  - The station whose MWI is to be set
  - The desired state (Off or On)
- **Item** — Flexible Billing Feature. To set the Flexible Rate, these elements must be specified:
  - The call (call_id) that this capability is to operate on
  - The type of billing change (for example, `NEW_RATE,FLAT_RATE`, etc.)
  - The rate to apply to the call

Value Query Capabilities

The Value Query capability group consists of:

- **Value Query**
- **Value Query Confirm**
- **Value Query Response**

The Abort capability is described later in this chapter.

Value Query allows a client to request and receive information about the status or value of features or services under server control. The following features or services can be queried:

- ACD split status
- Call classifier status
- Trunk group status
- Time of day
- ACD agent status
- Station status
- ACD Agent Login Audit
- Party ID
- UCID
- Extension
- Calls query
- Station Message Waiting Lamp Query
- Station Send-All-Calls Status Query
- Station Call Forwarding Status Query
- Name Query (Integrated Directory Data)
- UCID Query (Universal Call ID)

The feature or service is specified as a parameter. The server can send multiple replies to a single query.

Value Query

Description

The *Value Query* capability allows a client to request and receive information about the status of the value of a feature or service under server control.

The client issues a `C_VQ_REQ`, specifying the feature or service queried. The server responds with a `C_VQ_CONF` when only one reply is needed or a variable number of `C_VQ_RESP` messages when multiple replies are needed.

Request Parameters

Item — The object for which a value is being requested, including:

- **ACD Split Status** — Requests the number of agents logged into the ACD split; the number of agents available to receive calls; or the number of calls queued in the ACD split.
- **Call Classifier Status** — Requests the number of idle call classifiers; the number of call classifiers that are in use; the number of call classifiers that are "busied-out" for maintenance; or the number of call classifiers that are unavailable for reasons other than maintenance.
- **Trunk Group Status** — Requests the number of idle trunks in the trunk group; the number of trunks in use in the trunk group; the number of trunks that are "busied-out" for maintenance; or the number of trunks that are unavailable for reasons other than maintenance.
- **Time of Day** — Requests the time of day, specified in the following fields
  - Year: specified as 2 decimal digits (00-99)
  - Month: specified as 2 decimal digits (01-12)
  - Day: specified as 2 decimal digits (01-31)
  - Hour: specified as 2 decimal digits (00-23)
  - Minute: specified as 2 decimal digits (00-59)
  - Second: specified as 2 decimal digits (00-59)
- **ACD Agent Status** — Requests "talk state" of agent with respect to ACD split (on call or idle); or current work mode of agent (manual-in, auto-in, after call work, auxiliary work)
- **Station Status** — Requests talk state of extension (on call or idle)
- **ACD Agent Login Audit** — Requests a list of agents currently logged into a split/skill
- **Party Identifier** — Requests list of party identifiers for the call; or the extension number of each party to the call

- **Extension** — Requests the kind of extension number (VDN, ACD split, announcement, voice station, ASAI, other)
- **Calls Query** — Requests list of call identifiers for all calls present at the station extension; list of party identifiers and identification of station extension on the call; or list of endpoint call states (alerting, call initiated, connected, held, unknown) for each of the calls present at the station extension
- **Station Message Waiting Lamp Query** — Requests status of the Station Message Waiting Lamp for the given extension
- **Station Send-All-Calls Status Query** — Requests status of the Station Send-All-Calls for the given extension
- **Station Call Forwarding Status Query** — Requests status of the Station Call Forwarding for the given extension
- **Name Query** — Requests the name associated with the given extension from the Integrated Directory Database
- **UCID Query** — Requests the Universal Call Id associated with the call

Notes

There are no restrictions on this capability.

- The ECS software, Release 5, now supports up to 27 special characters in Integrated Directory Database. Before Release 5 only 15 characters were supported and only 15 characters will be returned, if the ECS software is G3V4, or if it is Release 5 or 6, and the ASAI link version is 1 or 2. Correct ASAI link version has to be negotiated for this functionality. For Release 5 and 6, the link version should be set to 3.

Value Query Response

Description

Value Query Response allows the server to provide multiple responses to a request.

The server issues a `C_VQ_RESP` to signal delivery of information requested by the client with a `C_VQ_REQ`. The server terminates this ASE with a `C_VQ_CONF` after all *Value Query Response* requests have been answered.

Request Parameter

Value — Depends on the item(s) specified in the Value Query request defined previously in this chapter.

Notes

There are no restrictions on this capability.

Request Feature Capabilities

The Request Feature capability group consists of the *Request Feature* capability and the two Abort capabilities which are common to all groups. The Abort capabilities are described later in this chapter.

Request Feature allows the client to request one of the following features:

- Agent login
- Agent logout
- Change Agent Work Mode
- Call Forwarding
- Send All Calls

Description

The *Request Feature* capability allows the client to request the ECS-provided feature. Both the feature and the operation requested (invoke or cancel) must be specified.

The client invokes this capability by issuing a `C_RF_REQ` to invoke or cancel an ECS-provided feature. The ECS responds with a `C_RF_CONF` which terminates the ASE. A negative acknowledgement should indicate the cause of the request failure.

Request/Indication Parameters

Feature Identifier — The following features can be invoked with the indicated parameters:

- Activate/Cancel Call Forwarding
  - — Forwarding Number
  - — Forwarded to Number
  - — Forwarding Type (Busy/no answer; No answer; Unconditional)
- ACD Agent Log In/Log Out/Change Work Mode
  - — Split
  - — Work Flow
  - — Agent Identifier
  - — Mode
  - — Password
- Activate/Cancel Send All Calls
  - — Extension

Notes

If a request to activate an *already active* feature is received or a request to cancel an *already inactive* feature, the server sends a positive acknowledgement.

Adjunct Routing Capabilities

The Adjunct Routing capability group consists of the following capabilities:

- **Routing**
- **Route Select**
- **Route End**

The Abort capability, described later in this chapter, is also part of this capability group.

This ASE does not provide any timing mechanism for the receipt of acknowledgements. The client can use its own timers to determine the length of time it will wait for a response from the server. The client or the server can terminate a routing capability, regardless of outstanding acknowledgements by issuing a U_ABORT.

Routing

Description

The *Routing* capability allows the client to request routing instructions based on the incoming call resources used by the call (such as originating address or destination address).

The client invokes this capability by issuing a C_RT_REQ to which the server responds with a C_RT_SEL capability that provides the route to use. The route can be a local or remote endpoint directory number (such as an ACD agent) to which the call should be routed. The server can also respond with a C_NEG_ACK using C_RT_SEL capability to deny the client's request. Figure 5-1 shows the actions for a client with an incoming call.

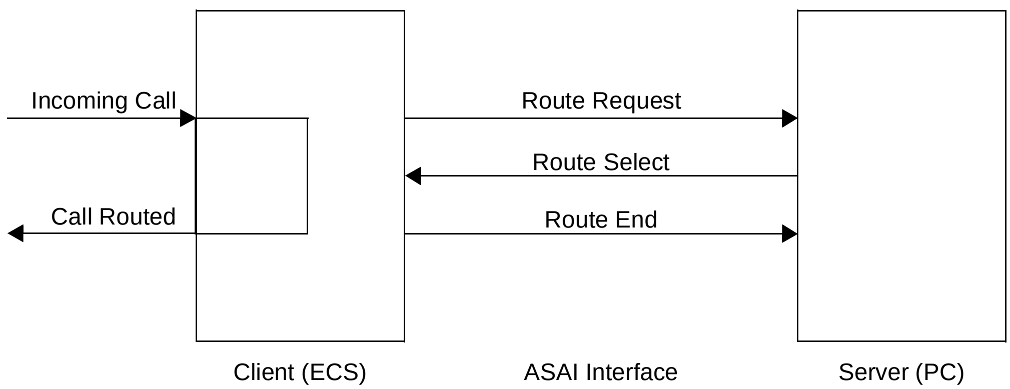


Figure 5-1. Routing an Incoming Call

The sequence of actions is as follows:

1. Incoming call to client
2. Route request issued by client
3. Route Select issued by server
4. Call routed by client
5. Route End issued by client

If the server rejects the request, the cause is indicated with the negative acknowledgement.

Request/Indication Parameters

- **Destination Address** — The called endpoint for which the route is needed

The following parameters are optional:

- **Originating Address** — The endpoint on whose behalf the call is routed
- **Call Setup** — Details of incoming call (look-ahead interflow and user-entered code)
- **Call Identifier** — Identifier of the call to be routed
- **Originating Line Information** `(oli)`— Identifies information indicator (II) digits received in the `oli` information element for the call.
- **User to User Data** — A substructure containing user data. The structure contains the following fields:
  - — `leng` field — An integer value that indicates the number of octets of user data included in the request. This field is set to 0 if no user_user data is present. Currently, the ECS will accept up to 32 bytes of data (`leng` = 32). If more than 32 bytes are specified, all the data is discarded.
  - — `protocol` field — Indicates the type of information and is restricted to `C_UU_USER` which indicates a user-specific protocol or `C_UU_IA5` which indicates IA5 or ASCII characters.
  - — `info` field — A pointer to an ASCII string of no more that 32 characters in length.
- **Universal Call ID (ucid)** — Identifier assigned by the ECS.

Notes

There are no restrictions on this capability.

Route Select

Description

The *Route Select* capability allows the server to respond to a client's Route Request. The parameters provide the client with all the information it needs to route the call over the route chosen.

The server uses the information obtained in a `C_RT_REQ` to determine a route for the call. Once a route is selected, the server responds to the client with a `C_RT_SEL` capability. The client then responds with a `C_RT_END` capability which terminates the association. This is a form of negative acknowledgement with a *Cause* returned to the server. If the client has set its local timer when no response is received from the server before time-out, the client can send a `C_ABORT` with a cause of "timer expired."

In the event that no appropriate route can be selected, the server may respond to the client with a cause value using a `C_RT_SEL` capability and `C_NEG_ACK`.

Request/Indication Parameters

- **Destination Address** — The endpoint for which the route has been obtained.

The following parameters are optional:

- **Originating Address** — The endpoint on whose behalf the call is to be routed (the value received in the Route Request or ''not available'').
- **Call Options** — Any of the following:
  - — Priority Calling
  - — Direct Agent Call
  - — ACD Split Identifier
- **User-to-User-Data** — A substructure containing user data. The structure contains the following fields:
  - — `leng` field — An integer value that indicates the number of octets of user data included in the request. This field is set to `0` if no user_user data is present. Currently, the ECS will accept up to 32 bytes of data (`leng` = 32). If more than 32 bytes are specified, all the data is discarded.
  - — `protocol` field — Indicates the type of information and is restricted to `C_UU_USER` which indicates a user-specific protocol or `C_UU_IA5` which indicates IA5 or ASCII characters.
  - — `info` field — A pointer to an ASCII string of no more that 32 characters in length.

Notes

The server may alter the Destination Address value from the value received from the client. The client must use the value provided by the server.

Route End

Description

The *Route End* capability allows the client to inform the server that it wants to terminate the routing association. The routed call, under the control of the client, is not affected by this termination. The reason for the termination is contained in the *Cause* parameter. This is an unacknowledged capability, and as soon as a Route End is issued or received, the routing association terminates.

The client invokes this capability by issuing a `C_RT_END` to terminate the routing ASE. Issuing this capability allows the server to infer that the route it provided in the Route Select was used.

Notes

There are no restrictions on this capability.

Maintenance Capabilities

Maintenance allows a client to request the *Heartbeat* capability. The Abort capability, described later in this chapter, is also part of this capability group. Both the ECS and the adjunct can issue a *Heartbeat* request.

Heartbeat

Description

The *Heartbeat* capability allows the client to query the server for the status of the ASAI link. Heartbeat is a two-way capability; either the ECS or host computer can be the client and can send a heartbeat request.

A positive response from the server means that the link is operating, that messages are received and parsed and that incoming ASAI messages will be delivered to the appropriate ASAI server. A positive response further guarantees that if a client makes a request when the link is operational but the necessary server is not, then an appropriate negative response will be returned.

No response means that the link or the server is not operating. A negative response with a cause of ‘‘response not available” means that the serving node is overloaded and servers cannot accept ASAI requests. The server can also provide a negative acknowledgement with a cause of ‘‘requested facility not subscribed.” In this case, the link is operational, but no conclusions can be drawn about the status of the server(s).

The client invokes this capability by issuing a `C_HB_REQ` to which the server replies with a `C_HB_CONF`.

Request/Indication Parameters

There are no mandatory parameters associated with this capability.

Notes

The Heartbeat indicates the readiness of server(s) on a specific ASAI interface. If there is no response to the heartbeat, then the ASAI link is not operational, but the servers may be serving incoming requests on other ASAI interfaces.

Abort Capabilities

The ASAI library includes an Abort capability that is part of every capability group. The Abort capability provides two types of Aborts:

- **U_ABORT** — A client-initiated abort
- **P_ABORT** — A library-initiated abort

Description

The *Abort* capability allows the client to inform the peer entity (client or server) that processing for the ASE is terminating. The *Cause* parameter indicates the reason for terminating.

The client invokes the Abort capability by issuing a `C_ABORT` with the type of abort specified as `C_USER_ABORT`. The library invokes the Abort capability by issuing a `C_ABORT` with the type of abort returned to the client as `C_PROV_ABORT`.

Request/Indication Parameters

- **Association Identifier**

Notes

There are no restrictions on this capability.

CV/LAN Programming

6

Introduction to CV/LAN

CV/LAN allows application software running on UnixWare, Solaris x86, SPARC Solaris, or Window NT 4.0 to access the DEFINITY ECS CallVisor ASAI features across a TCP/IP LAN in a client-server arrangement. The server runs on the CallVisor PC or the Multi-Application Platform for DEFINITY (MAPD). CallVisor PC and CV/LAN client applications use the same library and differ slightly in only one API call. For further information see ‘‘asai_open( )’’ on page 4-3, in Chapter 4 and ‘‘asai_open (3ASAI)’’ on page 9-10, in Chapter 9, ‘‘Programming Manual Pages.’’

The CV/LAN server supports multithreading and has an independent send and receive path. Most of the message processing and checking is local.

Applications Development Environment

The CV/LAN provides an executable for the server, `asaiserv`. `asaiserv` automatically runs after installation or system reboot.

UNIX Platforms

Basically, the client applications are multithreaded and require sockets. In UnixWare, -Kthread, -lsocket, -lnsl, and -DMTHREAD must be specified. In Solaris, -lthread, -lsocket, -lnsl, and -DMTHREAD must be specified.

Windows NT Platform

In Windows NT, the multithread or debug multithread run time libraries must be specified in the “code generation” category of the “C/C++” tab. The sockets library, wsock32.lib should be included in the list of libraries in the “Link” tab. In addition, the WIN_NTS, CVPC, and ASAIDLL (Dynamic-Link Library) must be included in the settings.

For Windows NT, a Dynamic-Link Library (DLL) called `asaidll.dll` along with its import library `asaidll.lib`, is provided. Required ASAI header files are installed in `<target directory>\asai\include\asai`. Here `<TARGETDIR>` is the path you specify during the installation of the CV/LAN SDK.

After installation, `asaidll.dll and asaidll.lib` will be placed under `<Windows directory>\system32`.

In addition, under Windows NT, some Unix header files are required and provided along with the CV/LAN SDK. The required UNIX header files will be placed under `<targetdirectory>\include\unix`. Some of those header files include their NT namesake as well. For instance, the Unix `fcntl.h` includes the NT `fcntl.h` in the following manner:

```
#include “m:\include\fcntl.h”
```

Here "`m`:" is the drive and path leading to "`include\fcntl.h`". For example, "`m`:" can refer to:

```
c:\Program Files\DevStudio\Vc
```

You MUST properly set "`m`:" with the "`subst`" command at the DOS prompt to avoid getting errors during compilation. For example:

```
subst m: c:\Program Files\DevStudio\Vc
```

The following paths to the ASAI and Unix header files under Windows NT must be included in your development environment:

```
<TARGETDIR>\include\unix

<TARGETDIR>\asai\include
```

By default `<TARGETDIR>` is set as:

```
C:\Program Files\Lucent Technologies\Cvlan
```

Future Upgrade Considerations

Motivation for New CV/LAN Server

In this release, the interface to CV/LAN was changed to increase the likelihood that your application will continue to run with new releases of the ECS to work in a multi-threaded environment, and to improve the throughput by a factor of 2 to 10.

Previously, the applications did not work. From this release forward, the following rules will apply:

1. If you have new client, new ECS with an old server then all the new fields in the existing messages will be sent to the client. But if the client attempts to use the new messages then an error is returned as the old server is not programmed to receive these new information fields.
2. If you have an older client with new ECS and a new server, then the new fields in the existing messages will not be received by the client since the client is not programmed to receive these new information fields.

The CV/LAN speeds up the ASAI applications development cycle by eliminating the need to develop client-server connectivity software. The CV/LAN provides a simple client Application Program Interface (API) that facilitates the CV/LAN application development.

The CV/LAN server will be installed on a UNIX CallVisor PC and will use the ASAI library to interface with the CallVisor stack. Once the system reboots, the CV/LAN server will be started.The server will wait in listen mode for the next client connection. For each client connection it will generate a child server process.

The CV/LAN client API is described below.

Client API

The CV/LAN library functions provide the application program an easy method of sending and receiving information (in the form of ASAI capabilities) to and from the ECS. Before you can send and receive information, however, you must establish a communication path to the CV/LAN server `asaiserv`. This is done by calling `asai_open` with an IP address or machine name instead of “`/dev/asai/asai`” or a similar address.

The CV/LAN client API provides the following library functions:

```
asai_open()
asai_close()
asai_set_env()
asai_get_env()
asai_send()
```

```
asai_rcv()
asai_errval()
```

These are the same routines that are available to all the ASAI applications on the server.

Table 6-1. Client API Library Functions

Function	Purpose
`asai_open`	Open communication path
`asai_errval`	Write error message if function call fails
`asai_set_env`	Establish communication path by setting node ID; set server
`asai_get_env`	Check library version (optional) or server settings
`asai_send`	Send a capability
`asai_rcv`	Receive a capability
`asai_close`	Close the communication path (optional)

The above table describes each function and shows how it works. The functions are presented in a logical, rather than in an alphabetical, order. Thus, `asai_open()` comes first and `asai_close()` comes last as indicated in the above table.

It is possible to process messages from more than one node as long as all the nodes are on the same server.

asai_open

Description

The `asai_open()` is the first library function that your application calls. This function opens a communication path (socket) to the CV/LAN server.

Prototype

```
int asai_open ( const char *path, int flags )
```

Arguments

Path represents a pointer to the hostname or IP address of the machine running the CV/LAN server. `flags` may be set to `0` or `O_NDELAY`.

Return Value

On successful completion, the message `Socket file descriptor` is displayed. If an error occurs, the function returns `-1.`

Example:

```
open_routine()
     {
         int fd;
         if (( fd = asai_open(“135.20.70.72“, O_NDELAY))
                0 )
         {
             exit(-asai_errno);
         }
         .
         .
         .
     }
```

For further information see “asai_open( )’’ on page 4-3, in Chapter 4 and “asai_open (3ASAI)’’ on page 9-10, in Chapter 9, “Programming Manual Pages.’’

asai_errval

The asai_errval is the standard ASAI library function. See ‘‘asai_errval ( )’’ on page 4-4 in Chapter 4, and “asai_errval (3ASAI)’’ on page 9-6 in Chapter 9, “Programming Manual Pages.’’ For error messages, see Chapter 7, “Error Messages.’’

asai_set_env()

The asai_set_env function is also a standard ASAI library function. See “asai_set_env( )’’ on page 4-6, in Chapter 4 and ‘‘asai_set_env (3ASAI)’’ on page 9-18 in Chapter 9, “Programming Manual Pages.’’ However, the node id(s) refer to the node id(s) on the server.

NOTE:
`asai_set_env` should not be used after `asai_rcv` has been called.

asai_get_env()

Description

The asai_get_env() function allows you to check the version of the ASAI library.

C_LIB_VER

The version number of the ASAI library includes three values: major, minor, and delta. A change to a major value indicates significant functional changes to the ASAI library, for example, addition of new capabilities. A change to a minor value indicates changes made to existing capabilities. A delta change indicates a correction with no change in parameters or functionality. The parameters major_ver, minor_ver, and delta_ver represent these values respectively.

C_NUM_NODE

The C_NUM_NODE parameter requests the number of servers that have been assigned by the most recent call to the asai_set_env() function.

arg -> num_node is returned.

C_NODE_ID

The C_NODE_ID parameter requests the node identifier(s) for a server.

C_SERVER

The C_SERVER parameter requests information on the number and type of servers previously set by calls to asai_set_env(). Data of structure type server_type_t, is returned. If no service requests have been set, then num_server is returned as 0.

Prototype

```
long asai_get_env (int socketfd, long attr, get_type * value)
```

Argument

The first argument is the socket file descriptor returned by an asai_open() call. The second argument attr gets the environment attribute. The third argument points to the new value of the attribute.

Return Value

The function returns 0 on successful completion and it returns -1 on failure.

Example

```
if ( asai_get_env( socketfd, attr, &envbuf) < 0 )
 {
    asai_errval("asai_get_env failed");
    exit(-asai_errno);
 }
 .
 .
  .
```

For further information see “asai_get_env( )” on page 4-9 in Chapter 4 and “asai_get_env (3ASAI)” on page 9-7 in Chapter 9, ‘‘Programming Manual Pages.’’

asai_send()

Description

`asai_send()` and `asai_rcv()` together are the central functions in the library. These two functions send all ASAI capabilities and data across an association.

Prototype

```
long asai_send (int socketfd, asai_info_t * buf, long length)
```

Arguments

The first argument is the file descriptor of the socket connection to the CV/LAN server. The second argument is the capability buffer to be sent. The third argument is the size of the buffer being sent.

Return Value

Upon successful completion, the function returns a nonnegative value. If an error occurs, the function returns -1.

Example

```
if (asai_send (socketfd, (char *)(&a3predir),
    sizeof(&a3predir_info_t) < 0)
{
     asai_errval("3rd party redirect alerting call
     request");
     exit(-asai_errno);
}
.
.
```

For further information see"asai_send( )'' on page 4-10 in Chapter 4 and "asai_send (3ASAI)'' on page 9-15 in Chapter 9, "Programming Manual Pages.''

asai_rcv()

Description

`asai_rcv()` allows the user to receive indication primitives. Indications may take either the form of requests, or positive, or negative acknowledgments.

Prototype

```
long asai_rcv (int socketfd, asai_info_t * buf, long length)
```

Arguments

The first argument is the descriptor that identifies the communication path. The second argument is the information buffer to receive the ASAI message. The third argument is the maximum size of the information buffer that may receive the ASAI message.

Return Value

`asai_rcv()` returns the size of the information buffer that must be a value greater than `0` on success and `-1` on failure. `asai_errno` is set to indicate the reason for failure.

Example

```
if (asai_rcv(socketfd,(char *)&rt_info,sizeof
     (rt_info_t)) < 0)
{
          asai_errval("2-minute timer indicates a
           Heartbeat was not issued");
exit(-asai_errno);
}
.
.
.
```

For further information see "asai_rcv( )'' on page 4-12 in Chapter 4 and "asai_rcv (3ASAI)'' on page 9-12 in Chapter 9, "Programming Manual Pages.''

asai_close()

Description

Close a socket to the CV/LAN Server.

Prototype

```
long asai_close(int socketfd)
```

Argument

File descriptor of the socket connection to the CV/LAN server.

Return Value

If the socket was closed successfully, the return value is 0 but if there is an error it is -1.

Example

```
close_routine()
{
      extern int fd;
      if (asai_close(fd) < 0)
      {
          asai_errval("error closing communication
          path");
      }
      return;
}
.
.
.
```

For further information see ‘‘asai_close( )’’ on page 4-13 and
“asai_close (3ASAI)’’ on page 9-5 in Chapter 9, ‘‘Programming Manual Pages.’’

NOTE:
An error may occur if an `asai_rcv` is blocked in another thread.

Error Messages

7

Library Error Messages

Library error messages are listed alphabetically below with a brief explanation of the probable cause of each error. The header file in which these library error messages are found is `asai_err.h`.

`C_NOENT`	No such file or directory. This value is set when the file passed to `asai_open()` does not exist. This will also be returned by CV/LAN, when the machine name cannot be found.
`C_BADCHAR`	Unknown or improper context for a characteristic. This value is set when an invalid characteristic is passed to either `asai_set_env()` or `asai_get_env()`.
`C_BADCHARVAL`	Characteristic's value is invalid. This value is set when `asai_set_env()` is passed an invalid number of servers, or a bad type of server.
`C_BADCLUSTID`	The `Cluster_ID` (also known as `sao_id`) is invalid for the given stream. This value is set when `asai_send()` is passed an initiating request with an ID that matches an existing SAO, or when it passes a noninitiating request with an ID that matches no existing SAO.
`C_BADFD`	File descriptor was not returned by `asai_open()`. This value is set when the file descriptor passed to `asai_close()`, `asai_get_env()`, `asai_rcv()`, `asai_send()`, or `asai_set_env()` is invalid possibly because it had been closed previously.

C_BADFLOW	Communications are flow controlled. This value is set in asai_rcv() if there was no message pending when it was called and the stream was opened in no-delay mode. Also, asai_send() will set this value when it cannot send a message.
C_BADFLAG	An invalid value was given for the asai_open() flags.
C_SYSER	ASAI service error. This error is set in asai_close(), asai_rcv(), and in asai_errval() whenever an error is detected in the operation of the ASAI Application Entity.
C_SERVEX	Service is being provided by another application. This error is set in asai_set_env() whenever it has been passed a service that another Application Process is already providing.
C_BADNODE	Node is not available. This error is set in asai_set_env() whenever it has been passed a Node ID that has no communication path.
C_INTR	A system call was interrupted by a signal. This error is set in all ASAI functions whenever an external event causes a function to return before the operation requested could be performed.
C_OSER	A system call failed. This error is set in all ASAI functions whenever a failure in the operating system causes a function to return before the operation requested could be performed.
C_BADMSG	A corrupt message was received on the given stream. This error is set in asai_send() and asai_rcv() whenever a malformed message was read from a stream. It is recommended that the stream be closed as soon as possible.
C_BADLNG	The send or receive buffer is too small for the capability. This error is set in asai_send() and asai_rcv() whenever the buffer size passed is too small.
C_UNCAP	Cannot send an unknown capability. This error is set in asai_send() whenever an invalid capability is found in the user buffer.

`C_BADPMATCH`	The request has a missing or invalid matching parameter. This error is set in `asai_send()` and `asai_rcv()` whenever a mandatory parameter is missing or when two parameters are used inconsistently. For example, the values in the capability and primitive type parameters must match; an initiating capability with an acknowledgment type is an error. When returned by `asai_rcv()`, this error indicates that a message has been lost.
`C_BADVALUE`	The request has an invalid parameter value. This error is set in `asai_send()` whenever a parameter has an invalid value. Typically, this will result from using a definition not meant for the parameter being set.
`C_TOOBIG`	A variable length field pointed to by a parameter was too big. This error is set in `asai_send()` whenever the user request cannot be sent because of protocol limitations. This may be the result of using variable length strings such as extensions that contain too many characters.
`C_ACTIVE`	The stream has active associations. This error is set in `asai_set_env()` whenever an attempt to change the Node ID cannot be performed because currently SAOs exist.
`C_INVALID_CLIENT`	This error is returned only by MAPD, when a client cannot be validated, that is, its IP address is not administered on the MAPD.
`C_LINKDOWN`	This error is returned by MAPD to notify the application that the ASAI has been taken out of service by the administrator.

ASAI Capability Primitives

8

The capabilities available to the ASAI library functions manage the communications process. This section provides information on data structures common to most or all of the capabilities.

Beginning with G3V2, the server provides additional information for certain capabilities and messages. In order to provide this information to the application, new fields in certain structures have been provided, and in some cases, new structures have been defined. These modifications have been made with the following design goals:

- Affect the API as little as possible
- Maintain consistency
- Require no extraneous information from the application
- *Minimize future changes*

CAUTION:

However, there are instances where it has not been possible to maintain the capability. All messages from the ECS that contain redirecting, calling, called, or connected number IEs can now potentially contain new information. The new information consists of the Type of Address and Numbering Plan fields for the affected IEs. See Chapter 1 of DEFINITY Enterprise Communications Server Release 6 CallVisor ASAI Protocol Reference, 555-230-221, for a description of the Type of Address and Numbering Plan fields as they exist for the Called Party Number IE.

Many messages contain two or more of the affected IEs: redirecting, calling, called, and connected number. To avoid confusing these IEs, it is desirable that the Type of Address and Numbering Plan fields be closely associated to the string of ASCII digits which they are intended to describe.

These redirecting, calling, called, and connected number fields are immediately followed with a structure (of type `plan_type_t`) that contains the Type of Address and Numbering Plan fields.

Another `number_id_t` structure contains a `plan_type_t` structure and a pointer to the ASCII digit string as well. As a result, applications that need to access the Type of Address and Numbering Plan information can now do so. It is recommended that any information stored by these constructs be moved (or cast) to a `number_id_t` structure.

A new structure `ucid_t` has been added. `ucid_t` contains a pointer to the ASCII digit string `id_ptr` and a character string `id_length`.

In those cases where a redirected, calling, called, or connected number field is added to a message (for example, `third party make call ack`), a `number_id_t` structure has been added to the capability structure.

asai_common

The structure `asai_common` defined by typedef `asai_common_t` is part of the data included for each capability. As its name implies, this structure contains information common to all capabilities. This common information is defined as follows:

```
typedef struct{
        capability_t        capability;
        primitive_t         primitive_type;
        long                sao_id;
        long                reserved;
}asai_common_t;
```

Within `asai_common`, the `sao_id` (also known as `cluster_id`) parameter identifies the particular association for which the message is intended. The value of this parameter is an even integer when the capability is user-initiated and an odd integer when the association is initiated by the ASAI library.

The type of service received or transmitted, is identified in capability and is defined as follows:

```
typedef enum {

              C_3PAD,
              C_3PAD_CONF,
              C_3PANS,
              C_3PANS_CONF,
              C_3PCC,
              C_3PCC_CONF,
              C_3PCE,
              C_3PDC_REQ,
              C_3PDC_CONF,
              C_3PDCE,
              C_3PM,
              C_3PM_CONF,
              C_3PMC_REQ,
              C_3PMC_CONF,
              C_3PR,
              C_3PR_CONF,
              C_3PRC,
              C_3PRC_CONF,
              C_3PREDIR,
              C_3PREDIR_ACK,
              C_3PTC_REQ,
              C_3PTC_CONF,
              C_3PSD,
              C_3PSD_CONF,
              C_3PSDS,
              C_3PSDS_CONF,
```

```
                C_3PSH,
                C_3PSH_CONF,
                C_3PSL_DISC
                C_3PSL_DISC_ACk,
                C_3PSL_RECONN
                C_3PSL_RECONN_ACK,
                C_3PSSC_REQ
                C_3PSSC_CONF
                C_3PTC_CONF
                C_3PTC_REQ
                C_ABORT,
                C_EN_CAN,
                C_EN_CAN_CONF,
                C_EN_CONF,
                C_EN_END,
                C_EN_REP,
                C_EN_REQ,
                C_HB_CONF,
                C_HB_REQ,
                C_RF_CONF,
                C_RF_REQ,
                C_RT_END,
                C_RT_REQ,
                C_RT_SEL,
                C_SV_CONF,
                C_SV_REQ,
                C_VQ_CONF,
                C_VQ_REQ,
                C_VQ_RESP,
                C_EN_SCN,
                C_EN_SCN_CONF,
                C_RM_REQ,
                C_RM_CONF,
                C_SM_REQ,
                C_SM_CONF,
} capability_t;
```

The type of request or indication is identified in `primitive_type`:

```
typedef enum {
        C_REQUEST,
        C_POS_ACK,
        C_NEG_ACK
} primitive_t;
```

The values of `C_REQUEST`, `C_POS_ACK`, and `C_NEG_ACK` indicate a request, a positive acknowledgement, and a negative acknowledgement, respectively.

The *reserved* parameter, which appears in most of these common structures, simply means that the parameter is being reserved for future use.

Identifiers

The ASAI library routinely uses a number of identifiers, defined in structures of the types shown. All of these structures include a field reserved for future use. The most common identifiers are:

- Call identifier (`call_id`) — Identifies a specific call.
- Party identifier (`party_id`) — Identifies specific parties on a call.
- Old party identifier (`old_party_id`) — Identifies an ECS-assigned identifier before two calls are merged.
- Trunk identifier (`trunk_id`) — Identifies a trunk.
- Party list identifier (`party_ext`) — Identifies a list of parties with their corresponding extensions.
- Station information (`stn_info`) — Identifies a list of calls with their corresponding parties and call states.
- Merge extension (`merge_ext`) — Identifies a list of parties with their corresponding extensions.
- User to user (`user_user`) — Identifies information between ISDN users.
- Originating Line Identifier (`oli`) — Identifies Information Indicator (II) digits received in the originating line information IE for the call.
- User data — (`user_data`) Identifies user-supplied digits.
- Universal Call ID (ucid) — identifies the UCID assigned by the ECS.

call_id

```
typedef struct{
        char                *id_ptr;
        long                id_length;
        long                reserved;
}call_id_t;
```

Within the structure of type `call_id_t`, the parameter `id_length` indicates the number of binary bytes contained in the call identifier and `id_ptr` is the value of a pointer to an array of binary bytes that identifies the call. If call id is present id length must be 2. If its value is 0, the value of `id_ptr` is undefined. If an illegal value other than 0 is encountered, `asai_send()` will reject the message with an error code of `C_BADPMATCH`. Future releases of the ECS may change the legal range of values for `id_length`. Defensive programming practices are recommended.

Note that unlike null-terminated character strings such as `calling_num` and `called_num`, `id_ptr` is not a null-terminated character string. Rather, it is an array of bytes containing binary values. The parameter `id_length` indicates the number of binary bytes contained in the call identifier.

NOTE:
call_id is not an ASCII null-terminated string but rather an array of binary bytes of the specified length. The length is subject to change.

party_id

```
typedef struct{
        char                    *id_ptr;
        long                    id_length;
        long                    reserved;
}party_id_t;
```

Within party_id_t, the parameter id_length indicates the number of bytes contained in the party identifier, and id_ptr is the value of a pointer to the array of binary bytes identifying the party. The legal value for id_length is 1; if its value is 0, the value of id_ptr is undefined. If an illegal value other than 0 is encountered, asai_send() will reject the message with an error code of C_BADPMATCH. The legal value of id_length may change in future releases of the ECS. Defensive programming practices are recommended.

Note that unlike null-terminated character strings such as calling_num and called_num, id_ptr is not a null-terminated character string. Rather, it is an array of binary bytes. The parameter id_length indicates the number of bytes contained in the call identifier.

NOTE:
party_id is *not* an ASCII null-terminated string but rather an array of binary bytes of the specified length. The length is subject to change.

old_party_id

```
typedef struct{
        char        *id_ptr;
        long        which_call;
        long        id_length;
        long        reserved;
} old_party_id_t;
```

Within old_party_id_t, *id_ptr is a pointer to an array of binary bytes of length id_length that indicates an ECS-assigned party identifier before two calls are merged. See “party_id” for more information.

The parameter which_call can assume the values of C_RESULTING_CALL or C_OTHER_CALL and indicates whether the old party belongs to the resulting call or to other calls.

Note that in the structure of type party_id_t, id_ptr is not a null-terminated character string.

trunk_id

```
typedef struct{
        char        *gid_ptr;
        char        *id_ptr;
        long        gid_length;
        long        id_length;
        long        direct;
        long        reserved;
} trunk_id_t;
```

Within trunk_id_t, gid_ptr is a pointer to an array of binary bytes indicating an ECS-assigned trunk group identifier of length gid_length. id_ptr is a pointer to an array of binary bytes indicating an ECS-assigned identifier, in the group identified by gid_ptr. The length of this identifier is id_length. The optional parameter direct can assume the values of C_NODIRET, C_INCTRK or C_OUTTRK, the direction in which the trunk was used: no direction, incoming or outgoing. If this parameter is not specified, it assumes the value C_NUSE_LONG.

If the trunk group number (TGN) is less than or equal to 127, then the number pointed to by gid_ptr matches the TGN. Otherwise, it is 16 times the TGN, (divide by 16 to get TGN).

plan_type

```
typedef struct{
        short        addr_type;
        short        numb_plan;
}  plan_type_t;
```

Within plan_type_t, the parameter addr_type is a short integer that can have these possible values: unknown(0), international(1), national(2) and subscriber(4). The parameter numb_plan is also a short integer and can have these values: unknown(0), ISDN/telephony(1), reserved(2) and private numbering plan(9). Additional values for both parameters may be coded in the future.

number_id

```
typedef struct{
        char             *s;
        plan_type_t      type_plan;
}  number_id_t;
```

Within number_id_t, the s field, when not NULL, is an ASCII string that specifies the connected number. If s is NULL, the following field, type_plan, is undefined. The type_plan field is of type plan_type_t.

party_ext

```
typedef struct{
        party_id_t      party_id;
        char            *extension;
        plan_type_t     ext_type;
}party_ext_t;
```

Within party_ext_t, the parameter party_id is a structure of type party_id_t. The extension parameter points to a null-terminated string that specifies the extension corresponding to the party identifier.

ext_type is a structure of type plan_type_t that supplies additional information about the extension field that immediately precedes it (providing that the extension field is not NULL).

stn_info

```
typedef struct{
        call_id_t            call_id;
        party_id_t           party_id;
        long                 pty_state;
        long                 reserved;
}stn_info_t;
```

Within stn_info_t, the call_id parameter is a structure of type call_id_t, where id_length specifies the length of an ECS-assigned call identifier and id_ptr is a character string that specifies the call identifier. The party_id parameter points to a structure of type party_id_t, where id_length specifies the length of an ECS-assigned party identifier and id_ptr points to a character string that specifies the party identifier. The pty_state parameter specifies the current state (busy, alert, etc.) of the corresponding party identifier. In stn_info the return value of the pty_state will have the same value as the “cause fields” in various event report messages. See Table 9-1 on page 9-115 in Chapter 9 for more information.

Whenever the structures of type party_id_t, call_id_t or trunk_id_t are used as optional parameters and the information is not supplied, id_ptr is a null pointer and id_length has a value of zero.

merge_ext

```
typedef struct{
        old_party_id_t       old_pid;
        party_id_t           party_id;
        char                 *extension;
        plan_type_t          ext_type;
}merge_ext_t;
```

Within `merge_ext_t`, the parameter `old_pid` is a structure of type `old_party_id_t`, which provides the old party identifier (see "old_party_id" on page 8-6).

The parameter `party_id` is a structure of type `party_id_t` that indicates an ECS-assigned party identifier after two calls are merged (see "party_id" on page 8-6 for details).

The `extension` parameter points to a null-terminated string that specifies the extension that corresponds to the party.

`ext_type` is a structure of type `plan_type_t` that supplies additional information about the `extension` field that immediately precedes it (providing that the `extension` field is not NULL).

user_user

```
typdef struct {
        long       leng;
        long       protocol;
        char       *info;
} user_user_t;
```

The purpose of the `user_user` structure is to convey information between ISDN users.

`uudata` is a structure of the type `user_user_t` where the `leng` field is an integer value that indicates the number of bytes of user data included in the request. This field is set to 0 if no `user_user` data is present. Currently, the ECS will accept up to 32 bytes of data (`leng` = 32). If more than 32 bytes is specified, an error will be returned. ASAI supports a maximum `user_user` data length of 32 bytes, although `user_user` data generated from a PRI trunk can be up to 127 bytes in length. The DEFINITY ECS will truncate this to 32 bytes before forwarding it to ASAI link.

The `protocol` field indicates the type of information and is restricted to `C_UU_USER` that indicates a user-specific protocol or `C_UU_IA5`, which indicates IA5 or ASCII characters. The `info` field is a pointer to an ASCII string of no more that 32 characters in length.

user_data

```
typedef struct{
   long                type;
   long                collect;
   long                timeout;
   char                *digits;
}user_code_t;
```

The user_data structure is used to convey the information about the DTMF digits to the ECS, and with the collected digits event report to convey information to the adjunct. The leng field is an integer value that indicates the number of octets (digits) of user_data included in the request. The type field indicates the type of information and is restricted to C_UU_IA5, which indicates IA5 or ASCII characters.

The digits field is a pointer to an ASCII string of no more that 32 characters in length.

oli

```
typedef struct {
         char    *oli_ptr;
         long    oli_length;
         long    reserved;
}oli_t;
```

Since the ECS does not accept any II-digits from an ASAI adjunct, CallVisor PC only supports receiving this IE not sending it.

The field oli_length is the length of the byte array pointed to by the field oli_ptr. If the IE is missing from the message or its length is zero, oli_length is zero. In all other cases, it is the size of the array.

The field oli_ptr is a pointer to an array of binary bytes.

This substructure is included in the following ASAI event report capability structures:

1. Call Offered to Domain (incall_list_t)
2. Alerting (alert_list_t)
3. Connected (connect_list_t)

It is also included in the Route Request capability structure (rt_info_t).

ucid

```
typedef struct {
        char    *id_ptr;
        long    id_length;
}ucid_t;
```

Since the ECS does not accept any UCID from an ASAI adjunct, CallVisor PC only supports receiving this IE not sending it.

The field id_ptr is a pointer to an array of binary bytes.

The field id_length is the length of the byte array pointed to by the field id_ptr. If the IE is missing from the message or its length is zero, id_length is zero. In all other cases, it is the size of the array.

This substructure is included in the following ASAI event report capability structures:

1. Alerting (alert_list_t)
2. Call conferenced (conf_list_t)
3. Call initiated (initiate_list_t)
4. Call offered to Domain (incall_list_t)
5. Call connected (connect_list_t)
6. Call transferred (tran_list_t)

It is also included in the Route Request capability structure (rt_info_t).

cause_value

Certain capabilities return a parameter called `cause_value` if a negative acknowledgement is sent or received. These `cause_values` are noted with each capability. In general, a `cause_value` is returned by the ECS and mapped to an ASAI `cause_value`. Table 8-1 lists ASAI `cause_values` by name, shows the DEFINITY ECS value, the ASAI library value, and provides a brief explanation of the probable cause.

Table 8-1. ASAI Library and the DEFINITY ECS Cause Values

Cause Value	DEFINITY ECS Value	ASAI Value	Description
C_ADMIN_PROGRESS	CS3/46	30	Administration is in progress; request cannot be serviced.
*C_AGT_STATE	CS3/16	23	Request to put agent in same state he/she is already in.
C_ANSWERING_MACHINE	CS3/24	55	Answering machine detected.
*C_BAD_ADMIN	CS3/41	24	ACD not provisioned or optioned.
*C_BAD_IE	CS0/99	27	Unknown information element detected.
C_BEARER_CAP_UNAVAIL	CS0/58	38	Bearer capability not presently available.
C_BEARER_SVC_NOT_IMPL	CS0/65	51	Bearer service not implemented.
C_CALL_REJECTED	CS0/21	33	Call rejected.
*C_CALLID_TERM	CS3/86	8	Call is no longer in active state.
*C_CLUST_TERM	CS3/87	18	Association terminated because service is not active.
*C_CAUSE_UNKNOWN	Undefined	48	Undefined value returned from the ECS.
C_FACILITY_REJECTED	CS0/29	56	Facility rejected.
*C_FACUNSUB	CS0/50	3	Capability is implemented but not subscribed to by requester.
C_FEATURE_REJECTED	CS3/53	31	The ECS has rejected a request from the adjunct.
C_INCOMPAT_DESTINATION	CS0/88	39	Incompatible destination.
*C_INCOM_ST	CS0/98	9	Message not compatible with call state.
*C_INCOM_OPT	CS3/80	11	Incompatible options used to establish the call.
*C_INC_PASWD	CS3/14	22	Invalid login password.

Table 8-1. ASAI Library and the DEFINITY ECS Cause Values — *Continued*

Cause Value	DEFINITY ECS Value	ASAI Value	Description
*C_INCS_AGT_ST	CS3/12	20	Agent not in compatible state.
C_INVALID_MESSAGE	CS0/95	40	Invalid message, unspecified (backward compatibility).
*C_INVALID_CRV	CS0/81	10	Invalid call identifier (`sao_id` also known as `cluster_id`) used or call does not exist.
*C_INVLDIE	CS0/100	6	Value specified in parameter is not allowed or defined.
*C_INVLDNUM	CS0/28	0	Invalid origination or destination address.
*C_MAND_INFO	CS0/96	5	One of the required parameters is missing.
*C_MAXLOGIN	CS3/13	21	Agent logged into maximum number of splits.
C_NO_TONE_CHANNEL	CS0/82	54	Channel and/or tone does not exist (no tone connected to the specified call).
C_NUM_CHANGED	CS0/22	34	Number changed.
*C_NETCONJ	CS0/42	26	Association terminated because of network congestion.
C_NETWORK_OUT_OF_ORDER	CS3/38	47	Network out of order.
C_NO_ANSWER	CS3/19	43	No answer.
C_NO_CIRCUIT	CS0/34	36	No circuit or channel available.
C_NO_CLASSIFIERS	CS3/21	45	Classifiers not available.
*C_NOLOGIN	CS3/15	13	Agent not logged into split.
C_NON_EXIST_MESSAGE	CS0/97	41	Message nonexistent/not implemented.
*C_NORMAL	CS0/16	25	Normal termination; call routed successfully.
C_NORMAL_UNSPECIF	CS0/31	35	Normal, unspecified.
*C_NOSPLIT_MEM	CS3/11	14	Agent not member of specified split or split number specified incorrectly.
C_NO_TRUNKS	CS3/20	44	Trunks not available.
*C_NOUSE_RESP	CS0/18	16	Originating address does not respond to service.
*C_NUSE_LONG	none	-MAX_LONG	No value was returned by the ECS.

Table 8-1. ASAI Library and the DEFINITY ECS Cause Values — *Continued*

Cause Value	DEFINITY ECS Value	ASAI Value	Description
*C_OUT_CALL_BARRED	CS0/52	49	Outgoing call has been barred.
*C_OUT_OF_SERV	CS3/27	19	Domain has been removed by administration.
*C_PERM_DENIED	CS3/43	17	Permission checks for service have failed.
*C_PROTERR	CS0/111	1	Capability sequence has been violated or underlying protocol error has been detected; an unrecognized value was returned by the ECS.
*C_QUEFULL	CS3/22	28	Queue is full.
*C_REC_TIMER	CS0/102	12	Timer expired.
C_REDIRECT	CS3/30	46	Redirected.
C_REMAINS_IN_Q	CS3/23	50	Call remains in queue.
C_REORDER_DENIAL	CS3/42	29	Reorder/Denial.
*C_RESUNAVL	CS3/40	2	Resources to fulfill service are not available.
*C_SER_UNIMP	CS3/79	4	Noncompatible options selected.
*C_SERV_UNAVIL	CS3/63	7	Domain or call is being monitored by another adjunct.
C_TEMP_FAILURE	CS0/41	37	Temporary Failure.
C_TIMED_ANSWER	CS3/17	52	Assumed answer based on internal timer.
C_UNASSIGNED_NUM	CS0/1	32	Unassigned number.
C_UNSPECIFIED	CS0/127	42	Unspecified.
*C_USER_BUSY	CS0/17	15	Domain or call is being monitored by another adjunct.
C_VOICE_ENERGY_ANSWER	CS3/18	53	Voice energy detected by the ECS.
C_COVER_BUSY	CS3/26	58	Redirection cause.
C_COV_DONT_ANS	CS3/28	59	Redirection cause.
C_FORWARD_ALL	CS3/31	60	Redirection cause.
C_FORWARD_BUSY	CS3/25	57	Redirection cause.
C_LISTEN_ONLY	CS3/8	61	Single-Step Conference listen only
C_LISTEN_TALK	CS3/9	62	Single-Step Conference listen-talk

NOTE:
Whenever the server returns `cause_value` with an unrecognized value, the value is mapped to `C_PROTERR`. If the server does not return a value, then `cause_value` is mapped to `C_NUSE_LONG`.

Future releases of the ASAI library may include new `cause_values`. The application can be programmed for upward compatibility by providing for unexpected `cause_values` in the code.

Pool

The pool parameter, used with a number of capabilities, identifies a buffer area for storing user information. The various call_identifier values returned by these capabilities (`call_id, party_id`) are pointers to the pool buffer.[1] The `pool` buffer (`pool [C_DATSZ]`) contains various call-identifier numbers in the form of null-terminated strings.

1. The programmer must do more than simply copy these identifiers to guarantee that the space to which they point, will exist in the future.

Programming Manual Pages

9

ASAI Library Manual Pages

Access to ASAI services is provided through a collection of functions known as the ASAI library. A single library can:

- Access multiple ports (IPCI boards) or LAN Gateway links
- Communicate with one or more switches
- Manage communications for multiple ASAI links and ports

These capabilities are provided by specifying a different `node_id` argument to the `asai_set_env` function for each newly opened file descriptor on which the programmers would like to start new associations. Up to 8 `node_ids` are supported; signal01 through signal08 correspond to links 1 through 8, respectively.

A program can request client services (that is, create a new association) from only one `c_node_id` per file descriptor. To obtain services from four switches, the program must obtain four file descriptors by opening the library four times. Different `c_node_ids` must then be specified to the `set_env` function for each file descriptor.

ASAI functions are not reentrant but they are threadsafe. Because of this, interrupt handlers that are invoked when an application is interrupted during a function call should not themselves invoke another library function. Application programmers must set interrupt levels with this restriction in mind.

The library is identified by its version number. Various releases of the library are distinguished using *major, minor* and *delta* version numbers.

A major release is defined by a technological level or major change in operation from a previous release. A minor release is an enhancement or refinement to an existing major release. A delta release is a small enhancement or bug fix, differently tuned, or specific to a hardware platform.

Applications Development Environment for CV/LAN

The CV/LAN provides an executable for the server, `asaiserv`. `asaiserv` automatically runs after installation or system reboot.

UNIX Platforms

Basically, the client applications are multithreaded and require sockets. In UnixWare, -Kthread, -lsocket, -lnsl, and -DMTHREAD must be specified. In Solaris, -lthread, -lsocket, -lnsl, and -DMTHREAD must be specified.

The required UNIX header files will be placed under `<targetdirectory>\include\unix.`

Windows NT Platform

In Windows NT, a MS Visual C 5.0 environment must be set as follows:

The multithread or debug multithread run time libraries must be specified in the “code generation” category of the “C/C++” tab. The sockets library, wsock32.lib should be included in the list of libraries in the “Link” tab. In addition, the WIN_NTS, CVPC, and ASAIDLL (Dynamic-Link Library) must be included in the settings.

For Windows NT, a Dynamic-Link Library (DLL) called `asaidll.dll` along with its import library `asaidll.lib,` is provided. Required ASAI header files are installed in `<target directory>\asai\include\asai.` Here `<TARGETDIR>` is the path you specify during the installation of the CV/LAN SDK.

After installation, `asaidll.dll and asaidll.lib` will be placed under `<Windows directory>\system32`.

In addition, under Windows NT, some Unix header files are required and provided along with the CV/LAN SDK. Some of those header files include their NT namesake as well. For instance, the Unix `fcntl.h` includes the NT `fcntl.h` in the following manner:

```
#include “m:\include\fcntl.h”
```

Here "m:" is the drive and path leading to "include\fcntl.h". For example, "m:" can refer to:

```
c:\Program Files\DevStudio\Vc
```

You MUST properly set "m:" with the "subst" command at the DOS prompt to avoid getting errors during compilation. For example:

```
subst m: c:\Program Files\DevStudio\Vc
```

The following paths to the ASAI and Unix header files under Windows NT must be included in your development environment:

```
<TARGETDIR>\include\unix

<TARGETDIR>\asai\include
```

By default <TARGETDIR> is set as:

```
c:\Program Files\Lucent Technologies\Cvlan
```

ASAI Library Functions

ASAI library functions do the following:

- Manage communications paths
  - asai_open
  - asai_close
  - asai_set_env
  - asai_get_env
- Manage associations over an established communications path
  - asai_rcv
  - asai_send
- Assist in debugging
  - asai_errval

On the following pages, each library function is described by some or all of the following sections:

Name

This section names and succinctly describes the function.

Synopsis

This section illustrates the declaration of the function, including necessary header files.

Description

This section describes the function, including its parameters.

Return Value

This section gives the return values of the function on success and failure.

Errors

This section lists error values and describes the errors that are set in `asai_errno`. `asai_errno` is defined as follows:

```
long asai_errno;
```

Notes

This section is used to describe any exceptions, caveats, or usage suggestions pertaining to the function.

See Also

This section lists related functions and ASAI capabilities, where appropriate.

asai_close (3ASAI)

Name

asai_close — Close the communication path

Synopsis

```
#include  <asai/asai_def.h>
#include  <asai/asai_str.h>
#include  <asai/asai_err.h>

long  asai_close(fd)
int  fd;
```

Description

This function closes the communication path identified by fd. All activity associated with the specified fd is terminated.

Return Value

Upon successful completion, the function returns 0. If an error occurs, the function returns -1.

Errors

The type of error is indicated by the error value set in asai_errno:

C_BADFD — Invalid file descriptor supplied.

C_SYSER — ASAI service error, cannot close the Streams device. Check errno.

See Also

asai_open()

asai_errval (3ASAI)

Name

`asai_errval` — Write specified error message to `stderr`

Synopsis

```
#include  <asai/asai_def.h>
#include  <asai/asai_str.h>
#include  <asai/asai_err.h>

long  asai_errval(mes_buf)
char  mes_buf[C_MESIZE];
```

Description

This function allows the client to incorporate a specified error message with a message provided by the ASAI library. The programmer specifies the message in the null-terminated string `mes_buf` of maximum size `C_MESIZE`. The ASAI-supplied message maps to the value `asai_errno`. The entire message is written to the standard error device, the program-supplied message first, then a colon (:) and then the ASAI message.

Return Value

Upon successful completion, the function returns 0. If an error occurs, the function returns -1.

Errors

The type of error is indicated by the error value set in `asai_errno`:

`C_SYSER` — ASAI service error. It indicates that `stderr` cannot be written; check errno.

NOTE:
This function is not available on Windows/NT.

asai_get_env (3ASAI)

Name

asai_get_env — Request information about a specified characteristic of a communication path

Synopsis

```
#include  <asai/asai_def.h>
#include  <asai/asai_str.h>
#include  <asai/asai_err.h>

typedef union{
        version_t      version;
        ulong          num_node;
        server_type_t  server;
        char           node_id[C_NODSIZ];
}get_type;

long  asai_get_env(fd, characteristic, arg)
int  fd;
long  characteristic;
get_type  *arg;
```

Description

This function is used to obtain a specified characteristic of the communication path associated with `fd`. The parameter `characteristic` can assume any of the following values:

`C_LIB_VER`

The `C_LIB_VER` parameter requests the version of the ASAI library. Data of the structure type `version_t` (shown above) is returned, pointed to by the third argument.

`C_NUM_NODE`

The `C_NUM_NODE` parameter requests the number of servers that have been assigned by the most recent call to the `asai_set_env()` function.

`arg -> num_node` is returned.

`C_NODE_ID`

The `C_NODE_ID` parameter requests the node identifier(s) for a server.

C_SERVER

The C_SERVER parameter requests information on the number and type of servers previously set by calls to asai_set_env(). Data of structure type server_type_t, defined below, is returned. If no service requests have been set, then num_server is returned as 0.

```
typedef struct{
        long            num_node;
        service_spec_t  *buf;
}server_type_t;
```

Within this structure, num_node indicates the number of structures of the type service_spec_t, defined below, that can be contained in the buffer pointed to by buf.

```
typedef struct{
        long  server_type;
        char  node_id[C_NODSIZ];
}service_spec_t;
```

Upon return, the ASAI library indicates the number of structures actually supplied in buf, but no more than the number requested by the user. Within the service_spec_t structure, server_type indicates the service requests that will be accepted and node_id indicates the node from which the service requests are accepted. Valid server types that can be put together with the option of selecting either one are:

C_RT_SER — Routing

C_MAINT_SER — Maintenance

Return Value

Upon successful completion, the function returns 0. If an error occurs, the function returns -1.

Errors

The type of error is indicated by the error value set in asai_errno:

C_BADFD — Invalid file descriptor supplied

C_BADCHAR — Unknown or improper context for a characteristic

Notes

The library returns information *only* on the number of nodes you specify. If you specify fewer than the actual number of nodes, information is returned on the nodes set in the first calls to `asai_set_env`. It is recommended that you specify `num_node` as 30 or to any configurable parameter. For example:

```
# define MAXNUMSERV 30
server_type_t    server = {maxnumserv, services};
service_spec_t   services[MAXNUMSERV];
```

See Also

`asai_open() asai_rcv() asai_send() asai_set_env()`

asai_open (3ASAI)

Name

asai_open — Open a communication path

Synopsis

```
# include <fcntl.h>
#include  <asai/asai_def.h>
#include  <asai/asai_str.h>
#include  <asai/asai_err.h>

long  asai_open(pathname, ndelay_flag)
char  *pathname;
long  ndelay_flag;
```

Description

This function opens a communication path identified by pathname, a null-terminated character string; the default path is /dev/asai/asai. If the ndelay_flag is set to O_NDELAY, the operation is asynchronous and the send or receive request does not block. If the ndelay_flag is set to !_NDELAY, the operation is synchronous and the send or receive request blocks. In the synchronous mode, control is not returned to the user until the send or receive request has been completed or an error occurs.

Return Value

Upon successful completion, the function returns the integer value of fd. If an error occurs, the function returns -1. The file descriptor can be used by subsequent calls to ASAI library functions.

Errors

The type of error is indicated by the error value set in asai_errno:

C_BADFD — Invalid file descriptor supplied

C_BADFLAG — An invalid value was given for the asai_open() flag

C_NOENT — No such file or directory

C_OSER — A system call failed, check errno

C_INVALID_CLIENT — This error is returned only by MAPD, when a client cannot be validated, that is, its IP address is not administered on the MAPD.

C_LINKDOWN — This error is returned by MAPD to notify the application that the ASAI has been taken out of service by the administrator.

See Also

asai_close() asai_rcv() asai_send()

asai_rcv (3ASAI)

Name

asai_rcv — Receive message via the communication path

Synopsis

```
#include  <asai/asai_def.h>
#include  <asai/asai_str.h>
#include  <asai/asai_err.h>

typedef union{
        asai_common_t      asai_common;
        abort_info_t       abort_info;
        vq_buf_t           vq_buf;
        vqr_buf_t          vqr_buf;
        vqc_buf_t          vqc_buf;
        en_buf_t           en_buf;
        en_rsp_t           en_rsp;
        evr_buf_t          evr_buf;
        enc_rsp_t          enc_rsp;
        ene_info_t         ene_info;
        a3pmc_info_t       a3pmc_info;
        a3pmc_ack_t        a3pmc_ack;
        a3pmc_nak_t        a3pmc_nak;
        a3pmc_rsp_t        a3pmc_rsp;
        a3ptc_info_t       a3ptc_info;
        a3ptc_ack_t        a3ptc_ack;
        a3ptc_nak_t        a3ptc_nak;
        a3psh_info_t       a3psh_info;
        a3psh_rsp_t        a3psh_rsp;
        a3psd_info_t       a3psd_info;
        a3psd_rsp_t        a3psd_rsp;
        a3psl_disc_t       a3psl_disc;
        a3psl_disc_ack_t   a3psl_disc_ack;
        a3psl_rconn_t      a3psl_reconn;
        a3psl_reconn_ack_t a3psl_reconn_ack;
        a3pr_info_t        a3pr_info;
        a3pr_rsp_t         a3pr_rsp;
        a3prc_rsp_t        a3prc_rsp;
        a3pm_info_t        a3pm_info;
        a3pm_ack_t         a3pm_ack;
        a3pm_nak_t         a3pm_nak;
        a3pcc_rsp_t        a3pcc_rsp;
        a3pce_info_t       a3pce_info;
        a3pad_info_t       a3pad_info;
        a3pad_ack_t        a3pad_ack;
        a3pad_nak_t        a3pad_nak;
```

```
        a3pans_info_t      a3pans_info;
        a3pans_nak_t       a3pans_nak;
        a3pdc_info_t       a3pdc_info;
        a3pdc_ack_t        a3pdc_ack;
        a3pdc_nak_t        a3pdc_nak;
        a3pdce_info_t      a3pdce_info;
        a3pssc_info_t      a3pssc_info;
        a3pssc_ack_t       a3pssc_ack;
        a3pssc_nak_t       a3pssc_nak;
        en_scn_t           en_scn;
        en_scn_nak_t       en_scn_nak;
        sv_buf_t           sv_buf;
        sv_rsp_t           sv_rsp;
        rf_buf_t           rf_buf;
        rf_rsp_t           rf_rsp;
        rt_info_t          rt_info;
        rts_info_t         rts_info;
        rte_info_t         rte_info;
        rt_nak_t           rt_nak;
        hb_nak_t           hb_nak;
        hb_rsp_t           hb_rsp;
        en_scn_rsp_t       en_scn_rsp;
        a3psds_info_t      a3psds_info;
        a3psds_rsp_t       a3psds_rsp;
        a3predir_info_t    a3predir_info;
        a3predir_nak_t     a3predir_nak;
        a3predir_ack_t     a3predir_ack;
}asai_info_t;

long  asai_rcv(fd, buf, length)
int   fd;
char  *buf;
long  length;
```

Description

This function is used to read a message (that is, a packet containing primitives, capabilities and/or program-specific information) from the communication path identified by file descriptor `fd`. The program passes a pointer to the buffer, buf, of size `length`, to the ASAI library. The ASAI library then returns the message into the program-specified buffer.

The maximum useful size of the user buffer is `sizeof (asai_info_t)`.

In the event that a received message will not fit in the buffer supplied by the programmer, the ASAI library returns a -1 (error) and sets `asai_errno` to -14 `(C_BADLNG)`. The message is saved for the next call.

NOTE:
If a program ignores this error and simply calls the library again, an infinite loop is created. Because of this, all application programs must check for the presence of a `C_BADLNG` error and provide a larger buffer if it occurs.

If the function is used in a synchronous mode of operation, `asai_rcv()` blocks until an entire message has been received, until an error occurs, or until `fd` is closed by an interrupt function. If the function is used in an asynchronous mode of operation, `asai_rcv()` does not block.

The library can assume the role of the server (that is, a program that responds to the ECS initiated associations) only for the route and maintenance (heartbeat) services. For all other services it must assume a client role. `asai_rcv()` will never return an inappropriate message for the current role(s).

Return Value

Upon successful completion, the function returns the size of the message. If an error occurs, the function returns -1. A return code of 0 does not indicate an error; the received message was intended for the library and no program data is available at this time.

Errors

The type of error is indicated by the error value set in `asai_errno`:

`C_BADFD` — Invalid file descriptor supplied

`C_BADFLOW` — Cannot issue the request because of flow control

`C_SYSER` — ASAI service error received a message on an unknown association. This message will be returned only if `sao_id` (formerly known as `cluster_id`) in a route request is already in use.

`C_BADLNG` — The send or receive buffer is too small for the capability

`C_BADMSG` — A corrupt message was received on the given stream

`C_INTR` — A system call was interrupted by a signal

`C_OSER` — A system call failed, check errno

See Also

`asai_open()  asai_send()  asai_set_env()`

NOTE:
The buf argument may be filled with zeroes even if the function call fails.

asai_send (3ASAI)

Name

asai_send — Send message via the communication path

Synopsis

```
#include  <asai/asai_def.h>
#include  <asai/asai_str.h>
#include  <asai/asai_err.h>

typedef union{
        asai_common_t      asai_common;
        abort_info_t       abort_info;
        vq_buf_t           vq_buf;
        vqr_buf_t          vqr_buf;
        vqc_buf_t          vqc_buf;
        en_buf_t           en_buf;
        en_rsp_t           en_rsp;
        evr_buf_t          evr_buf;
        enc_rsp_t          enc_rsp;
        ene_info_t         ene_info;
        a3pmc_info_t       a3pmc_info;
        a3pmc_ack_t        a3pmc_ack;
        a3pmc_nak_t        a3pmc_nak;
        a3pmc_rsp_t        a3pmc_rsp;
        a3ptc_info_t       a3ptc_info;
        a3ptc_ack_t        a3ptc_ack;
        a3ptc_nak_t        a3ptc_nak;
        a3psh_info_t       a3psh_info;
        a3psh_rsp_t        a3psh_rsp;
        a3psd_info_t       a3psd_info;
        a3psd_rsp_t        a3psd_rsp;
        a3psl_disc_t       a3psl_disc;
        a3psl_disc_ack_t   a3psl_disc_ack;
        a3psl_rconn_t      a3psl_reconn;
        a3psl_reconn_ack_t a3psl_reconn_ack;
        a3pr_info_t        a3pr_info;
        a3pr_rsp_t         a3pr_rsp;
        a3prc_rsp_t        a3prc_rsp;
        a3pm_info_t        a3pm_info;
        a3pm_ack_t         a3pm_ack;
        a3pm_nak_t         a3pm_nak;
        a3pcc_rsp_t        a3pcc_rsp;
        a3pce_info_t       a3pce_info;
        a3pad_info_t       a3pad_info;
        a3pad_ack_t        a3pad_ack;
        a3pad_nak_t        a3pad_nak;
```

```
        a3pans_info_t      a3pans_info;
        a3pans_nak_t       a3pans_nak;
        a3pdc_info_t       a3pdc_info;
        a3pdc_ack_t        a3pdc_ack;
        a3pdc_nak_t        a3pdc_nak;
        a3pdce_info_t      a3pdce_info;
        a3pssc_info_t      a3pssc_info;
        a3pssc_ack_t       a3pssc_ack;
        a3pssc_nak_t       a3pssc_nak;
        en_scn_t           en_scn;
        en_scn_nak_t       en_scn_nak;
        sv_buf_t           sv_buf;
        sv_rsp_t           sv_rsp;
        rf_buf_t           rf_buf;
        rf_rsp_t           rf_rsp;
        rt_info_t          rt_info;
        rts_info_t         rts_info;
        rte_info_t         rte_info;
        rt_nak_t           rt_nak;
        hb_nak_t           hb_nak;
        hb_rsp_t           hb_rsp;
        en_scn_rsp_t       en_scn_rsp;
        a3psds_info_t      a3psds_info;
        a3psds_rsp_t       a3psds_rsp;
        a3predir_info_t    a3predir_info;
        a3predir_nak_t     a3predir_nak;
        a3predir_ack_t     a3predir_ack;
   }asai_info_t;

   long  asai_send(fd, buf, length)
   int   fd;
   char  *buf;
   long  length;
```

Description

This function formats and sends an ASAI message to the ECS through the stream file descriptor `fd`. The information used to construct this message is contained in a buffer pointed to by `buf` of size `length`.

The maximum useful size of the user buffer is size of`(asai_info_t)`.

If the function is used in a synchronous mode of operation, `asai_send()` blocks until an entire message has been queued for transmission, or an error occurs. If the function is used in an asynchronous mode of operation, `asai_send()` does not block and returns an error.

The library can assume the role of server only for the route and maintenance (heartbeat) services. For all other services it must assume a client role. Any

attempt to send a message in an inappropriate role will be rejected.
Notice that an application can be simultaneously both a client and a server.
See the ''asai_set_env( )'' on page 4-6 in Chapter 4, ''ASAI Library Functions'' for further information.

Return Value

Upon successful completion, the function returns a nonnegative value. If an error occurs, the function returns -1.

Errors

The type of error is indicated by the error value set in `asai_errno`:

`C_BADFD` — Invalid file descriptor supplied

`C_BADFLOW` — Cannot issue the request because of flow control

`C_CLUSTID` — The cluster identifier is invalid for the given stream

`C_BADLNG` — The send or receive buffer is too small for the capability

`C_BADMSG` — A corrupt message was received on the given stream

`C_INTR` — A system call was interrupted by a signal

`C_OSER` — A system call failed

`C_UNCAP` — Cannot send an unknown capability

`C_TOOBIG` — The field is too big

See Also

`asai_open()  asai_rcv()  asai_set_env()`

asai_set_env (3ASAI)

Name

asai_set_env — Set a specified characteristic for a communication path

Synopsis

```
#include  <asai/asai_def.h>
#include  <asai/asai_str.h>
#include  <asai/asai_err.h>

typedef union{
        server_type_t  server;
        char  node_id[C_NODSIZ];
}set_type;

long  asai_set_env(fd, characteristic, arg)
int  fd;
long  characteristic;
set_type  *arg;
```

Description

This function is used to set a specified characteristic of the communication path associated with fd. The parameter characteristic can assume any of the following values:

C_NODE_ID

The C_NODE_ID parameter must be used before any initiating capability is sent. Its purpose is to assign the destination node (ECS) for all new associations on the communication path. If any association is active on the communication path, the function fails.

The arg parameter points to a null-terminated string, specifying the node identifier. A node identifier is the name of the IPCI board or LAN Gateway link connected to ECS with which you wish to communicate. These are typically "signal01," "signal02," "signal03," or "signal04." The use of arg as a null pointer clears a destination. Only one destination can be set for a communication path at a time.

C_SERVER

The C_SERVER parameter sets the type of service requests to be accepted for a specified capability group from the specified node.

When you use this characteristic, previously registered servers are superseded by the new values.

If no service requests have been sent by ECS or if the application has not used asai_set_env to register for handling them, then no service requests are passed to the application. Use of arg as a null pointer clears any previously set characteristic server value.

arg, defined below, is a pointer to a structure of type server_type_t and supplies the server information.

```
typedef struct{
        long  num_node;
        service_spec_t  *buf;
}server_type_t;
```

Within this structure, num_node indicates the number of structures of the type service_spec_t to be registered with the ASAI library and buf is a pointer to an array of structures of the type service_type_t, defined as follows:

```
typedef struct{
        long  server_type;
        char  node_id[C_NODSIZ];
}service_spec_t;
```

Here server_type shows which service requests will be accepted and node_id shows where they will be accepted. Valid server types that can be put together with the option of selecting either one are:

C_RT_SER — Routing

C_MAINT_SER — Maintenance

The node_id selects the sources of the requests that will be serviced.

Return Value

Upon successful completion, the function returns 0. If an error occurs, the function returns -1.

Errors

The type of error is indicated by the error value set in asai_errno:

C_BADFD — Invalid file descriptor supplied

C_BADCHAR — Unknown or improper context for a characteristic

C_BADCHARVAL — Unknown characteristic value specified

C_SERVEX — Cluster service is being provided by another application

C_BADNODE — Node is not available

C_SYSER — ASAI service error

C_INTR — A system call was interrupted by a signal

C_ACTIVE — Stream has active cluster instance

C_LINKDOWN — ASAI link on MAPD taken out of service by the administrator

See Also

asai_open() asai_rcv() asai_send() asai_set_env()

ASAI Capability Manual pages

In these manual pages, each capability is described by some or all of the following sections:

Name

This section names and succinctly describes the capability.

Type

This section gives the capability type. Based on its use within an association, each capability can be classified as initiating, continuing or terminating. These classifications reflect the transition of the association through the states of idle, active and cluster terminating. All unused or uninitiated associations are said to be idle. The capability types are defined as follows:

Initiating	This type of capability starts an association that did not previously exist. A new `sao_id` (previously known as `cluster_id`) must be supplied. If an existing `sao_id` is supplied, an error will be returned. If ECS initiates this capability, it will create an odd `sao_id`. Consequently, CV/PC developers (especially, those who planned to register as routing and heartbeat servers) should use even numbers for `sao_id`.
Continuing	This type of capability assumes that the association already exists and continues to exist after the capability is sent or received. An existing `sao_id` must be supplied. If a new `sao_id` is supplied, an error will be returned.
Terminating	This type of capability assumes that the association already exists. However, the association will be terminated after this capability is processed. It is worth noting that this processing does not happen instantaneously and that the application may have to deal with messages for this association that were queued earlier and have been read after the terminating capability was sent.

In addition, this section also classifies each capability as Acknowledged or Unacknowledged. No additional requests can be made until the outstanding acknowledgement is received.

If a capability is used in an association state for which it has not been defined, the use of the capability is considered a protocol violation. A protocol violation in the case of an acknowledged capability results in a negative acknowledgment. A protocol violation in the case of an unacknowledged capability aborts the association.

Usage

This section shows the associated data structures used with the capability.

Description

This section describes the capability, including its associated parameters.

Return Value

This section gives the values returned by `asai_send` or `asai_rcv` when the capability is used in those functions.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This section is used to describe any exceptions, caveats or usage suggestions pertaining to the capability.

See Also

This section lists related capabilities and ASAI functions, as appropriate. In general, it describes all the other capabilities in the appropriate capability group and to `C_EN_REP`. The latter is included because Event Notification differs according to the release of ECS equipment in use.

C_3PAD (3ASAI)

Name

C_3PAD — Establish an auto-dial

Type

Continuing — Acknowledged

Usage

```
typedef struct{
        asai_common_t       asai_common;
        char                *called_num;
        user_user_t         uudata;
        long                ofacility;
        char                *ofac_ext;
        long                prio_call;
        long                ack_flag;
        char                pool[C_DATSZ];
}a3pad_info_t;
```

Description

The client sends this capability to request the call between a third party domain-controlled extension and another number. A structure of type a3pad_info_t contains the third party call-setup parameters.

The C_3PAD capability is a continuing capability (FACility messages), created by C_3PDC_REQ. C_3PDC_REQ is described as an initiating capability. ECS will return an error if C_3PAD is used as an initiating capability, or before the C_3PCD_CONF is sent.

asai_common

Within this structure, the programmer sets capability to C_3PAD and primitive_type to C_REQUEST. The programmer also sets sao_id (also known as cluster_id) to identify the domain-control association.

called_num

This parameter is a pointer to a null-terminated string that specifies the called number.

uudata

The purpose of the user_user information is to convey information between ISDN users. This information is not interpreted by ECS, but rather is carried transparently and delivered to the remote user. If the info field is IA5 characters (ASCII), the string is null-terminated.

The leng field is an integer value that indicates the number of octets of user data included in the request. This field is set to 0 if no user_user data is present. Currently, ECS will accept up to 32 bytes of data (leng = 32). If more than 32 bytes is specified, an error will be returned. ASAI supports a maximum user_user data length of 32 bytes, although user_user data generated from a PRI trunk can be up to 127 bytes in length. (DEFINITY ECS will truncate this to 32 bytes before forwarding it to ASAI link.)

The protocol field indicates the type of information and is restricted to C_UU_USER that indicates a user-specific protocol or C_UU_IA5 which indicates IA5 or ASCII characters.

The info field is a pointer to an ASCII string of no more that 32 characters in length.

ofacility

This optional parameter identifies the type of facility requested. It can be C_TRUNK_GROUP or C_NUSE_LONG, indicating a trunk group or facility not used. Note that the C_ACD_SPLIT value is not allowed since this indicates which trunk group should be used when dialing from the controlled station.

ofac_ext

This parameter is a pointer to a null-terminated string that identifies the trunk access code of the trunk group.

prio_call

This parameter is C_PRIOR_ON for a priority call or C_PRIOR_OFF for a nonpriority call.

ack_flag

This parameter allows the client to request acknowledgment of the auto-dial request. C_ACK_ON indicates that acknowledgment is expected; C_ACK_OFF indicates that acknowledgment is not expected. If the client sets ack_flag to C_ACK_OFF, an INITIATE event report instead of an acknowledgment will be sent.

A Third Party Domain Control association will be aborted if a Third Party Auto Dial request is issued for that association and the ack_flag parameter is set to C_ACK_OFF.

This problem will be experienced in the following situation:

> A station is monitored using the Third Party Domain Control capability. A Third Party Auto Dial capability is sent to the ECS to make a call from the monitored station to another party. The ack_flag parameter is set to `C_ACK_OFF` in the Third Party Auto Dial request to indicate that an acknowledgment to the C_3PAD is **NOT** expected from the ECS. If the adjunct issues a Third Party Clear Call for the Auto-Dial call, and receives the acknowledgment from the ECS for that request, the adjunct will then abort the Third Party Domain Control association due to inappropriate state changes.
>
> in order to avoid this problem, you must set the `ack_flag` parameter to `C_ACK_ON` for every `C_3PAD` request.

`pool`

The `pool` parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See the section ‘‘Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value upon success and -1 on failure. A return value of 0 indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_send()`

`C_3PDC_REQ C_3PDC_CONF C_3PAD C_3PAD_CONF C_EN_REP C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PRC C_3PRC_CONF C_3PSDS C_3PSDS_CONF`

C_3PAD_CONF (3ASAI)

Name

C_3PAD_CONF — Acknowledge an auto-dial request

Type

Continuing — Unacknowledged

Usage

```
typedef struct{
        asai_common_t      asai_common;
        call_id_t          call_id;
        party_id_t         party_id;
        ucid_t             ucid;         /* G3V6 */
        char               pool[C_DATSZ];
 }a3pad_ack_t;

 typedef struct{
        asai_common_t      asai_common;
        long               cause_value;
 }a3pad_nak_t;
```

Description

The client receives this capability as a positive or negative acknowledgment to a C_3PAD_REQ request. The C_3PAD_CONF capability is always continuing and will not be carried in a RELease COMPlete message.

asai_common

Capability is returned as C_3PAD_CONF and primitive_type is returned as C_POS_ACK for a positive acknowledgment and as C_NEG_ACK for a negative acknowledgement. The sao_id (also known as cluster_id) specified in the auto-dial request is also returned.

A positive acknowledgment confirms that the C_3PAD_REQ request has been received and understood. Subsequent event reports will indicate the progress of the call.

call_id

This parameter identifies the resulting call.

party_id

This parameter indicates the identifier of the originator.

`ucid`

This parameter identifies the UCID (Universal Call ID) assigned by ECS.

`cause_value`

This parameter can be any of the following:

```
C_INVALID_CRV
C_USER_BUSY
C_NOUSE_RESP
C_OUT_OF_SERV
C_NETCONJ
C_INVLDIE
C_PROTERR
C_RESUNAVL
```

See Table 8-1 on page 8-12 in Chapter 8 for further information on cause values.

`pool`

The `pool` parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See ‘‘Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. Return value of 0 does not indicate an error, but that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

```
C_3PANS C_3PDC_REQ C_3PDC_CONF C_3PAD C_3PAD_CONF C_EN_REP
C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD
C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PRC C_3PRC_CONF
```

C_3PANS (3ASAI)

Name

C_3PANS — Answer a call

Type

Continuing — Acknowledged

Usage

```
typedef struct{
        asai_common_t   asai_common;
        call_id_t       call_id;
        char            pool[C_DATSZ];
}a3pans_info_t;
```

Description

This capability allows the adjunct to request, on behalf of a station user, to “answer” a ringing, bridged or held call that is present at the station.

Answering a ringing, bridged or held call means to connect a call by:

- Forcing the station off-hook, if the user is on-hook
- Cutting through the call to the head or handset, if the user is off-hook

The effect is as if the station user selected the call appearance of the alerting, bridged or held call and then went off-hook.

asai_common

Within this structure, the programmer sets `capability` to `C_3PANS` and `primitive_type` to `C_REQUEST`. The programmer also sets `sao_id` (also known as `cluster_id`) to identify the association that controls the station, or call.

call_id

Indicates the alerting, bridged or held call to be connected at the controlled station.

pool

The `pool` parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See “Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_send()`

`C_3PANS_CONF C_3PDC_REQ C_3PDC_CONF C_3PAD C_3PAD_CONF C_EN_REP C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PRC C_3PRC_CONF`

C_3PANS_CONF (3ASAI)

Name

C_3PANS_CONF — Acknowledge an answer-call request

Type

Continuing — Unacknowledged

Usage

```
typedef struct{
        asai_common_t   asai_common;
        long            cause_value;
}a3pans_nak_t;
```

Description

The client receives this capability as a positive or negative acknowledgement to a C_3PANS request.

asai_common

Capability is returned as C_3PANS_CONF and primitive_type is returned as C_POS_ACK for a positive acknowledgement and as C_NEG_ACK for a negative acknowledgement. The sao_id (also known as cluster_id) specified in the answer call request is also returned.

A positive acknowledgment confirms that the C_3PANS request has been received and understood. Subsequent event reports will indicate the progress of the call. In the case of a positive acknowledgement, the structure of asai_common_t is the only one needed.

cause_value

This parameter can be any of the following:

```
C_INVALID_CRV
C_USER_BUSY
C_NOUSE_RESP
C_OUT_OF_SERV
C_NETCONJ
C_INVLDIE
C_PROTERR
C_RESUNAVL
```

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, but that no data has been received.

Errors

See Chapter 7, ''Error Messages'' for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

```
C_3PANS C_3PDC_REQ C_3PDC_CONF C_3PAD C_3PAD_CONF C_EN_REP
C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF
C_3PSD C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PRC C_3PRC_CONF
```

C_3PCC (3ASAI)

Name

C_3PCC — Third party request to clear a call

Type

Continuing — Acknowledged

Usage

```
asai_common_t  asai_common;
```

Description

The client sends this capability to request that a client-controlled call be cleared. Clearing disconnects all parties to the specified call. To receive the server's acknowledgement (C_3PCC_CONF), the client must use asai_rcv().

The asai_common structure contains the parameters needed to service this request.

Within asai_common, the programmer sets capability to C_3PCC and primitive_type to C_REQUEST. The programmer also sets sao_id (also known as cluster_id). The acknowledgement sent by the server will identify the call to be cleared by this sao_id (cluster_id).

Return Value

When this capability is used in an asai_send(), the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a client.

See Also

`asai_send()`

`C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PSD_CONF C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF`

C_3PCC_CONF (3ASAI)

Name

C_3PCC_CONF — Acknowledge a third party clear call request

Type

Terminating — Unacknowledged, when used as a positive acknowledgement

Continuing — Unacknowledged, when used as a negative acknowledgement

Usage

```
typedef struct{
        asai_common_t    asai_common;
        long             cause_value;
}a3pcc_rsp_t;
```

Description

The client receives this capability from the server in acknowledgement of a C_3PCC request. The structure a3pcc_rsp_t contains the parameters necessary to service this capability.

asai_common

Within this structure, capability is returned as C_3PCC_CONF; primitive_type is returned as C_POS_ACK for positive acknowledgement and C_NEG_ACK for negative acknowledgement. The sao_id (also known as cluster_id) specified in the C_3PCC request is also returned.

cause_value

This parameter can be any of the following:

C_CALLID_TERM
C_FACUNSUB
C_INVLDIE
C_MAND_INFO
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, but that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by the server.

See Also

`asai_rcv()`

`C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PSD_CONF C_3PCC C_3PCE C_3PRC C_3PRC_CONF`

C_3PCE (3ASAI)

Name

C_3PCE — End a third party call controlled by the client

Type

Terminating — Unacknowledged

Usage

```
typedef struct {
        asai_common_t       asai_common;
        long                cause_value;
        call_id_t           call_id;
        char                pool[C_DATSZ];
}a3pce_info_t;
```

Description

The server sends this capability to request termination of an existing call controlled by a client. This capability disconnects all parties to the specified call and terminates the association by freeing the sao_id (also known as cluster_id).

asai_common

Within this structure, capability is returned as C_3PCE and primitive_type is set to C_REQUEST. The sao_id (also known as cluster_id) of the association to be terminated is also returned.

cause_value

This parameter can be any of the following:

C_CALLID_TERM
C_FACUNSUB
C_INVLDIE
C_MAND_INFO
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL

See Table 8-1 on page 8-12 for further information on cause values.

`call_id`

Within this structure, `id_length` specifies the length of the ECS-assigned call identifier and `id_ptr` points to an array of binary bytes that specifies ECS-assigned call identifier of the call to be terminated.

`pool`

The `pool` parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See ‘‘Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a server.

See Also

`asai_send() asai_rcv()`

`C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PRC C_3PRC_CONF`

C_3PDC_CONF (3ASAI)

Name

C_3PDC_CONF — Acknowledge a third party domain control request

Type

Continuing — Unacknowledged, when used as a positive acknowledgement

Terminating — Unacknowledged, when used as a negative acknowledgement

Usage

```
typedef struct{
        asai_common_t       asai_common;
        long                num_callid;
        stn_info_t          *stninfo_list;
        char                pool[C_DATSZ];
}a3pdc_ack_t;

typedef struct{
        asai_common_t       asai_common;
        long                cause_value;
}a3pdc_nak_t;
```

Description

The client receives this capability as a positive or negative acknowledgement of a domain control request.

For positive acknowledgment, a structure of type a3pdc_ack_t is returned. if the domain_type has been set to C_ACD_SPLIT in the domain control request (C_3PDC_REQ), then this capability includes only the asai_common structure. Positive acknowledgment of the control of an ACD agent does not return these parameters.

If domain_type is set to C_EXTENSION in the domain control request, a3pdc_ack_t contains status information on the call.

asai_common

Within this structure, capability is returned as C_3PDC_CONF; primitive_type is returned as C_POS_ACK for a positive acknowledgement and as C_NEG_ACK for a negative acknowledgement. The sao_id (also known as cluster_id) specified in the domain control request is also returned.

`num_callid`

This parameter shows the number of structures and calls, listed in `stninfo_list`.

`stninfo_list`

See ‘‘Identifiers’’ on page 8-5 in Chapter 8, ‘‘ASAI Capability Primitives.’’

`pool`

The `pool` parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See “Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

`cause_value`

This parameter can be any of the following:

```
C_FACUNSUB
C_INVLDIE
C_INVLDNUM
C_MAND_INFO
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL
```

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, but that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

`C_EN_REP C_3PSH C_3PDC_REQ C_3PDC_CONF C_3PAD C_3PAD_CONF C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF`

C_3PDC_REQ (3ASAI)

Name

C_3PDC_REQ — Domain control of an ACD or station

Type

Initiating — Acknowledged

Usage

```
typedef struct{
        asai_common_t        asai_common;
        long                 domain_type;
        char                 *extension;
        char                 pool[C_DATSZ];
}a3pdc_info_t;
```

Description

The client sends this capability to request control of an extension domain. The parameters used in gaining control of the extension domain are defined in the structure of type a3pdc_info_t.

When a station domain control request is positively acknowledged by the ECS, all the third party call control capabilities can be applied to the controlled domain, except for the following:

C_3PTC_REQ/CONF, C_3PMC_REQ/CONF,C_3PCC/_CONF, C_3PCE.

If the request is denied by ECS, error and cause values are returned.

asai_common

Within this structure, the programmer sets capability to C_3PDC_REQ and primitive_type to C_REQUEST. The programmer also sets the sao_id (also known as cluster_id).

extension

The extension parameter is a pointer to a character array that specifies the domain to be controlled.

pool

The pool parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See “Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_send()`

`C_EN_REP C_3PSH C_3PDC_REQ C_3PDC_CONF C_3PAD C_3PAD_CONF C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PRC C_3PRC_CONF`

C_3PDCE (3ASAI)

Name

C_3PDCE — End the domain control

Type

Terminating — Unacknowledged

Usage

```
typedef struct {
       asai_common_t        asai_common;
       long                 cause_value;
}a3pdce_info_t;
```

Description

The server sends this capability to indicate that it will end a previous C_3PDC_REQ. This indication terminates the association and is generally provided when the affected domain is removed or changed administratively from ECS.

asai_common

Within this structure, the programmer sets capability to C_3PDCE and primitive_type to C_REQUEST. The programmer also sets the association identifier to the sao_id (also known as cluster_id) specified in the C_3PDC_REQ.

cause_value

This parameter can be any of the following:

C_FACUNSUB
C_INVLDIE
C_INVLDNUM
C_MAND_INFO
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL

See Table 8-1 on page 8-12 for further information on cause values.

`pool`

The `pool` parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See "Pool" on page 8-15 in Chapter 8, "ASAI Capability Primitives" for more information.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, "Error Messages" for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a server.

See Also

`asai_send()`

`C_EN_REP C_3PDC_REQ C_3PDC_CONF C_3PAD C_3PAD_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PRC C_3PRC_CONF`

C_3PM (3ASAI)

Name

C_3PM — Merge two calls

Type

Continuing — Acknowledged

Usage

```
typedef struct{
        asai_common_t       asai_common;
        party_id_t          com_party_id;
        call_id_t           call_id;
        call_id_t           held_call_id;
        long                conf_flag;
        char                pool[C_DATSZ];
}a3pm_info_t;
```

Description

The client sends this capability to request that two client-controlled calls that have a common party be merged. The common party can be retained on the call by conferencing the two calls or it can be disconnected after effectively transferring one of the other parties. A structure of type a3pm_info_t contains the parameters needed to service this capability.

asai_common

Within this structure, the programmer sets capability to C_3PM and primitive_type to C_REQUEST. The programmer sets sao_id (also known as cluster_id) to the association identifier of one of the calls.

com_party_id

The common party is a conditional parameter and is identified by a party_id structure. The com_party_id parameter must be used when the association has been started through C_3PMC_REQ or C_3PTC_REQ. When it is started with C_3PDC_REQ, the parameter is not used. See also party_id.

call_id

Specifies the call to be merged with the held call. The call identified by this parameter must be a call initiated by the C_3PMC_REQ capability or a call controlled by the C_3PDC_REQ capability.

held_call_id

Specifies the held call. The held call parameter is used only if the extension is controlled by C_3PDC_REQ.

conf_flag

When conf_flag is set to C_CONF_ON, the party identified by com_party_id remains connected to the merged call; when conf_flag is set to C_CONF_OFF, the common party is disconnected.

pool

The pool parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See ‘‘Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an asai_send(), the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

The use of this capability in asai_send() is restricted to clients.

See Also

asai_send() asai_rcv()

C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM_CONF C_3PSD C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF

C_3PM_CONF (3ASAI)

Name

C_3PM_CONF — Acknowledge a third party merge request

Type

Continuing — Unacknowledged

Usage

```
typedef struct{
        asai_common_t       asai_common;
        call_id_t           call_id;
        long                num_merge_ext;
        merge_ext_t         *party_list;
        ucid_t              ucid;           /* G3V6 */
        char                pool[C_DATSZ];
}a3pm_ack_t;

typedef struct{
        asai_common_t       asai_common;
        long                cause_value;
}a3pm_nak_t;
```

Description

The client receives this capability as a positive or negative acknowledgment of a C_3PM request. Structures of the type a3pm_ack_t and a3pm_nak_t contain the asai_common structure that specifies the parameters used in acknowledgments. Positive acknowledgment parameters are specified by the structure of type a3pm_ack_t. Negative acknowledgment parameters are specified by the structure of type a3pm_nak_t.

asai_common

Within this structure, capability is returned as C_3PM_CONF and primitive_type as C_POS_ACK for a positive acknowledgment and C_NEG_ACK for a negative acknowledgment. The sao_id (also known as cluster_id) specified in the C_3PM request is also returned.

call_id

Within a3pm_ack_t, call_id specifies ECS-assigned call identifier of the merged call.

num_merge_ext

This parameter specifies the number of parties connected to the merged call.

`party_list`

Within `a3pm_ack_t`, `party_list` is a pointer to a structure of the type `merge_ext_t`. Each element of this list, up to and including the element identified by `num_merge_ext`, contains information about the parties connected to the merged call. In each of these structures, `id_ptr` is a pointer to ECS-assigned party identifier; `id_length` indicates the length of the party identifier. `extension` points to a null-terminated character string, indicating the extension or station number associated with that party.

`ucid`

Within a `3pm_ack_t`, `this parameter` specifies the resulting ucid assigned by the ECS.

`pool`

The `pool` parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See “Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

`cause_value`

This parameter can be any of the following:

```
C_CALLID_TERM
C_FACUNSUB
C_INCOM_ST
C_INVALID_CRV
C_INVLDIE
C_INVLDNUM
C_MAND_INFO
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL
```

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, but that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

asai_rcv()

C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF
C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PSD C_3PSD_CONF
C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF

C_3PMC_CONF (3ASAI)

Name

C_3PMC_CONF — Acknowledge a third party make call request

Type

Continuing — Unacknowledged, when used as a positive acknowledgment

Terminating — Unacknowledged, when used as a negative acknowledgment

Usage

```
typedef struct{
        asai_common_t   asai_common;
        party_id_t      party_id;
        call_id_t       call_id;
        number_id_t     connected;
        ucid_t          ucid;         /* G3V6 */
        char            pool[C_DATSZ];
}a3pmc_ack_t;

typedef struct{
        asai_common_t   asai_common;
        long            cause_value;
}a3pmc_nak_t;
```

Description

The client receives this capability as a positive or negative acknowledgment of a C_3PMC_REQ request.

asai_common

Within this structure, capability is returned as C_3PMC_CONF, and primitive_type is returned as C_POS_ACK for a positive acknowledgment and as C_NEG_ACK for a negative acknowledgment. The sao_id (also known as cluster_id) specified in the C_3PMC request is also returned.

A positive acknowledgment confirms that the C_3PMC_REQ request has been received and understood. Subsequent event reports will indicate the progress of the call. For a positive acknowledgment, a3pmc_ack is used. For a negative acknowledgment, a3pmc_nak is used.

party_id

This parameter indicates the originating party on the call.

call_id

This parameter identifies the call that was created by the third party make call request.

connected

This parameter identifies the connected number which, along with other information, is contained in the number_id_t structure. The s field of the number_id_t structure, when not NULL, is an ASCII string containing the connected number. If s is NULL, the following field, type_plan, is undefined. The type_plan field is of type plan_type_t. For more information concerning plan_type_t, see the C_EN_REP: C_ALERTING description in this section. This field represents the calling party and is only present when the originator is an EAS agent.

ucid

This parameter identifies the ucid that was created by the third party make call request. This id is assigned by ECS.

pool

The pool parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See "Pool'' on page 8-15 in Chapter 8, ''ASAI Capability Primitives'' for more information.

cause_value

This parameter can be any of the following:

```
C_FACUNSUB
C_INCOM_OPT
C_INVLDIE
C_INVLDNUM
C_MAND_INFO
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL
```

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an asai_rcv(), the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, but that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

asai_rcv()

C_EN_REP C_3PMC_REQ C_3PTC_REQ C_3PTC_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF

C_3PMC_REQ (3ASAI)

Name

C_3PMC_REQ — Make a third party call

Type

Initiating — Acknowledged or unacknowledged

Usage

```
typedef struct{
        asai_common_t       asai_common;
        char                *calling_num;
        char                *called_num;
        user_user_t         uudata;
        long                ofacility;
        char                *ofac_ext;
        long                prio_call;
        long                alert_time;
        long                serv_cir;
        long                dir_agtcall;
        long                super_ast;
        long                alt_dest;
        long                ack_flag;
        long                ans_mach_treat;
        char                pool[C_DATSZ];
}a3pmc_info_t;
```

Description

The client sends this capability to request establishment of a third party call. A structure of type a3pmc_info_t contains the third party call-setup parameters.

asai_common

Within this structure, the programmer sets capability to C_3PMC_REQ and primitive_type to C_REQUEST. The programmer also sets the controlling association to sao_id (also known as cluster_id).

calling_num

The calling_num, a mandatory parameter, is a pointer to a null-terminated character string that specifies the calling extension.

called_num

The mandatory parameter called_num is a pointer to a null-terminated character string that specifies the called extension.

uudata

The purpose of the uudata is to convey information between ISDN users. This information is not interpreted by ECS, but rather is carried transparently and delivered to the remote user. If the info field is IA5 characters (ASCII), the string is null-terminated.

The leng field is an integer value that indicates the number of octets of user data included in the request.This field is set to 0 if no user-to-user data is present. Currently, ECS will accept up to 32 bytes of data (leng = 32). If more than 32 bytes is specified, an error will be returned. ASAI supports a maximum user-to-user data length of 32 bytes, although user-to-user data generated from a PRI trunk can be up to 127 bytes in length. (DEFINITY ECS will truncate this to 32 bytes before forwarding it to ASAI link.)

The protocol field indicates the type of information and is restricted to C_UU_USER that indicates a user-specific protocol or C_UU_IA5 which indicates IA5 or ASCII characters.

The info field is a pointer to an ASCII string of no more that 32 characters in length.

ofacility

This is an optional parameter that identifies the type of facility requested. It can be C_TRUNK_GROUP, C_ACD_SPLIT or C_NUSE_LONG. These values indicate a trunk group, an ACD group or facility not used.

ofac_ext

The ofac_ext parameter is a pointer to a null-terminated character string that identifies the trunk access code of the trunk group or the ACD split extension, depending on the value of ofacility.

prio_call

The requester can indicate call options to be used for the call. The prio_call parameter can take the value of C_PRIOR_ON (indicating that the call is a priority call) or C_PRIOR_OFF (indicating that the call is not a priority call).

alert_time

The alert_time parameter indicates the maximum number of rings to wait after the call has reached the alerting state.

serv_cir

The serv_cir parameter with a value of C_SERV_ON indicates that a call classifier is to be added to the call request. The default is C_SERV_OFF.

dir_agtcall

A value of C_DIRAGT_ON for dir_agtcall indicates that the call is directed to a specific ACD agent. When dir_agtcall is selected, ofac_ext contains the identifier of the ACD split containing the agent, and ofacility should be set to C_ACD_SPLIT.

super_ast

A C_SUP_AS_ON value for super_ast indicates that the call is an ACD call, directed to an ACD supervisor by an agent who needs assistance; otherwise, this parameter assumes a value of C_SUP_AS_OFF when super_ast is selected. When super_ast is selected, ofac_ext contains the identifier of the ACD split containing the agent, and ofacility should be set to C_ACD_SPLIT.

alt_dest

The alt_dest parameter can assume the values of C_ALERT_ON (indicating that the destination is to be alerted first) or C_ALERT_OFF.

ack_flag

The ack_flag parameter allows the client to request acknowledgement of the make call request. A value of C_ACK_ON indicates that acknowledgement is expected; C_ACK_OFF indicates that acknowledgement is not expected.

ans_mach_treat

If ans_mach_treat is CO_AM_DISC, the call is disconnected upon answering machine detection. If the treatment is C_CO_AM_SWITCH, it follows the setting in ECS administration. If the treatment is C_CO_AM_CONNECT, the call is disconnected.

This parameter must be coded -1 if unused.

pool

The pool parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See ‘‘Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an asai_send(), the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

asai_send()

C_EN_REP C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH
C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD
C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF

C_3PR (3ASAI)

Name

C_3PR — Reconnect a held party

Type

Continuing — Acknowledged

Usage

```
typedef struct{
        asai_common_t       asai_common;
        party_id_t          party_id;
        call_id_t           call_id;
        char                pool[C_DATSZ];
}a3pr_info_t;
```

Description

The client sends this capability to request reconnection of a held party to a call controlled by the client. The structure of type `a3pr_info_t` contains the parameters needed to fulfill the request.

asai_common

The programmer sets `capability` to `C_3PR` and `primitive_type` to `C_REQUEST`. The `sao_id` (also known as `cluster_id`) identifies the association that controls the call to be reconnected. The programmer also sets the `sao_id` (`cluster_id`) to the association identifier used to put the party on hold.

party_id

Specifies the party to be reconnected. The length of `id_ptr` is held in `id_length`, and is always 1. `party_id` is used only when the association is initiated by the `C_3PTC_REQ` or the `C_3PMC_REQ` capability.

`party_id` is *not* an ASCII null-terminated string but rather an array of binary bytes of the specified length. See ‘‘Identifiers’’ on page 8-5 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

call_id

`call_id` is ECS-assigned identifier of the call to reconnect. The value of this parameter is a character array of length `id_length`, pointed to by `id_ptr`. `call_id` is used only when the association is initiated by the `C_3PDC_REQ` capability.

`pool`

The `pool` parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See “Pool” on page 8-15 in Chapter 8, “ASAI Capability Primitives” for more information.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, “Error Messages” for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a client.

See Also

`asai_send()`

```
C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF
C_3PSH C_3PSH_CONF C_3PR_CONF C_3PM C_3PM_CONF C_3PSD
C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF
```

C_3PR_CONF (3ASAI)

Name

C_3PR_CONF — Acknowledge a third party reconnect request

Type

Continuing — Unacknowledged

Usage

```
typedef struct{
        asai_common_t        asai_common;
        long                 cause_value;
}a3pr_rsp_t;
```

Description

The client receives this capability as a positive or negative acknowledgement to a C_3PR request.

asai_common

Within asai_common, capability is returned as C_3PR_CONF, primitive_type is returned as C_POS_ACK for positive acknowledgement and as C_NEG_ACK for negative acknowledgement. The sao_id (also known as cluster_id) specified in the C_3PR request is also returned.

cause_value

This parameter can be any of the following:

C_CALLID_TERM
C_FACUNSUB
C_INCOM_ST
C_INVLDIE
C_INVLDNUM
C_MAND_INFO
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, but that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

```
C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF
C_3PSH C_3PSH_CONF C_3PR C_3PM C_3PM_CONF C_3PSD C_3PSD_CONF
C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF
```

C_3PRC (3ASAI)

Name

C_3PRC — Relinquish control of a call without disconnecting the parties to the call

Type

Terminating — Acknowledged

Usage

```
asai_common_t   asai_common;
```

Description

The client sends this capability to request that control of a client-controlled call be relinquished. This terminates the association. The `asai_common` structure contains the parameters needed to fulfill the request.

The programmer sets `capability` to `C_3PRC` and `primitive_type` to `C_REQUEST`. The programmer also sets the `sao_id` (also known as `cluster_id`). The use of this capability does not disconnect the parties to the call or free the resources necessary to maintain the call connection.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value on success and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a client.

See Also

asai_send()

C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF
C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD
C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PCE C_3PRC_CONF

C_3PRC_CONF (3ASAI)

Name

C_3PRC_CONF — Acknowledge a third party relinquish control request

Type

Terminating — Unacknowledged, when used as a positive acknowledgement

Continuing — Unacknowledged, when used as a negative acknowledgement

Usage

```
typedef struct{
        asai_common_t        asai_common;
        long                 cause_value;
}a3prc_nak_t;

typedef union{
        asai_common_t        asai_common;
        a3prc_nak_t          a3prc_nak;
}a3prc_rsp_t;
```

Description

The client receives this capability as a positive or negative acknowledgement of a C_3PRC request, specified by a3prc_rsp_t.

asai_common

Within asai_common, capability is returned as C_3PRC_CONF and primitive_type is returned as C_POS_ACK for positive acknowledgement and as C_NEG_ACK for negative acknowledgement. The sao_id (also known as cluster_id) specified in the C_3PRC request is also returned.

cause_value

cause_value can be any of the following values:

C_FACUNSUB
C_INVLDIE
C_MAND_INFO
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, but that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

`C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PCE C_3PRC`

C_3PREDIR (3ASAI)

Name

C_3PREDIR — Redirects an alerting call

Type

Continuing — Acknowledged

Usage

```
typedef struct{
        asai_common_t   asai_common;
        call_id_t       call_id;
        number_id_t     redir_num;
        party_id_t      party_id;
        char            pool[C_DATSZ];
} a3predir_info_t;
```

Description

The client sends this capability to request that an alerting call be redirected to another number. A structure of type `a3predir_info_t` contains the parameters needed to fulfill the request.

asai_common

The programmer sets `capability` to `C_3PREDIR` and `primitive_type` to `C_REQUEST`. The programmer also sets the `sao_id` (also known as `cluster_id`) to identify the association that controls the alerting call.

call_id

`call_id` is required for domain control associations and is ignored with call control associations. Two fields, `id_length` and `id_ptr`, are in a call id. If `id_length` is zero, the field is omitted; otherwise, `id_ptr` contains a binary representation for the `call_id` (typically, received in an earlier ASAI message).

redir_num

The required parameter `redir_num` identifies the destination of the redirected call. Two fields, `plan_type` and `s`, are in a number id. *s* is a null-terminated string containing the destination number. `plan_type` is described in ‘‘Identifiers’’ on page 8-5 of Chapter 8.

`party_id`

`party_id` is required for call control associations. It specifies the `redirected_from` party on the call.

The length of the party identifier is `id_length`. The character array pointed to by `id_ptr` contains the `party_id.` See ‘‘Identifiers’’ on page 8-5 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

`Pool`

The pool parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See the section ‘‘Pool’’ in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that there is no data to be sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a client.

See Also

`asai_send()`

```
C_3PREDIR_ACK C_3PSL_RECONN_ACK C_EN_REP C_3PMC_REQ
C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH C_3PSH_CONF
C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD_CONF C_3PCC
C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF
```

C_3PREDIR_ACK (3ASAI)

Name

C_3PREDIR_ACK — Acknowledge a third party redirect alerting call request

Type

Continuing — Unacknowledged

Usage

```
typedef struct{
        asai_common_t        asai_common;
        long                 cause_value;
}a3predir_nak_t;

typedef union{
        asai_common_t        asai_common;
        a3psl_disc_nak_t     a3psl_disc_nak;
} a3predir_ack_t;
```

Description

The client receives this capability as a positive or negative acknowledgement of a C_3PREDIR request.

asai_common

Within this structure, capability is returned as C_3PREDIR_ACK; primitive_type is returned as C_POS_ACK for positive acknowledgement and as C_NEG_ACK for negative acknowledgement.

The sao_id (also known as cluster_id) specified in the C_3PREDIR request is also returned.

cause_value

This parameter can be any of the following:

C_USER_BUSY
C_INVLDNUM
C_PROTERR
C_RESUNAVL
C_MAND_INFO
C_INVLDIE
C_INCOM_ST
C_NETCONJ
C_INVALID_CRV

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, but that no data has been received.

Errors

See Chapter 7, ''Error Messages'' for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

`C_3PPREDIR C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF`

C_3PSD (3ASAI)

Name

C_3PSD — Drop selected parties from a client-controlled call

Type

Continuing — Acknowledged

Usage

```
typedef struct{
        asai_common_t      asai_common;
        party_id_t         party_id;
        call_id_t          call_id;
        user_user_t        uudata;
        long               resource;
        char               pool[C_DATSZ];
}a3psd_info_t;
```

Description

The client sends this capability to request that a party to a call controlled by the client be dropped from the call. Any tone connected to a call, except ringback tones, may be dropped. A structure of type `a3psd_info_t` contains the parameters needed to fulfill the request.

asai_common

The programmer sets `capability` to `C_3PSD` and `primitive_type` to `C_REQUEST`. The programmer also sets the `sao_id` (also known as `cluster_id`) to identify the association that controls the call to be dropped.

party_id

`party_id` is a conditional parameter and specifies the party to be dropped. The length of the identifier is `id_length`. `party_id` must be used when the association is initiated by the `C_3PMC_REQ` or `C_3PTC_REQ` capabilities and the resource identifier is not specified.

NOTE:
`party_id` and `call_id` are not ASCII null-terminated strings but rather an array of binary bytes of the specified length. See ''Identifiers'' on page 8-5 in Chapter 8, "ASAI Capability Primitives'' for more information.

call_id

call_id is ECS-assigned identifier of the call to be dropped. call_id is used only when the association is initiated by the C_3PDC_REQ capability.

NOTE:
party_id and call_id are not ASCII null-terminated strings but rather an array of binary bytes of the specified length. See ''Identifiers'' on page 8-5 in Chapter 8, ''ASAI Capability Primitives'' for more information.

uudata

The purpose of the user-user information is to convey information between ISDN users. This information is not interpreted by ECS, but rather is carried transparently and delivered to the remote user. If the info field is IA5 characters (ASCII), the string is null-terminated.

The leng field is an integer value that indicates the number of octets of user data included in the request.This field is set to 0 if no user_user data is present. Currently, ECS will accept up to 32 bytes of data (leng = 32). If more than 32 bytes is specified, an error will be returned. ASAI supports a maximum user_user data length of 32 bytes, although user_user data generated from a PRI trunk can be up to 127 bytes in length. (DEFINITY ECS will truncate this to 32 bytes before forwarding it to ASAI link.)

The protocol field indicates the type of information and is restricted to C_UU_USER which indicates a user-specific protocol or C_UU_IA5 which indicates IA5 or ASCII characters.

The info field is a pointer to an ASCII string of no more that 32 characters in length.

resource

The resource field specifies the resource to be dropped from the call. The available resources are C_CALL_CLASSIFIER and C_TONE_GENERATOR. The "tone generator" is defined in this case as being any locally applied denial tone that is timed by ECS. The resource field may be included in the C_3PSD capability when the association is initiated by the either C_3PMC_REQ or C_3PTC_REQ capability and when the party_id is not included. The field is zero if no resources are to be dropped.

pool

The pool parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See "Pool'' on page 8-15 in Chapter 8, ''ASAI Capability Primitives'' for more information.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a client.

See Also

`asai_send()`

```
C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF
C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF
C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF
```

C_3PSD_CONF (3ASAI)

Name

C_3PSD_CONF — Acknowledge a third party selective drop request

Type

Continuing — Unacknowledged

Usage

```
typedef struct{
        asai_common_t        asai_common;
        long                 cause_value;
}a3psd_rsp_t;
```

Description

The client receives this capability as a positive or negative acknowledgement of a C_3PSD request.

asai_common

Within this structure, capability is returned as C_3PSD_CONF; primitive_type is returned as C_POS_ACK for positive acknowledgement and as C_NEG_ACK for negative acknowledgement. The sao_id (also known as cluster_id) specified in the C_3PSD request is also returned.

cause_value

This parameter can be any of the following:

C_INVLDNUM
C_PROTERR
C_RESUNAVL
C_FACUNSUB
C_SER_UNIMP
C_MAND_INFO
C_INVLDIE
C_SERV_UNAVIL
C_CALLID_TERM

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, but that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

`C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF`

C_3PSDS (3ASAI)

Name

C_3PSDS — Send DTMF Signals on behalf of a party on a call

Type

Continuing — Acknowledged

Usage

```
typedef struct{
   asai_common_t   asai_common;
   call_id_t       call_id;
   party_id_t      tx_party_id;
   party_id_t      tx_party_ids[5];
   user_code_t     user_data;
   long            tone_duration;
   long            pause_duration;
   char            pool [C_DATSZ];
} a3psds_info_t;
```

Description

The client sends this capability to request that DTMF tones be sent to all parties on a call that are connected to ECS via ports that support end-to-end signaling and that have active listen paths to the sender. A structure of type a3psds_info_t contains the parameters needed to fulfill the request.

asai_common

The programmer sets capability to C_3PSDS and primitive_type to C_REQUEST. The programmer also sets the sao_id (also known as cluster_id) to identify the association that controls the call to apply DTMF signals.

tx_party_id

tx_party_id is a conditional parameter and specifies the party on whose transmit path the DTMF signals will be applied. The length of the identifier is id_length. tx_party_id must be used when the association is initiated by the C_3PMC_REQ and C_3PTC_REQ capabilities. Otherwise, the parameter is not used. tx_party_id is not an ASCII null-terminated string but rather an array of binary bytes of the specified length. See ‘‘Identifiers’’ on page 8-5” in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

call_id

call_id is ECS-assigned identifier of the active call on which the DTMF signals will be applied. call_id is used only when the association is initiated by the C_3PDC_REQ capability.

user_data

The user_data structure is used to convey the information about the DTMF digits to ECS.

The leng field is an integer value that indicates the number of octets (digits) of user data included in the request.

The type field indicates the type of information and is restricted to C_UU_IA5, which indicates IA5 or ASCII characters.

The digits field is a pointer to an ASCII string of no more that 32 characters in length.

tone_duration

The tone_duration is an optional parameter that specifies in .01 second increments, the duration of each tone. The default duration is .35 seconds (350 milliseconds). Valid values are from .06 seconds to .35 seconds.

pause_duration

The pause_duration is an optional parameter that specifies in .01 second increments, the duration of the pause in between tones. The default duration is .10 seconds (100 milliseconds). Valid values are from .04 seconds to .10 seconds.

pool

The pool parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See “Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an asai_send(), the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a client.

See Also

```
asai_send()
```

```
C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH
C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD_CONF
C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF
```

C_3PSDS_CONF (3ASAI)

Name

C_3PSDS_CONF — Acknowledge a send DTMF request

Type

Continuing — Unacknowledged

Usage

```
typedef struct{
        asai_common_t        asai_common;
        long                 cause_value;
}a3psds_rsp_t;
```

Description

The client receives this capability as a positive or negative acknowledgement of a C_3PSDS request.

asai_common

Within this structure, capability is returned as C_3PSDS_CONF, and primitive_type is returned as C_POS_ACK for positive acknowledgement and as C_NEG_ACK for negative acknowledgement. The sao_id (also known as cluster_id) is specified in the C_3PSDS request is also returned.

cause_value

This parameter can be any of the following:

```
C_INVLDNUM
C_PROTERR
C_RESUNAVL
C_FACUNSUB
C_SER_UNIMP
C_MAND_INFO
C_INVLDIE
C_SERV_UNAVIL
C_CALLID_TERM
```

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

`C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF`

C_3PSH (3ASAI)

Name

C_3PSH — Place selected party on hold

Type

Continuing — Acknowledged

Usage

```
typedef struct{
        asai_common_t       asai_common;
        party_id_t          party_id;
        call_id_t           call_id;
        char                pool[C_DATSZ];
}a3psh_info_t;
```

Description

The client sends this capability to request that a party be placed on hold. The party must be on a call controlled by the client. A structure of type a3psh_info_t contains the parameters needed to fulfill this request.

asai_common

The programmer sets capability to C_3PSH and primitive_type to C_REQUEST. The programmer also sets the sao_id (also known as cluster_id) to identify the association that controls the call to be held.

party_id

For call control associations, that is, those initiated with C_3PTC_REQ or C_3PMC_REQ, party_id specifies the party to be placed on hold. For domain-controlled associations the parameter is not used.

The party_id structure contains two fields, id_ptr and id_length. The field, id_ptr, does not point to a null-terminated character string but rather an array of binary bytes of the specified length.

call_id

call_id is ECS-assigned identifier of the call to be placed on hold. call_id is used only when the association is initiated by the C_3PDC_REQ capability.

`pool`

The `pool` parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See ‘‘Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a client.

See Also

`asai_send()`

`C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF`

C_3PSH_CONF (3ASAI)

Name

C_3PSH_CONF — Acknowledge a third party selective hold request

Type

Continuing — Unacknowledged

Usage

```
typedef struct{
        asai_common_t        asai_common;
        long                 cause_value;
}a3psh_rsp_t;
```

Description

The client receives this capability as a positive or negative acknowledgement of a C_3PSH request.

asai_common

Within this structure, capability is returned as C_3PSH_CONF, and primitive_type is returned as C_POS_ACK for a positive acknowledgement and as C_NEG_ACK for a negative acknowledgement. The sao_id (also known as cluster_id) specified in the selective hold request is also returned.

cause_value

This parameter can be any of the following:

```
C_CALLID_TERM
C_FACUNSUB
C_INCOM_ST
C_INVLDIE
C_INVLDNUM
C_MAND_INFO
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL
```

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message upon success and -1 on failure. A return value of 0 does not indicate an error, but that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

`C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF`
`C_3PSH C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PSD_CONF`
`C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF`

C_3PSL_DISC (3ASAI)

Name

C_3PSL_DISC — Disconnect the listen path for selected parties

Type

Continuing — Acknowledged

Usage

```
typedef struct {
    asai_common_t   asai_common;
    party_id_t      from_party_id;
    party_id_t      to_party_id;
} a3psl_disc_t;
```

Description

The client sends this capability to request that listen path(s) between parties on a controlled call be dropped. A structure of type `a3psl_disc_t` contains the parameters needed to fulfill the request.

asai_common

The programmer sets `capability` to `C_3PSL_DISC` and `primitive_type` to `C_REQUEST`. The programmer also sets the `sao_id` (also known as `cluster_id`) to identify the association that controls the call to be disconnected.

from_party_id, to_party_id

`from_party_id` is a required parameter and specifies the party to be disconnected from listening. `to_party_id` is an optional parameter. When the `to_party_id` is specified, the only path that will be removed is between the "talk" and "listen" parties. Otherwise, all listening paths to `from_party_id` are disconnected. For each party id that is specified, the length of the party identifier is `id_length`. The character array pointed to by `id_ptr` contains the party id. See ''Identifiers'' on page 8-5 in Chapter 8, ''ASAI Capability Primitives'' for more information.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value upon success and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ''Error Messages'' for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a client.

See Also

`asai_send()`

`C_3PSL_RECONN C_3PSL_DISC_ACK C_3PSL_RECONN_ACK C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF`

C_3PSL_DISC_ACK (3ASAI)

Name

C_3PSL_DISC_ACK — Acknowledge a third party selective listen disconnect

Type

Continuing — Unacknowledged

Usage

```
typedef struct {
      asai_common_t        asai_common;
      long                 cause_value;
}a3psl_disc_nak_t;

typedef union{
      asai_common_t        asai_common;
      a3psl_disc_nak_t     a3psl_disc_nak;
} a3psl_disc_ack_t;
```

Description

The client receives this capability as a positive or negative acknowledgement of a C_3PSL_DISC request.

asai_common

Within this structure, capability is returned as C_3PSL_DISC_ACK; primitive_type is returned as C_POS_ACK for positive acknowledgement and as C_NEG_ACK for negative acknowledgement. The sao_id (also known as cluster_id) specified in the C_3PSL_DISC request is also returned.

cause_value

This parameter can be any of the following:

C_INVLDNUM
C_PROTERR
C_RESUNAVL
C_MAND_INFO
C_INVLDI
C_INCOM_ST

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

`C_3PSL_DISC C_3PSL_RECONN C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF`

C_3PSL_RECONN (3ASAI)

Name

C_3PSL_RECONN — Reconnect the listen path for selected parties

Type

Continuing — Acknowledged

Usage

```
typedef struct {
      asai_common_t  asai_common;
      party_id_t      from_party_id;
      party_id_t      to_party_id;
} a3psl_reconn_t;
```

Description

The client sends this capability to request that a listen path between parties on a controlled call be reconnected. A structure of type a3psl_reconn_t contains the parameters needed to fulfill the request.

asai_common

The programmer sets capability to C_3PSL_RECONN and primitive_type to C_REQUEST. The programmer also sets the sao_id (also known as cluster_id) to identify the association that controls the call to be reconnected.

from_party_id, to_party_id

from_party_id is a required parameter and specifies the party listening. to_party_id is an optional parameter. When the to_party_id is specified, the only path that will be reconnected is between the “talk” and “listen” parties. Otherwise, all parties who were disconnected from_party_id are reconnected. For party identifiers, the length of the identifier is id_length. The binary array pointed to by id_ptr contains the party id. See the ‘‘Identifiers’’ section in Chapter 8, “ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an asai_send(), the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a client.

See Also

`asai_send()`

`C_3PSL_DISC C_3PSL_DISC_ACK C_3PSL_RECONN_ACK C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF`

C_3PSL_RECONN_ACK (3ASAI)

Name

C_3PSL_RECONN_ACK — Acknowledge a third party selective listen reconnect

Type

Continuing — Unacknowledged

Usage

```
typedef struct{
      asai_common_t        asai_common;
      long                 cause_value;
}a3psl_reconn_nak_t;

typedef union{
      asai_common_t        asai_common;
      a3psl_reconn_nak_t   a3psl_reconn_nak;
} a3psl_reconn_ack_t;
```

Description

The client receives this capability as a positive or negative acknowledgement of a C_3PSL_RECONN request.

asai_common

Within this structure, capability is returned as C_3PSL_RECONN_ACK; primitive_type is returned as C_POS_ACK for positive acknowledgement and as C_NEG_ACK for negative acknowledgement. The sao_id (also known as cluster_id) specified in the C_3PSL_RECONN request is also returned.

cause_value

This parameter can be any of the following:

C_INVLDNUM
C_PROTERR
C_RESUNAVL
C_MAND_INFO
C_INVLDIE
C_INCOM_ST

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

```
C_3PSL_DISC C_3PSL_RECONN C_3PSL_RECONN_ACK C_EN_REP
C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH
C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD
C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF
```

C_3PSSC_REQ (3ASAI)

Name

C_3PSSC_REQ — Single-Step Conference — adds a station to an existing call

Type

Continuing — Acknowledged

Usage

```
typedef struct {
      asai_common_t  asai_common;
      char           *station_ext;
      call_id_t      call_id;
      long           alt_dst;
      long           visibility;
      char           pool [C_DATSZ];
} a3pssc_info_t;
```

Description

The client sends this capability to request that a station be conferenced into an existing call. A structure of type a3pssc_info_t contains the parameters needed to fulfill the request.

asai_common

The programmer sets the capability to C_3PSSC_REQ and primitive_type to C_REQUEST. The programmer also sets the sao_id (also known as cluster_id) to identify the association that controls the call to be conferenced.

station_ext

station_ext is a pointer to a null-terminated string that specifies the station extension that is to be conferenced. station_ext is used only when the association is initiated by the C_3PTC_REQ or the C_3PMC_REQ capability.

call_id

Call_id is the ECS-assigned identifier of the existing call into which the station is to be conferenced. Call_id is used only when the association is initiated by the C_3PDC_REQ capability. See the ‘‘Identifiers’’ section in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

`alt_dest`

For this capability, the value `C_ALERT_OFF` will always be used by the ECS in G3V6 release. `C_ALERT_OFF` corresponds to "add station without alerting" (0x80).

`visibility`

The `visibility` parameter can assume either the value `C_VISIBLE_ON` to indicate that the added party should be fully visible to the other parties on the call, or `C_VISIBLE_OFF` to indicate that the added party should not be visible to the other parties on the call.

`pool`

The `pool` parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See "Pool" on page 8-15 in Chapter 8, ''ASAI Capability Primitives'' for more information.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ''Error Messages'' for a list of error messages with brief explanations of their probable causes. If the association specified is under Call Control and the `station_ext` is null, the API will return `C_BADMATCH` and the request will be aborted. If the association is under Domain Control and the `call _id` length is 0, the API will return `C_BADMATCH` and abort the request.

Notes

This capability can be issued only by a client.

See Also

`asai_send()`

`C_3PSL_DISC C_3PSL_DISC_ACK C_3PSL_RECONN_ACK C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF C_3PSSC_CONF`

C_3PSSC_CONF (3ASAI)

Name

C_3PSSC_CONF— Acknowledge a third party Single-Step Conference

Type

Continuing — Unacknowledged

Usage

```
typedef struct{
      party_id_t           party_id;
      char                 *extension;
      plan_type_t          ext_type;
} ssconf_ext_t;

typedef struct{
      asai_common_t        asai_common;
      call_id_t            call_id;
      long                 num_conf_ext;
      ssconf_ext_t         *party_list;
      ucid_t               ucid;            /* new in R6 */
      char                 pool [C_DATSZ];
}a3pssc_ack_t;

typedef struct{
      asai_common_t        asai_common;
      long                 cause_value;
} a3pssc_nak_t;
```

Description

The client receives this capability as a positive or negative acknowledgement of a C_3PSSC_REQ request.

asai_common

Within this structure, capability is returned as C_3PSSC_CONF; primitive_type is returned as C_POS_ACK for positive acknowledgement with an array of parties on the call. Primitive_type is returned as C_NEG_ACK for negative acknowledgement with the cause value. The sao_id (also known as cluster_id) specified in the C_3PSSC_CONF request is also returned.

call_id

Within a3pssc_ack_t, call_id specifies the ECS-assigned call identifier of the call.

num_conf_ext

This parameter specifies the number of parties connected to the call.

party_list

This parameter, party_list is a pointer to a structure of type conf_ext_t. Each element of this list, up to and including the element identified by num_conf_ext contains information about the parties connected to the call. In each of these party_id structures, id_ptr is a pointer to the ECS-assigned party identifier. id_length indicates the length of the party identifier, extension points to a null-terminated character string indicating the extension or station number associated with that party; ext_type contains information about the numbering plan of the extension number.

ucid

During a positive acknowledgment, a structure of type a3pssc_ack_t contains information regarding the UCID (Universal Call ID) assigned by the ECS.

pool

The pool parameter is a character array that provides the buffer space for the ASAI library to store program-specific information.

cause_value

This parameter can be any of the following:

```
C_INVLDNUM
C_SERV_UNAVAIL
C_REORDER_DENIAL
C_INCOM_ST
C_INVALID_CRV
C_USER_BUSY
C_NUSE_RESP
C_MAND_INFO
C_PERM_DENIED
C_PROTERR
```

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

`C_3PSL_DISC C_3PSL_RECONN C_3PSL_RECONN_ACK C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PTC_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF C_3PSSC_CONF`

C_3PTC_CONF (3ASAI)

Name

C_3PTC_CONF — Acknowledge a third party take control request

Type

Continuing — Unacknowledged, when received as a positive acknowledgement

Terminating — Unacknowledged, when received as a negative acknowledgement

Usage

```
typedef struct{
        asai_common_t       asai_common;
        long                num_party_ext;
        party_ext_t        *party_ext_list;
        ucid_t              ucid;               /*G3V6 */
        char                pool[C_DATSZ];
}a3ptc_ack_t;

typedef struct{
        asai_common_t       asai_common;
        long                cause_value;
}a3ptc_nak_t;
```

Description

The client receives this capability as a positive or negative acknowledgement of a C_3PTC_REQ request. Positive acknowledgement parameters are specified by a3ptc_ack_t and negative acknowledgement parameters by a3ptc_nak_t.

asai_common

Within this structure, capability is returned as C_3PTC_CONF; primitive_type is returned as it is set to C_POS_ACK for positive acknowledgement and to C_NEG_ACK for negative acknowledgement. The sao_id (also known as cluster_id) specified in the C_3PTC request is also returned.

party_ext_list,num_party_ext

This parameter points to a list of structures. During a positive acknowledgment, a structure of type a3ptc_ack_t contains information concerning the existing parties to the call. The number of structures in the list is given by num_party_ext. Within party_ext_list, party_id specifies a party to the call. The extension parameter is a pointer to a null-terminated string that indicates the extension of the party. See ‘‘Identifiers’’ on page 8-5 in Chapter 8, “ASAI Capability Primitives’’ for a description of `party_id`.

`ucid`

During a positive acknowledgment, a structure of type `a3ptc_ack_t` contains information regarding the UCID (Universal Call ID) assigned by ECS.

`pool`

The `pool` parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See “Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

`cause_value`

This parameter can be any of the following:

```
C_CALLID_TERM
C_FACUNSUB
C_INVLDIE
C_MAND_INFO
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVAIL
```

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error but, that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

```
C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_REQ C_3PSH
C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD
C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF
```

C_3PTC_REQ (3ASAI)

Name

C_3PTC_REQ — Take control of an existing third party call

Type

Initiating — Acknowledged

Usage

```
typedef struct{
        asai_common_t       asai_common;
        call_id_t           call_id;
        char                pool[C_DATSZ];
}a3ptc_info_t;
```

Description

The client sends this capability to request control of an existing call. The `a3ptc_info_t` structure contains the parameters needed to fulfill the request.

asai_common

The programmer sets `capability` to `C_3PTC_REQ` and `primitive_type` to `C_REQUEST`. The programmer also sets the association identifier to the `sao_id` (also known as `cluster_id`) that identifies the call.

call_id

This structure contains a pointer, `id_ptr`, to a binary array of length `id_length` that specifies the ECS-assigned value of the call identifier for which the request is intended.

pool

This parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See ‘‘Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ''Error Messages'' for a list of error messages with brief explanations of their probable causes.

See Also

`asai_send()`

`C_EN_REP C_3PMC_REQ C_3PMC_CONF C_3PTC_CONF C_3PSH C_3PSH_CONF C_3PR C_3PR_CONF C_3PM C_3PM_CONF C_3PSD C_3PSD_CONF C_3PCC C_3PCC_CONF C_3PCE C_3PRC C_3PRC_CONF`

C_ABORT (3ASAI)

Name

C_ABORT — Abort an association

Type

Terminating — Unacknowledged

Usage

```
typedef struct{
        asai_common_t       asai_common;
        long                abort_type;
        long                cause_value;
}abort_info_t;
```

Description

The client sends this capability to request a user (program-initiated) abort or receives this capability as a provider (library-initiated) abort request. When the client sends an abort (U_ABORT), the cause is passed end to end. When the client receives an abort (P_ABORT), the cause is local to the library. A structure of type abort_info_t contains the parameters needed to fulfill the request.

asai_common

Within this structure, the programmer sets the capability to C_ABORT and the primitive_type to C_REQUEST to send a user abort request. When the abort is a provider abort, capability is returned as C_ABORT and primitive_type as C_REQUEST. The sao_id (also known as cluster_id) identifying the association to be aborted is set by the programmer or returned.

abort_type

This parameter indicates the type of abort. C_USER_ABT is initiated by the application program; C_PROV_ABT is initiated by the library. The programmer sets C_USER_ABT to request an abort. When the library sends an abort capability, abort_type is returned as C_PROV_ABT.

As soon as abort request is sent or received, the specified association terminates. Any outstanding messages for that association are ignored and the resources for those messages are freed. Aborts from the ECS also have an abort_type of C_USER_ABT.

cause_value

This mandatory parameter carries the cause of the abort request. It can be any of the following:

```
C_CLUST_TERM
C_INCOM_ST
C_INVALID_CRV
C_INVLDNUM
C_MISSING_IE
C_NETCONJ
C_NOLOGIN
C_NOSPLIT_MEM
C_NOUSE_RESP
C_PERM_DENIED
C_PROTERR
C_REC_TIMER
C_RESUNAVL
C_USER_C_BUSY
```

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure; when it is used in an `asai_send()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent or received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

Use of this capability through the `asai_send()` function always indicates a client abort. Only ASAI library service providers can issue a provider abort.

See Also

`asai_send() asai_rcv()`

C_EN_CAN (3ASAI)

Name

C_EN_CAN — Request termination of an event notification request

Type

Continuing — Acknowledged

Usage

```
asai_common_t    asai_common;
```

Description

The client sends this capability to request termination of event notification. A structure of type asai_common_t contains the parameters needed to fulfill the request. Within asai_common, the programmer sets capability to C_EN_CAN and primitive_type to C_REQUEST. The programmer must also set the sao_id (also known as cluster_id) that identifies the Event Notification association to be terminated.

Return Value

When this capability is used in an asai_send(), the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a client.

See Also

asai_send()

C_EN_REQ C_EN_CAN_CONF C_EN_END

C_EN_CAN_CONF (3ASAI)

Name

C_EN_CAN_CONF — Acknowledge an event notification cancel request

Type

Terminating — Unacknowledged, when received as a positive acknowledgement

Continuing — Unacknowledged, when received as a negative acknowledgement

Usage

```
typedef struct{
        asai_common_t        asai_common;
        long                 cause_value;
}enc_nak_t;

typedef union{
        asai_common_t        asai_common;
        enc_nak_t            enc_nak;
}enc_rsp_t;
```

Description

The client receives this capability as a positive or negative acknowledgement of a C_EN_CAN request. The union enc_rsp_t defines the necessary fields for the messages.

asai_common

Within this structure, capability is returned as C_EN_CAN_CONF and primitive_type is returned as C_POS_ACK for positive acknowledgement, C_NEG_ACK for negative acknowledgement. The sao_id (also known as cluster_id) returned is the association specified in the cancel request.

`cause_value`

This parameter can be any one of the following:

```
C_FACUNSUB
C_INVLDIE
C_MAND_INFO
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL
```

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, but that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

`C_EN_REQ C_EN_CONF C_EN_REP C_EN_CAN_CONF C_EN_END`

C_EN_CONF (3ASAI)

Name

C_EN_CONF — Acknowledge an event notification request

Type

Continuing — Unacknowledged, when received for positive acknowledgement

Terminating — Unacknowledged, when received for negative acknowledgement

Usage

```
typedef struct{
        asai_common_t        asai_common;
        long                 cause_value;
}en_nak_t;

typedef union{
        asai_common_t        asai_common;
        en_nak_t             en_nak;
}en_rsp_t;
```

Description

The client receives this capability as a positive or negative acknowledgement of a C_EN_REQ. The union en_rsp_t defines the necessary fields for the messages.

asai_common

Within asai_common, the capability is returned as C_EN_CONF and the primitive_type is returned as C_POS_ACK for positive acknowledgement, C_NEG_ACK for negative acknowledgement. The sao_id (also known as cluster_id) returned is the association specified in the event notification request.

cause_value

This parameter can be any of the following:

```
C_FACUNSUB
C_INVLDIE
C_INVLDNUM
C_MAND_INFO
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL
```

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, but that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

`C_EN_REQ C_EN_REP C_EN_CAN C_EN_CAN_CONF C_EN_END`

C_EN_END (3ASAI)

Name

C_EN_END — Terminate the generation of event reports

Type

Terminating — Unacknowledged

Usage

```
typedef struct{
      asai_common_t        asai_common;
      long                 cause_value;
}ene_info_t;
```

Description

The ECS sends this capability as an unacknowledged request to terminate a C_EN_REQ. A structure of type ene_info_t contains the information needed to fulfill this request.

asai_common

The programmer sets the capability to C_EN_END and the primitive_type to C_REQUEST. The programmer also sets the sao_id (also known as cluster_id).

cause_value

This parameter can be any of the following:

```
C_FACUNSUB
C_INVLDIE
C_MAND_INFO
C_OUT_OF_SERV
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL
```

If the ECS does not supply a cause_value, the library supplies the value of C_NUSE_LONG. Table 8-1 on page 8-12 provides further information on cause values.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ''Error Messages'' for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a server.

See Also

`asai_send()`

`C_EN_REQ C_EN_CONF C_EN_REP C_EN_CAN C_EN_CAN_CONF`

C_EN_REP (3ASAI)

Name

C_EN_REP — Send or receive an event report

Type

Continuing — Unacknowledged

Usage

```
typedef struct{
        asai_common_t              asai_common;
        long                       event_name;
        call_id_t                  call_id;
        long                       cause;
        char                       *connect_num;
        plan_type_t                con_num_type;
        party_id_t                 party_id;
        char                       *calling_num;
        plan_type_t                call_num_type;
        char                       *dial_num;
        plan_type_t                dial_type;
        user_user_t                uudata;
        char                       *acd_split_ext;
        trunk_id_t                 trk;
        oli_t                      oli;
        ucid_t                     ucid;    /* G3V6 */
        char                       pool[C_DATSZ];
} alert_list_t;

typedef struct {
        asai_common_t              asai_common;
        long                       event_name;
        long                       cause;
        char                       *connect_num;
        plan_type_t                con_num_type;
        call_id_t                  call_id;
        char                       *calling_num;
        plan_type_t                call_num_type;
        char                       *called_num;
        plan_type_t                called_type;
        user_user_t                uudata;
        party_id_t                 party_id;
        char                       pool[C_DATSZ];
} orig_list_t;
```

```
  typedef struct{
        asai_common_t               asai_common;
        long                        event_name;
        long                        cause;
        char                        *connect_num;
        plan_type_t                 con_type;
        char                        *calling_num;
        plan_type_t                 calling_type;
        char                        *called_num;
        plan_type_t                 called_type;
        call_id_t                   call_id;
        party_id_t                  party_id;
        trunk_id_t                  trk;
        oli_t                       oli;
        ucid_t                      ucid;       /* G3V6 */
        char                        pool[C_DATSZ];
} connect_list_t;

typedef struct{
        asai_common_t               asai_common;
        long                        event_name;
        call_id_t                   call_id;
        long                        answered_cause;
        party_id_t                  party_id;
        char                        *connect_num;
        plan_type_t                 connect_type;
        char                        *called_num;
        plan_type_t                 called_type;
        char                        pool[C_DATSZ];
} answered_list_t;

typedef struct{
        asai_common_t               asai_common;
        long                        event_name;
        call_id_t                   call_id;
        char                        *called_num;
        plan_type_t                 called_type;
        long                        num_inque;
        char                        *acd_split_ext;
        char                        pool[C_DATSZ];
} queued_list_t;

typedef struct{
        asai_common_t               asai_common;
        long                        event_name;
        call_id_t                   call_id;
        char                        *hold_num;
        plan_type_t                 hold_type;
        party_id_t                  party_id;
        char                        pool[C_DATSZ];
```

```
} hold_list_t;

typedef struct{
        asai_common_t              asai_common;
        long                       event_name;
        call_id_t                  call_id;
        char                       *reconnect_num;
        plan_type_t                recon_type;
        party_id_t                 party_id;
        char                       pool[C_DATSZ];
} reconnect_list_t;

typedef struct{
        asai_common_t              asai_common;
        long                       event_name;
        call_id_t                  call_id;
        party_id_t                 party_id;
        ucid_t                     ucid;        /* G3V6 */
        char                       pool[C_DATSZ];
} initiate_list_t;

typedef struct{
        asai_common_t              asai_common;
        long                       event_name;
        call_id_t                  call_id;
        char                       pool[C_DATSZ];
} redirected_list_t;

typedef struct{
        asai_common_t              asai_common;
        long                       event_name;
        char                       *extension;
        plan_type_t                ext_type;
        long                       reason_code;
        char                       pool[C_DATSZ];
} logout_list_t;

typedef struct{
        asai_common_t              asai_common;
        long                       event_name;
        char                       *phys_ext;
        char                       *log_ext;
        long                       work_mode;
        char                       pool[C_DATSZ];
} login_list_t;

typedef struct{
        asai_common_t              asai_common;
        long                       event_name;
        call_id_t                  call_id;
```

```
        long                        drop_cause;
        char                        *droped_num;
        plan_type_t                 droped_type;
        user_user_t                 uudata;
        party_id_t                  party_id;
        char                        pool[C_DATSZ];
} drop_list_t;

typedef struct{
        asai_common_t               asai_common;
        long                        event_name;
        call_id_t                   call_id;
        party_id_t                  party_id;
        char                        *called_num;
        plan_type_t                 called_type;
        long                        busy_cause;
        char                        pool[C_DATSZ];
} busy_list_t;

typedef struct{
        asai_common_t               asai_common;
        long                        event_name;
        call_id_t                   call_id;
        char                        *called_num;
        plan_type_t                 called_type;
        long                        denial_cause;
        char                        pool[C_DATSZ];
} denial_list_t;

typedef struct{
        asai_common_t               asai_common;
        long                        event_name;
        call_id_t                   call_id;
        party_id_t                  party_id;
        long                        location;
        long                        prog_discp;
        char                        pool[C_DATSZ];
} cuthrw_list_t;

typedef struct{
        asai_common_t               asai_common;
        long                        event_name;
        call_id_t                   call_id;
        party_id_t                  party_id;
        char                        *called_num;
        plan_type_t                 called_type;
        trunk_id_t                  trk;          /* G3V6 */
        char                        pool[C_DATSZ];
} trkszd_list_t;
```

```
typedef struct{
        asai_common_t                asai_common;
        long                         event_name;
        call_id_t                    bef_call_id;
        call_id_t                    aft_call_id;
        char                         *calling_num;
        plan_type_t                  calling_type;
        char                         *called_num;
        plan_type_t                  called_type;
        long                         part_num;
        merge_ext_t                  *party_list;
        ucid_t                       ucid;   /* G3V6*/
        char                         pool[C_DATSZ];
} conf_list_t;

typedef struct{
        asai_common_t                asai_common;
        long                         event_name;
        call_id_t                    bef_tranid;
        call_id_t                    aft_tranid;
        char                         *calling_num;
        plan_type_t                  calling_type;
        char                         *called_num;
        plan_type_t                  called_type;
        long                         part_num;
        merge_ext_t                  *party_list;
        ucid_t                       ucid;          /* G3V6*/
        char                         pool[C_DATSZ];
} tran_list_t;

typedef struct{
        asai_common_t                asai_common;
        long                         event_name;
        call_id_t                    call_id;
        char                         *calling_num;
        plan_type_t                  calling_type;
        char                         *called_num;
        plan_type_t                  called_type;
        user_user_t                  uudata;
        trunk_id_t                   trk;
        oli_t;                       oli;
        long                         domain_type;
        long                         val_length;
        char                         *domain_val;
        long                         leng_udata;
        long                         udata_type;
        long                         collect;
        long                         timeout;
        char                         *incomg_udata;
        long                         priority_level;
```

```
        long                        interflow_type;
        long                        hour;
        long                        minute;
        long                        second;
        char                        *dnis_chars;
        long                        feat_type;
        wchar_t                     *w_dnis_chars;
        ucid_t                      ucid;          /* G3V6 */
        char                        pool[C_DATSZ];
} incall_list_t;

typedef struct{
        asai_common_t               asai_common;
        long                        event_name;
} erep_common_t;

typedef struct{
        asai_common_t               asai_common;
        long                        event_name;
        call_id_t                   call_id;
        long                        calend_cause;
        char                        pool[C_DATSZ];
} calend_list_t;

typedef struct{
        asai_common_t               asai_common;
        long                        event_name;
        call_id_t                   call_id;
        user_code_t                 user_data;
        char                        pool[C_DATSZ];
} collected_digits_t;

typedef struct{
        asai_common_t               asai_common;
        long                        event_name;
        call_id_t                   call_id;
        party_id_t                  party_id;
        char                        *charged_num;
        char                        *called_num;
        trunk_id_t                  trk;
        long                        type_of_charge;
        long                        value_of_charge;
        long                        cause_value;
        char                        pool[C_DATSZ];
} charge_advice_t;

typedef union {
        erep_common_t               erep_common;
        orig_list_t                 orig_list;
        alert_list_t                alert_list;
```

```
        connect_list_t              connect_list;
        answered_list_t             answered_list;
        queued_list_t               queued_list;
        hold_list_t                 hold_list;
        reconnect_list_t            reconnect_list;
        initiate_list_t             initiate_list;
        redirected_list_t           redirected_list;
        logout_list_t               logout_list;
        login_list_t                login_list;
        drop_list_t                 drop_list;
        busy_list_t                 busy_list;
        denial_list_t               denial_list;
        cuthrw_list_t               cuthrw_list;
        trkszd_list_t               trkszd_list;
        conf_list_t                 conf_list;
        tran_list_t                 tran_list;
        incall_list_t               incall_list;
        calend_list_t               calend_list;
        collected_digits_t          collected_digits;
        charge_advice_t             charge_advice;
} evr_buf_t;
```

Description

The server sends this capability to report requested information in the form of an event report. An event report can be generated for calls controlled by the adjunct via the Call Control Capability Group for calls controlled by the adjunct via the Domain (Station) Call Control Capability Group, Domain Control of ACD and for calls for which the adjunct has requested Event Reports via the Notification Capability Group.

The union `evr_buf_t` contains the information shared by all event reports in `erep_common_t` and the specific information for each report in the remaining members.

erep_common

Within this structure, `asai_common` identifies the specific capability corresponding to the report, and the association about which the reports are to be generated. The programmer sets the `capability` to `C_EN_REP` and the `primitive_type` to `C_REQUEST`.

The event is identified by `event_name`. This parameter can be any of the values shown in the table on the following pages.

For each event there is a `evr_buf_t` member that carries the specific information related to that event. The table below gives the name of the event, its associated `evr_buf` member and a description of each item within the member.

Table 9-1. Redirection Reason Codes

Redirection Reason	Cause Value	ASAI Value
Send All Calls	CS3/31	C_FORWARD_ALL
Cover All Calls	CS3/31	C_FORWARD_ALL
Go to Cover Active	CS3/31	C_FORWARD_ALL
Adjunct-Redirected-Alerting-Call	CS3/31	C_FORWARD_ALL
Call Forwarding CAFW2	CS3/25	C_FORWARD_BUSY
Cover, principal Busy	CS3/26	C_COVER_BUSY
Cover, all call appearances busy	CS3/26	C_COVER_BUSY
Cover, does not answer	CS3/28	C_COV_DONT_ANS

Table 9-2. Event-Name Members and Descriptions

Event Name	evr_buf Member	Description
C_ALERTING	alert_list	call_id is a structure of the type call_id_t. call_id specifies the call being reported on. See‘‘Identifiers’’ on page 8-5 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.
		cause is a long integer value returned by ECS that indicates the cause of the alert. The new (4.0) Alerting Event Report redirection-reason Cause Value information will be provided when any type of domain-controlled monitored endpoint receives an alerting call as result of a redirection. Although the Cause IE in the Alerting Event is optional, it will always be provided when alerting at an endpoint is the result of a redirection. See Table 9-1 on page 9-115 for redirection reason codes.
		connect_num is a pointer to an optional null-terminated character string that indicates the alerting extension. If an extension number is not supplied, connect_num is a null pointer.
		con_num_type is an optional plan_type_t structure that supplies additional information about the structure field that immediately precedes it (providing that the preceding field is not NULL). plan_type_t has two fields: addr_type and numb_plan. addr_type can have two possible values: unknown(0) and international(1). Currently, numb_plan can have only one value, unknown(0). However, additional values for both fields may be coded in the future.
		party_id is a mandatory structure of the type party_id_t and specifies the party being alerted. See ‘‘Identifiers’’ on page 8-5 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.
		calling_num is a pointer to an optional null-terminated character string that indicates the extension of the calling party. If the extension number is not supplied, calling_num is a null pointer.
		call_num_type is an optional plan_type_t structure that supplies additional information about the structure field that immediately precedes it (providing that the preceding field is not NULL). See con_num_type above for more information concerning plan_type_t.
		dial_num is a pointer to an optional null-terminated character string that indicates the extension of the dialed number. If the extension number is not supplied, dial_num is a null pointer.

Table 9-2. Event-Name Members and Descriptions — *Continued*

Event Name	evr_buf **Member**	Description
C_ALERTING (Continued)	alert_list (Continued)	dial_type is an optional plan_type_t structure that supplies additional information about the structure field that immediately precedes it (providing that the preceding field is not NULL). See con_num_type above for more information concerning plan_type_t.
		uudata is a structure of the type user_user_t where the leng field is an integer value that indicates the number of octets of user data included in the request.This field is set to 0 if no user_user data is present. Currently, ECS will accept up to 32 bytes of data (leng = 32). If more than 32 bytes is specified, an error will be returned. ASAI supports a maximum user_user data length of 32 bytes, although user_user data generated from a PRI trunk can be up to 127 bytes in length. (DEFINITY ECS will truncate this to 32 bytes before forwarding it to ASAI link.)
		The protocol field indicates the type of information and is restricted to C_UU_USER that indicates a user-specific protocol or C_UU_IA5 that indicates IA5 or ASCII characters. The info field is a pointer to an ASCII string of no more that 32 characters in length.
		acd_split_ext is a pointer to an optional null-terminated character string that indicates the ACD split extension that has distributed the call to an alerting agent. If the number is not supplied, acd_split_ext is a null pointer.
		trk is a structure of type trunk_id_t. Within this structure, direct is a parameter that can be C_NODIRET, C_INCTRK or C_OUTTRK. These values indicate the direction the trunk was used (no direction, incoming or outgoing). If the ECS does not supply this item, the library supplies a value of C_NUSE_LONG. grp_ptr is a pointer to an optional character string of length gid_length that indicates a ECS-assigned identifier of the trunk that was used. If this item is not ECS-provided, the pointer is a null pointer. Similarly, id_ptr is a pointer to a character string of length id_length that indicates the ECS-assigned identifier of the trunk that was used. If this item is not ECS-provided, id_ptr is a null pointer.
		oli is an optional oli_t structure that specifies II digits from the originating line information for the call.

Table 9-2. Event-Name Members and Descriptions — *Continued*

Event Name	evr_buf Member	Description
C_ALERTING (Continued)	alert_list (Continued)	ucid (Universal Call ID) is an optional 8 byte binary value assigned by ECS.
		pool is a character array that provides buffer space for the ASAI library to store program specific information. See ''Pool'' on page 8-15 in Chapter 8, ''ASAI Capability Primitives'' for more information.
C_CALL_ORIG	orig_list	call_id: See C_ALERTING.
		cause: See C_ALERTING.
		connect_num (optional) A pointer to a null-terminated string that indicates the originating device. Normally this is the same as the call number and then omitted, however, in this case where the call is originated from a logical agent extension, this will indicate the physical extension from which the call was made.
		con_num_type: See C_ALERTING.
		call_id: See C_ALERTING.
		calling_num: is a pointer to a null terminated string that indicates the number that originated the call, usually the extension number. For Third Party Make Calls originating from a logical agent, this is the logical agent number of the agent that is logged into the station making the call.
		call_num_type: See C_ALERTING.
		called_num is a pointer to an optional null-terminated string that indicates the called extension. If the extension number is not supplied, called_num is a null pointer.
		called_type is an optional plan_type_t structure that supplies additional information about the structure field that immediately precedes it (providing that the preceding field is not NULL). See C_ALERTING: con_num_type for more information concerning plan_type_t.
		uudata: See C_ALERTING.
		party_id: the party number of the originating device on the call.
		pool: See C_ALERTING
C_COLLECTED	collected_ digits	call_id: See C_ALERTING.
		user_data provides the user information that will be sent from ECS.
		pool: See C_ALERTING.

Table 9-2. Event-Name Members and Descriptions — *Continued*

Event Name	evr_buf Member	Description
C_CONNECTED	connect_list	cause: See C_ALERTING.
		connect_num: See C_ALERTING.
		con_type is an optional plan_type_t structure that supplies additional information about the structure field that immediately precedes it (providing that the preceding field is not NULL). See C_ALERTING: con_num_type for more information concerning plan_type_t.
		calling_num: See C_ALERTING.
		calling_type is an optional plan_type_t structure that supplies additional information about the structure field that immediately precedes it (providing that the preceding field is not NULL). See C_ALERTING: con_num_type for more information concerning plan_type_t.
		called_num: See C_CALL_ORIG.
		called_type: See C_CALL_ORIG.
		call_id: See C_ALERTING.
		party_id: See C_ALERTING.
		trk: See C_ALERTING.
		oli: See C_ALERTING.
		ucid: See C_ALERTING.
		pool: See C_ALERTING.

Table 9-2. Event-Name Members and Descriptions — *Continued*

Event Name	evr_buf Member	Description
C_ANSWERED		call_id: See C_ALERTING.
		answered_cause is an optional item that indicates a cause value. This item can assume any ECS-specified cause value or C_NUSE_LONG.
		party_id: See C_ALERTING.
		connect_num: See C_ALERTING.
		connect_type is an optional plan_type_t structure that supplies additional information about the structure field that immediately precedes it (providing that the preceding field is not NULL). See: C_ALERTING: con_num_type for more information concerning plan_type_t.
		called_num: See C_CONNECTED.
		called_type: See C_CONNECTED.
		pool: See C_ALERTING.
C_QUEUED	queued_list	call_id: See C_ALERTING.
		called_num: See C_CONNECTED.
		called_type: See C_CONNECTED.
		num_inque is an optional integer value that indicates the number of calls in the queue, including the current call. If a value is not supplied, the default value is C_NUSE_LONG.
		acd_split_ext is a pointer to an optional null-terminated string that indicates the extension of the ACD split servicing the queue. If the extension number is not supplied, acd_split_ext is a null pointer.
		pool: See C_ALERTING.

Table 9-2. Event-Name Members and Descriptions — *Continued*

Event Name	evr_buf **Member**	Description
C_HOLD	hold_list	call_id: See C_ALERTING.
		hold_num is a pointer to a null-terminated string indicating the number held. If the number is not supplied, the parameter is a null pointer.
		hold_type is an optional plan_type_t structure that supplies additional information about the structure field that immediately precedes it (providing that the preceding field is not NULL). See C_ALERTING: con_num_type for more information concerning plan_type_t.
		party_id: See C_ALERTING.
		pool: See C_ALERTING.
C_RECONNECT	reconnect_list	call_id: See C_ALERTING.
		reconnect_num is a pointer to a null-terminated string indicating the number held. If the number is not supplied, the parameter is a null pointer.
		recon_type is an optional plan_type_t structure that supplies additional information about the structure field that immediately precedes it (providing that the preceding field is not NULL). See C_ALERTING: con_num_type for more information concerning plan_type_t.
		party_id: See C_ALERTING.
		pool: See C_ALERTING.
C_INITIATE	initiate_list	call_id: See C_ALERTING.
		party_id: See C_ALERTING.
		ucid: See C_ALERTING.
		pool: See C_ALERTING.
C_REDIRECTED	redirected_list	call_id: See C_ALERTING.
		pool: See C_ALERTING.
C_LOGIN	login_list	phys_ext is a pointer to a null-terminated string indicating the physical extension logged into a specific split/skill.
		log_ext is a pointer to a null-terminated string that indicates the logical extensions (EAS only) of the agent.
		work_mode can be C_AUX_WORK, C_AFTCAL_WK, C_AUTO_IN, C_MANUAL_IN (auxiliary mode, after call work mode, auto-in mode and manual-in-mode).
		pool: See C_ALERTING.

Table 9-2. Event-Name Members and Descriptions — *Continued*

Event Name	evr_buf **Member**	Description
C_LOGOUT	logout_list	extension is a pointer to a null-terminated string indicating the extension of the agent who was logged out.
		ext_type is an optional plan_type_t structure that supplies additional information about the structure field that immediately precedes it (providing that the preceding field is not NULL). See C_ALERTING: con_num_type for more information concerning plan_type_t.
		logical is a pointer to a null-terminated string indicating the logical extension of the agent logging out (EAS environment only).
		reason_code is a long integer from 1 to 9, indicating the reason why the agent logged out.
		pool: See C_ALERTING.
C_DROP	drop_list	call_id is a structure of the type call_id_t where id_ptr points to a mandatory character array that indicates the ECS-assigned identifier of the dropped call and id_length is the length of the call identifier.
		drop_cause is an optional integer value that indicates the cause of the dropped call. This item can be any ECS-specific cause value or C_NUSE_LONG.
		droped_num is a pointer to a null-terminated string that indicates the extension dropped. If the extension number is not supplied, droped_num is a null pointer.
		droped_type is an optional plan_type_t structure that supplies additional information about the structure field that immediately precedes it (providing that the preceding field is not NULL). See C_ALERTING: con_num_type for more information concerning plan_type_t.
		uudata: See C_ALERTING.
		party_id: See C_ALERTING.
		pool: See C_ALERTING.

Table 9-2. Event-Name Members and Descriptions — *Continued*

Event Name	evr_buf Member	Description
C_BUSY	busy_list	call_id is a structure of the type call_id_t, where id_ptr points to a character array that indicates the ECS-assigned identifier of the call that generated the busy condition. id_length indicates the length of the call identifier.
		party_id: See C_ALERTING.
		called_num: See C_CONNECTED.
		called_type: See C_CONNECTED.
		busy_cause is an optional parameter that indicates the cause of the busy event. It can be C_USER_BUSY, C_QUEFULL, C_INVDNUM, or C_NUSE_LONG.
		pool: See C_ALERTING.
C_CHARGE_ADVICE	charge_advice	call_id: See C_ALERTING.
		party_id: See C_ALERTING.
		charged_num is a pointer to the extension that placed the call.
		called_num: See C_CONNECTED
		trunk_grp is a long integer value returned by the switch that designates the trunk group for which AOC is provided.
		trunk_mem is a long returned by the switch that indicates the specific member of the trunk group for which AOC is provided.
		type_of_charge is a long that denotes the type of charge returned and can take the following values: C_INTERMEDIATE for an intermediate charge, C_FINAL for the final charge and C_SPLIT for a split charge.
		value_of_charge is a long returned by the switch to indicate the amount of charging units.
		cause_value is a long indicating that final charge was not received by the network in the first clearing message. The final charge will be set to 0 and “cause” will be set to CS3/38 “Network out-of-order.”

Table 9-2. Event-Name Members and Descriptions — *Continued*

Event Name	evr_buf **Member**	Description
C_DENIAL	denial_list	call_id: See C_ALERTING.
		called_num: See C_CONNECTED.
		called_type: See C_CONNECTED.
		denial_cause is an optional integer value that indicates the cause of the reorder condition. It can be any ECS-specified cause value or C_NUSE_LONG.
		pool: See C_ALERTING.
C_CUT_THROUGH	cuthrw_list	call_id: See C_ALERTING.
		party_id: See C_ALERTING.
		location is a mandatory parameter that shows where the progress indication is coming. It can be C_USER, C_PUBLU, C_PUBRU or C_PRIRU (user, public network serving local user, public network serving remote user or private network serving remote user).
		prog_discp is a mandatory parameter that describes the progress indication. It can be C_OFFISDN, C_DESTNISDN, C_ORIGNISDN, C_ONISDN or C_INBAND (call not end-to-end ISDN, destination is nonISDN, origination is nonISDN, call returned to ISDN, or in-band information is now available).
		pool: See C_ALERTING.
C_TRK_SEIZED	trkszd_list	call_id: See C_ALERTING.
		party_id: See C_ALERTING.
		called_num: See C_CONNECTED.
		called_type: See C_CONNECTED.
		trk: See C_ALERTING.
		pool: See C_ALERTING.

Table 9-2. Event-Name Members and Descriptions — *Continued*

Event Name	evr_buf Member	Description
C_CONFERENCED	conf_list	bef_call_id is a mandatory structure of the type call_id_t, where id_ptr points to a character array that indicates ECS-assigned identifier of the call before conferencing. id_length is the length of the call identifier.
		aft_call_id is a mandatory structure of the type call_id_t, where id_ptr points to a character array that indicates the ECS-assigned identifier of the call after conferencing. id_length is the length of the call identifier.
		calling_num: See C_ALERTING.
		calling_type: See C_CONNECTED.
		called_num: See C_CONNECTED.
		called_type: See C_CONNECTED.
		part_num is the number of structures of the type merge_ext_t contained in the buffer pointed to by party_list.
		party_list is a pointer to a list of structures of the type merge_ext_t. Within merge_ext_t, party_id identifies a party on the conferenced call, id_ptr points to a character array that identifies the ECS-assigned identifier of the party being conferenced. id_length is the length of the party identifier. extension is a pointer to a null-terminated string that specifies the extension of the party. old_pid indicates the old party identifier (before the call is merged), which call can assume the values of C_RESULTING_CALL or C_OTHER_CALL and indicates whether the old party belongs to the resulting call or to other calls. If this item is not supplied, extension is a null pointer.
		ucid: See C_ALERTING.
		pool: See C_ALERTING.

Table 9-2. Event-Name Members and Descriptions — *Continued*

Event Name	evr_buf Member	Description
C_TRANSFERED	tran_list	bef_tranid is a mandatory structure of the type call_id_t, where id_ptr points to a character array that indicates the ECS-assigned identifier of the call before transfer. id_length is the length of the call identifier.
		aft_tranid is a mandatory structure of the type call_id_t, where id_ptr points to a character array that indicates the ECS-assigned identifier of the call after transfer. id_length is the length of the call identifier.
		calling_num: See C_ALERTING.
		calling_type: See C_CONNECTED.
		called_num: See C_CONNECTED.
		called_type: See C_CONNECTED.
		part_num: See C_CONFERENCED.
		party_list: See C_CONFERENCED.
		ucid: See C_ALERTING.
		pool: See C_ALERTING.

Table 9-2. Event-Name Members and Descriptions — *Continued*

Event Name	evr_buf Member	Description
C_OFFERED	incall_list	call_id: See C_ALERTING.
		calling_num: See C_ALERTING.
		calling_type: See C_CONNECTED.
		called_num: See C_CONNECTED.
		called_type: See C_CONNECTED.
		uudata: See C_ALERTING.
		trk: See C_ALERTING.
		oli: See C_ALERTING.
		domain_type can be C_TRUNK_GROUP, C_ACD_SPLIT, C_EXTENSION, C_HUNTGRP, C_VDN, indicating the domain type of the party (trunk group access code, ACD split, extension, hunt group, or vector directory number).
		val_length is the number of characters pointed to by domain_val.
		domain_val is a pointer to a character string of length val_length that indicates the ECS-specified value of the domain extension.
		leng_udata is an integer value that indicates the number of octets of user data included in the offered call. If this item is not supplied by the ECS, it assumes the default value of C_NUSE_LONG.
		collect is a boolean field containing the collect or collected flag.

Table 9-2. Event-Name Members and Descriptions — *Continued*

Event Name	evr_buf Member	Description
C_OFFERED (Continued)	incall_list (Continued)	udata_type indicates the type of user data. It can be C_LOGIN_DIGITS, C_CALL_PROMPTER or C_NUSE_LONG (login digits, call prompter information or not used).
		timeout specifies the digit collection time out and is an integer value from 0 to 63 (the default value is 0).
		The incomg_udata is a pointer to an optional character string of user data supplied with the call offered. If the string is not supplied by the ECS, incomg_udata points to a null character as the first character of the string.
		priority_level is an integer value that indicates the priority of the incoming call. It can be C_NOT_IN_QUEUE, C_LOW, C_MEDIUM, C_HIGH, or C_TOP. If this item is not ECS-supplied, it assumes the default value of C_NUSE_LONG.
		interflow_type is an integer value that indicates the type of interflow. It can be C_ALL_INTERFC_LOW, C_THRESHOLD_INTERFC_LOW or C_VECTORING_INTERFC_LOW. If this item is not ECS-supplied, it assumes the value of C_NUSE_LONG.
		The parameters hour, minute and second indicate the elapsed time the offered call has spent in the queue. This time is given in the integer values. If elapsed time is not supplied, all three item(s) will assume the value of C_NUSE_LONG.
		dnis_chars is a pointer to a null-terminated string that indicates the identity of the interflow source. If this item is not ECS-supplied, it is a null pointer. w_dnis_chars points to a translation of the raw OPTREX data in dnis_chars into its locale equivalent ISO (four bytes) characters. (See OPTREX, locale, ISO characters in the glossary for more information.) feat_type is an integer value that indicates what feature (if any) is associated with this call, feat type can be C_FLEX_BILL, indicating this call is available for flexible billing.
		ucid: See C_ALERTING.
		pool: See C_ALERTING.

Table 9-2. Event-Name Members and Descriptions — *Continued*

Event Name	evr_buf Member	Description
C_CALEND	calend_list	call_id: See C_ALERTING.
		calend_cause can be any of the following: C_ANSWERING_MACHINE, C_CALLID_TERM, C_NORMAL, C_NUM_CHANGED,C_UNSPECIFIED. These values are similar to cause_value and are explained in Chapter 8, ‘‘ASAI Capability Primitives.’’ If this item is not ECS-supplied, it assumes the value of C_NUSE_LONG.
		pool: See C_ALERTING.
C_COLLECTED	collected_ digits	call_id: See C_ALERTING.
		user_data provides the user information that will be sent from ECS.
		pool: See C_ALERTING.

Return Value

When this capability is used in an asai_send(), the function returns a nonnegative value when successful and -1 on failure. A return value of 0 indicates that there is no data to be sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

asai_send()

C_EN_REQ C_3PMC_REQ

C_EN_REQ (3ASAI)

Name

C_EN_REQ — Request event reports for a specified domain

Type

Initiating — Acknowledged

Usage

```
typedef struct{
        asai_common_t            asai_common;
        long                     domain_type;
        char                     *domain_ext;
        char                     pool[C_DATSZ];
}acd_grp_info_t;

typedef struct{
        asai_common_t            asai_common;
        long                     domain_type;
        char                     *domain_ext;
        char                     pool[C_DATSZ];
}charge_advice_info_t;

typedef struct{
        asai_common_t            asai_common;
        long                     domain_type;
        char                     *domain_ext;
        char                     pool[C_DATSZ];
}cv_info_t;

typedef struct{
        asai_common_t            asai_common;
        long                     domain_type;
}en_common_t;

typedef union{
        en_common_t              en_common;
        acd_grp_info_t           acd_grp_info;
        cv_info_t                cv_info;
        charge_advice_info_t     charge_advice_info;
}en_buf_t;
```

Description

The client sends this capability to request event notification. The parameters associated with this capability are defined within `en_buf`. A set of common parameters for this capability is defined in `en_common`.

Within `en_common`, the programmer sets the `asai_common` parameters of `capability` to `C_EN_REQ` and `primitive_type` to `C_REQUEST`. `domain_type` indicates the domain of the events requested and can be `C_ACD_GROUP` or `C_CALL_VECTOR`, or `C_AOC_TRUNK` (ACD group or call vector, or AOC trunk). This capability initiates reports on the following items: `C_ALERTING`, `C_CONNECTED`, `C_ANSWERED`, `C_QUEUED`, `C_DROP`, `C_BUSY`, `C_DENIAL`, `C_HOLD`, `C_INITIATE`, `C_RECONNECT`, `C_LOGIN`, `C_LOGOUT`, `C_TRK_SEIZED`, `C_COLLECT`, `C_CALEND`, `C_CUT_THROUGH`, `C_CONFERENCED`, `C_TRANSFERED`, `C_CALL_ORIG`, `C_OFFERED`, and `C_CHARGE_ADVICE`.

The `en_buf` union members of `acd_grp_info` and `cv_info` contain the identifying information of the requested domain. The following table shows the domain, identifying parameter and a description of the parameter. The parameters listed in the table are mandatory.

Table 9-3. Domain Value Parameters and Description

Domain Value	Parameter	Description
C_ACD_GROUP	domain_ext	Pointer to a null-terminated string that indicates the extension of the ACD split.
	pool	Character array that provides buffer space for the ASAI library to store program-specific information. See ‘‘Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.
C_AOC_TRUNK	domain_ext	Pointer to the default string “#####” that designates all trunk groups. Only the requests containing the default string “#####” will be acknowledged.
	pool	Character array that provides buffer space for the ASAI library to store program-specific information. See the section ‘‘Pool’’ in Chapter 8, “ASAI Capability Primitives” for more information.
C_CALL_VECTOR	domain_ext	Pointer to a null-terminated string that indicates the extension of the vector directory number.
	pool	Character array that provides buffer space for the ASAI library to store program-specific information. See ‘‘Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_send()`

`C_EN_REP C_EN_CONF C_EN_CAN C_EN_CAN_CONF C_EN_END`

C_EN_SCN (3ASAI)

Name

C_EN_SCN — Stop sending event reports for a specified call

Type

Continuing — Acknowledged

Usage

```
typedef struct{
        asai_common_t      asai_common;
        call_id_t          call_id;
        char               pool[C_DATSZ];
}en_scn_t;
```

Description

The client sends this capability to request termination of event reports on a specified call. The call is identified by `call_id`. Only one `call_id` is allowed per request. The needed information is contained in the `en_scn_t` structure.

asai_common

The programmer sets the `capability` to `C_EN_SCN` and the `primitive_type` to `C_REQUEST`. The programmer also sets the `sao_id` (also known as `cluster_id`) to identify association inside which the call notification is being terminated.

call_id

`id_length` specifies the length of ECS-assigned call identifier and `id_ptr` points to a binary array that specifies ECS-assigned identifier of the monitored call.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a client.

See Also

`asai_send()`

`C_EN_SCN_CONF C_EN_REQ C_EN_CAN C_EN_CAN_CONF C_EN_END`

C_EN_SCN_CONF (3ASAI)

Name

C_EN_SCN_CONF — Acknowledge a stop call notification request

Type

Continuing — Unacknowledged, when used as a positive or negative acknowledgement

Usage

```
typedef struct{
        asai_common_t         asai_common;
        long                  cause_value;
}en_scn_nak_t;

typedef union{
        asai_common_t         asai_common;
        en_scn_nak_t          en_scn_nak;
}en_scn_rsp_t;
```

Description

The client receives this capability as a positive or negative acknowledgement to a C_EN_SCN request.

asai_common

The capability is returned as C_EN_SCN_CONF and the primitive_type is returned as C_POS_ACK for positive acknowledgement, C_NEG_ACK for negative acknowledgement. The sao_id (also known as cluster_id) specified in the stop call notification request is also returned.

cause_value

This parameter can be any of the following:

C_FACUNSUB
C_INVLDIE
C_MAND_INFO
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, but no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

`C_EN_SCN C_EN_REQ C_EN_CONF C_EN_REP C_EN_CAN C_EN_CAN_CONF C_EN_END`

C_HB_CONF (3ASAI)

Name

C_HB_CONF —Acknowledge a heartbeat request

Type

Terminating — Unacknowledged

Usage

```
typedef struct{
        asai_common_t        asai_common;
        long                 cause_value;
}hb_nak_t;

typedef union{
        asai_common_t        asai_common;
        hb_nak_t             hb_nak;
}hb_rsp_t;
```

Description

The client (that is, for this request, it can be ECS or the adjunct) receives this capability as a positive or negative acknowledgement of a C_HB_REQ request. The union hb_rsp_t contains the necessary information.

asai_common

The capability is returned as C_HB_CONF and primitive_type is returned as C_POS_ACK for positive acknowledgement, C_NEG_ACK for negative acknowledgement. The sao_id (also known as cluster_id) specified in the heartbeat request is also returned.

cause_value

This parameter can be any of the following:

```
C_FACUNSUB
C_INVLDIE
C_MAND_INFO
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL
```

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure; when it is used in an `asai_send()`, the function returns a nonnegative value on success and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent or received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_send() asai_rcv()`

`C_HB_REQ`

C_HB_REQ (3ASAI)

Name

C_HB_REQ — Request heartbeat

Type

Initiating — Acknowledged

Usage

```
asai_common_t    asai_common;
```

Description

The client (that is, ECS or the adjunct) sends this capability to request information on the state of the data link connection providing service (heartbeat). The response implies the status of the provider servicing the data link. The asai_common structure contains the information needed to fulfill the request.

Within asai_common, the programmer sets capability to C_HB_REQ and primitive_type to C_REQUEST. The programmer also sets the sao_id (also known as cluster_id).

Return Value

When this capability is used in an asai_rcv(), the function returns the size of the message when successful and -1 on failure; when it is used in an asai_send(), the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent or received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

asai_send() asai_rcv()

C_HB_CONF

C_RF_CONF (3ASAI)

Name

C_RF_CONF — Acknowledge a request feature request

Type

Terminating — Unacknowledged

Usage

```
typedef struct{
        asai_common_t        asai_common;
        long                 cause_value;
}rf_nak_t;

typedef union{
        asai_common_t        asai_common;
        rf_nak_t             rf_nak;
}rf_rsp_t;
```

Description

The client receives this capability as a positive or negative acknowledgement of a C_RF_REQ request. The union rf_rsp contains the necessary information.

asai_common

The capability is returned as C_RF_CONF and the primitive_type as C_POS_ACK for positive acknowledgement, C_NEG_ACK for negative acknowledgement. The sao_id (also known as cluster_id) is also returned.

cause_value

This parameter can be any of the following:

C_AGT_STATE
C_BAD_ADMIN
C_FACUNSUB
C_FEATURE_REJECTED
C_INCOM_OPT
C_INC_PASWD
C_INCS_AGT_ST
C-INVALID_CRV
C_INVLDIE
C_INVLDNUM
C_MAND_INFO
C_MAX_LOGIN
C_NETCONJ

```
C_NOLOGIN
C_NOSPLIT_MEM
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL
C_TEMP_FAILURE
C_USER_BUSY
```

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, but that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

`C_RF_REQ`

C_RF_REQ (3ASAI)

Name

C_RF_REQ — Activate or deactivate a call feature for a specified endpoint

Type

Initiating — Acknowledged

Usage

```
typedef struct{
        asai_common_t   asai_common;
        long            rf_item;
        char            *agent_id;
        char            *split_ext;
        char            *agt_ext;
        long            work_mode;
        char            pool[C_DATSZ];
}agt_login_t;

typedef struct{
        asai_common_t   asai_common;
        long            rf_item;
        char            *split_ext;
        char            *agt_ext;
        long            reason_code;
        char            pool[C_DATSZ];
}agt_logout_t;

typedef struct{
        asai_common_t   asai_common;
        long            rf_item;
        char            *split_ext;
        char            *agt_ext;
        long            work_mode;
        long            reason_code;
        char            pool[C_DATSZ];
}chg_wkmod_t;

typedef struct{
        asai_common_t   asai_common;
        long            rf_item;
        char            *called_num;
        char            *redir_num;
        char            pool[C_DATSZ];
}call_frwd_t;
```

```
typedef struct{
        asai_common_t   asai_common;
        long            rf_item;
        char            *called_num;
        char            pool[C_DATSZ];
}can_frwd_t;

typedef struct{
        asai_common_t   asai_common;
        long            rf_item;
        char            *called_num;
        char            pool[C_DATSZ];
}sac_t;

typedef struct{
        asai_common_t   asai_common;
        long            rf_item;
        char            *called_num;
        char            pool[C_DATSZ];
}sac_can_t;

typedef struct{
        asai_common_t   asai_common;
        long            rf_item;
}rf_common_t;

typedef union{
        rf_common_t     rf_common;
        agt_login_t     agt_login;
        agt_logout_t    agt_logout;
        chg_wkmod_t     chg_wkmod;
        call_frwd_t     call_frwd;
        can_frwd_t      can_frwd;
        sac_t           sac;
        sac_can_t       sac_can;
}rf_buf_t;
```

Description

The client sends this capability to request activation of a specific feature. The `rf_buf_t` union members contain either common or specific information needed to fulfill the request. The common parameters are contained in the `rf_common_t` structure; the specific parameters for each feature are contained in the remaining union members.

`rf_common_t`

Within `asai_common`, the programmer sets the `capability` to `C_RF_REQ` and the `primitive_type` to `C_REQUEST`. The association is identified by the `sao_id` (also known as `cluster_id`). The parameter `rf_item` identifies the feature being requested. It can be C_AGT_LOGIN, C_AGT_LOGOUT, C_CHAGT_WKMOD, C_CALL_FRWD, C_CAN_FRWD, C_SAC, or C_SAC_CAN (agent login, agent logout, change of agent work mode, call forwarding, cancel call forwarding, send all calls, and cancel send all calls).

agt_login_t

This structure contains the parameters needed to request the agent login feature. agent_id is a pointer to a null-terminated string that indicates the agent login identifier and/or password. With traditional ACD, the agent's password is in the agent_id field. With EAS, the logical agent id is in the agent_id field. If a password is required then a number and the password are added after the logical agent id. split_ext is a pointer to a null-terminated string that specifies the extension for which the agent is logging in. agt_ext is a pointer to a null-terminated string that specifies the agent extension. work_mode can be C_AUX_WORK, C_AFTCAL_WK, C_AUTO_IN, or C_MANUAL_IN (auxiliary mode, after call work mode, auto-in mode, and manual-in mode). pool is a character array that provides buffer space for the ASAI library to store program-specific information. See ''Pool'' on page 8-15 in Chapter 8, ''ASAI Capability Primitives'' for more information.

agt_logout_t

This structure contains the parameters needed to request the agent logout feature. split_ext is a pointer to a null-terminated string that specifies the extension from which the agent is logging out. agt_ext is a pointer to a null-terminated string that specifies the agent extension. reason_code (option) is a long integer indicating the reason for the agent logout. pool is a character array that provides buffer space for the ASAI library to store program-specific information. See ''Pool'' on page 8-15 in Chapter 8, ''ASAI Capability Primitives'' for more information.

chg_wkmod_t

This structure contains the parameters needed to request a change in the agent work mode feature. split_ext is a null-terminated string that specifies the extension for which the work mode is to be changed. agt_ext is a pointer to a null-terminated string that specifies the agent extension. work_mode can be C_AUX_WORK, C_AFTCAL_WK, C_AUTO_IN, or C_MANUAL_IN (auxiliary mode, after call work mode, auto-in mode, and manual-in mode). reason_code (option) is a long integer value indicating the reason for the change of work modes. pool is a character array that provides buffer space for the ASAI library to store program-specific information. See ''Pool'' on page 8-15 for more information.

call_frwd_t

This structure contains the parameters needed to request the call forwarding feature. called_num is a pointer to a null-terminated string that specifies the extension from which the call is to be forwarded. redir_num is a pointer to a null-terminated string that specifies the extension to which the call is to be forwarded. `pool` is a character array that provides buffer space for the ASAI library to store program-specific information. See “Pool” on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

`can_frwd_t`

This structure contains the parameters needed to request the cancel call forwarding feature. `called_num` is a pointer to a null-terminated string that specifies the extension from which call forwarding was invoked. `pool` is a character array that provides buffer space for the ASAI library to store program-specific information. See “Pool” on page 8-15 in Chapter 8, “ASAI Capability Primitives’’ for more information.

sac_t

This structure contains the parameters needed to request the send all calls feature. `called_num` is pointer to a null-terminated string that specifies the extension for which send all calls is to be invoked. `pool` is a character array that provides buffer space for the ASAI library to store program-specific information.

`sac_can_t`

This structure contains the parameters needed to cancel the send all calls feature for a specified extension. `called_num` is a pointer to a null-terminated string that specifies the extension for which the send all calls feature is to be cancelled. `pool` is a character array that provides buffer space for the ASAI library to store program-specific information. See “Pool’’ on page 8-15 in Chapter 8, “ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_send()`

`C_RF_CONF`

C_RT_END (3ASAI)

Name

C_RT_END — Terminate a route request

Type

Terminating — Unacknowledged

Usage

```
typedef struct{
        asai_common_t   asai_common;
        long            cause_value;
}rte_info_t;
```

Description

The client (that is, ECS) sends this capability to request termination of a routing request. A structure of type rte_info_t contains the information needed to fulfill the request.

asai_common

The programmer sets the capability to C_RT_END and the primitive_type to C_REQUEST. The programmer also sets the sao_id (also known as cluster_id) to identify the association in the route request.

cause_value

This parameter can be any of the following:

C_AGT_NOT_SPLIT_ME
C_CALLID_TERM
C_FACUNSUB
C_INCOM_ST
C_INVLDIE
C_INVLDNUM
C_MAND_INFO
C_NOLOGIN
C_NORMAL
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a client.

See Also

`asai_rcv()`

`C_RT_REQ C_RT_SEL`

C_RT_REQ (3ASAI)

Name

C_RT_REQ — Request a call route

Type

Initiating — Acknowledged

Usage

```
typedef struct{
      sai_common_t  asai_common;
      char          *calling_num;
      plan_type_t   calling_type;
      char          *called_num;
      plan_type_t   called_type;
      user_user_t   data;
      long          leng_udata;
      long          udata_type;
      long          collect;
      long          timeout;
      char          *call_udata;
      char          *vdn_num;
      long          priority_level;
      long          interflow_type;
      long          hour;
      long          minute;
      long          second;
      char          *dnis_chars;
      call_id_t     call_id:
      trunk_id_t    trk;
      oli_t         oli;
      wchar_t       *w_dnis_chars;
      ucid_t        ucid;               /* G3V6 */
      char          pool[C_DATSZ];
}rt_info_t;
```

Description

The client (that is, ECS) sends this capability to request a call route. A structure of type `rt_info_t` contains the information needed to fulfill the request.

asai_common

The programmer sets the `capability` to `C_RT_REQ` and the `primitive_type` to `C_REQUEST`. The programmer also sets the `sao_id` (also known as `cluster_id`).

`calling_num`

This parameter is a pointer to a null-terminated string that specifies the calling extension. (See `trk` parameter for more information.)

`calling_type`

This parameter is an optional `plan_type_t` structure that supplies additional information about the structure field that immediately precedes it (providing that the preceding field is not NULL).

`called_num`

This parameter is a pointer to a null-terminated string that specifies the called extension of the request.

`called_type`

This parameter is a `plan_type_t` structure that supplies additional information about the structure field that immediately precedes it.

`uudata`

The purpose of the user-user information is to convey information between ISDN users. This information is not interpreted by ECS, but rather is carried transparently and delivered to the remote user. If the `info` field is IA5 characters (ASCII), the string is null-terminated.

The `leng` field is an integer value that indicates the number of octets of user data included in the request. This field is set to 0 if no `user_user` data is present. Currently, ECS will accept up to 32 bytes of data (`leng` = 32). If more than 32 bytes is specified, an error will be returned. ASAI supports a maximum `user_user` data length of 32 bytes, although `user_user` data generated from a PRI trunk can be up to 127 bytes in length. (DEFINITY ECS will truncate this to 32 bytes before forwarding it to ASAI link.)

The `protocol` field indicates the type of information and is restricted to `C_UU_USER` that indicates a user-specific protocol or `C_UU_IA5` which indicates IA5 or ASCII characters.

The `info` field is a pointer to an ASCII string of no more that 32 characters in length.

`leng_udata`

This parameter is an integer value that indicates the number of octets of user data included in the offered call.

`udata_type`

This parameter indicates the type of user data. It can be `C_LOGIN_DIGITS`, `C_CALL_PROMPTER` or `C_NUSE_LONG` (login digits, call prompter information or not used).

`collect`

The `collect` parameter is a boolean field containing the collect or collected flag.

`timeout`

This parameter specifies the digit collection time out and is an integer value from 0 to 63 (the default is 0).

`call_udata`

This parameter is a pointer to a null-terminated character string of user data. If `leng_udata` is `0`, then it points to a null character as the first character.

`vdn_num`

This parameter contains the domain (VDN) from which the route request is being made.

`priority_level`

This parameter is an integer value that indicates the priority of the call. It can be `C_NOT_IN_QUEUE`,`C_LOW`,`C_MEDIUM`,`C_HIGH`, or `C_TOP`. If this item is not supplied, it assumes the value of `C_NUSE_LONG`.

`interflow_type`

This parameter is an integer value that indicates the type of interflow. It can be `C_ALLINT_FLOW`,`C_THRINT_FLOW` or `C_VECINT_FLOW`. If this item is not supplied, it assumes the value of `C_NUSE_LONG`.

`hour, minute, second`

These parameters indicate the time the routed call is to spend in the queue before interflow. Time is given in the integer values of `hour, minute, second`. If ECS does not supply a time value, the library sets all of these items to the value of `C_NUSE_LONG`.

`dnis_chars`

This parameter is a pointer to an optional ECS-specified null-terminated string that indicates an identifier of the interflow source. If this parameter is not supplied by ECS, it is a null pointer.

`w_dnis_chars`

This parameter points to a translation of the raw optrex data in `dnis_chars` into its locale equivalent ISO (four bytes) characters. (See optrex, locale, and special characters in the Glossary and Appendix A, "Special Characters" for more information.)

`call_id`

This parameter identifies the call to be routed.

`trk`

This parameter is optional. It identifies the trunk number from which the call originated. Note that `calling_num` and `trk` are mutually exclusive. One or the other will be present, but not both.

`oli`

This parameter is optional. It identifies the Information Indicator (II) digits received in the originating line information IE for the call.

`ucid`

This parameter identifies the ucid (Universal Call ID) assigned to the call by ECS.

`pool`

This parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See ‘‘Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

When this capability is used in an `asai_rcv()`, the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_send()`

`C_RT_SEL C_RT_END`

C_RT_SEL (3ASAI)

Name

C_RT_SEL — Inform a route requester of a proposed route selection

Type

Continuing — Unacknowledged

Usage

```
typedef struct{
   long                 type;
   long                 collect;
   long                 timeout;
   char                 *digits;
}user_code_t;

typedefstruct{
   asai_common_t        asai_common;
   char                 *calling_num;
   char                 *called_num;
   user_user_t          uudata;
   long                 ofacility;
   char                 *ofac_ext;
   long                 priority_call;
   long                 dir_agtcall;
   user_code_t          user_data;
   party_id_t           party_id;
   long                 specific_event;
   char                 pool[C_DATSZ];
}rts_info_t;

typedefstruct{
   asai_common_t        asai_common;
   long                 cause_value;
}rt_nak_t;
```

Description

The client (that is, ECS) sends this capability to request route selection information, relating to a route or route again request. Use of this capability is an implicit acknowledgement of the route request. A structure of type `rts_info_t` contains the information needed to fulfill the request.

asai_common

The programmer sets capability to C_RT_SEL and primitive_type to C_REQUEST for a positive acknowledgement. The primitive_type is set to C_NEG_ACK for a negative acknowledgement. The programmer also sets the sao_id (also known as cluster_id) to identify the association of the route request.

calling_num, called_num

The calling and called extensions are specified by the null-terminated strings pointed to by these parameters. The calling extension is optional; therefore, calling_num may be a null pointer. The called extension is mandatory.

uudata

The purpose of the user-user information is to convey information between ISDN users. This information is not interpreted by ECS, but rather is carried transparently and delivered to the remote user. If the info field is IA5 characters (ASCII), the string is null-terminated.

The leng field is an integer value that indicates the number of octets of user data included in the request. This field is set to 0 if no user_user data is present. Currently, ECS will accept up to 32 bytes of data (leng = 32). If more than 32 bytes is specified, an error will be returned. ASAI supports a maximum user_user data length of 32 bytes, although user_user data generated from a PRI trunk can be up to 127 bytes in length. (DEFINITY ECS will truncate this to 32 bytes before forwarding it to ASAI link.)

The protocol field indicates the type of information and is restricted to C_UU_USER that indicates a user-specific protocol or C_UU_IA5 which indicates IA5 or ASCII characters.

The info field is a pointer to an ASCII string of no more that 32 characters in length.

ofacility

This is an optional parameter that identifies the type of facility requested. It can be C_TRUNK_GROUP, C_ACD_SPLIT or C_NUSE_LONG. These values indicate a trunk group, an ACD group or facility not used.

ofac_ext

The ofac_ext parameter is a pointer to a null-terminated character string that identifies the trunk access code of the trunk group or the ACD split extension, depending on the value of ofacility.

priority_call

Setting this parameter to C_PRIOR_ON indicates that the call should be tagged as a priority call; otherwise, the parameter should be set to C_PRIOR_OFF.

dir_agtcall

Setting this parameter to C_DIRAGT_ON indicates that the call should be tagged as a direct agent call; otherwise, the parameter should be set to C_DIRAGT_OFF.

user_data

user_data provides the user information that will be sent to ECS.

type indicates the type of user code entered. The type field is restricted to the values C_TONE_DETECTOR and C_ADJUNCT_DIGITS. If C_ADJUNCT_DIGITS is specified, the time-out field is forced to zero(0) and the collect field is forced to C_COLLECTED. If C_TONE_DETECTOR is specified, the collect field is forced to C_COLLECT.

The timeout parameter specifies the digit collection time out and is an integer value from 0 to 63 (default is 0).

The value of digits depends on the flag specified by the collect field. If the collect field has the value C_COLLECTED, digits is a pointer to an ASCII string. If the collect field has the value C_COLLECT, digits is a single one-byte character containing a binary integer indicating the maximum number of digits to be collected (default is 0).

party_id

party_id is optional. party_id specifies the party generating the alerting event.

NOTE:
party_id is *not* an ASCII null-terminated string but rather an array of binary bytes of the specified length. See ''Identifiers'' on page 8-5 in Chapter 8, ''ASAI Capability Primitives'' for further information.

specific_event

specific_event is a long integer that indicates the call prompter disconnect condition and indicates either connect or drop.

pool

The pool parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See "Pool'' on page 8-15 in Chapter 8, ''ASAI Capability Primitives'' for more information.

Return Value

When this capability is used in an `asai_send()`, the function returns a nonnegative value upon success and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a server.

See Also

`asai_send()`

`C_RT_REQ C_RT_END`

C_SV_CONF (3ASAI)

Name

C_SV_CONF — Acknowledge a set value request

Type

Terminating — Unacknowledged

Usage

```
typedef struct {
         asai_common_t       asai_common;
         long                cause_value;
}sv_nak_t;

typedef union {
         asai_comon_t        asai_common;
         sv_nak_t            sv_nak;
}sv_rsp_t;
```

Description

The client receives this capability as a positive or negative acknowledgment to a set value request. The union `sv_resp` contains the necessary information for both.

Within `asai_common`, the `capability` is returned as `C_SV_CONF` and `primitive_type` is returned as `C_POS_ACK` for positive acknowledgment and `C_NEG_ACK` for negative acknowledgement. The `sao_id` (also known as `cluster_id`) of the association specified in the set value request is also returned.

cause_value

This parameter can be any of the following:

```
C_FACUNSUB
C_INVLDIE
C_MAND_INFO
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL
```

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an `asai_rcv()`, the function returns the size of the message when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that no data has been received at this time.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

`asai_rcv()`

`C_SV_REQ`

C_SV_REQ (3ASAI)

Name

C_SV_REQ — Set the value of an item in the domain of the provider

Type

Initiating — Acknowledged

Usage

```
typedef struct {
        asai_common_t            asai_common;
        long                     sv_item;
        long                     mwi_item;
        char                     called_num;
        char                     pool [C_DATSZ];
}sv_mwi_val_t;

typedef struct {
        sv_common_t              sv_common;
        call_id_t                call_id;
        long                     bill_type;
        float                    bill_rate;
        char                     pool [C_DATSZ];
}sv_flex_t;

typedef struct {
        asai_common_t            asai_common;
        long                     sv_item;
}sv_common_t;

typedef Union {
        sv_common_t              sv_common;
        sv_mwi_val_t             sv_mwi_val;
        sv_flex_t                sv_flex;
}sv_buf_t;
```

Description

The client sends this capability to request that the value of specified items be set.These requests are initiated from the adjunct; the time of the day can be set only by ECS.

Within the union sv_buf_t, the parameters contained in the sv_common member define a set of common parameters needed to fulfill the request. Specific item parameters are contained in the sv_mwi_val or sv_flex members.

asai_common

Within this structure, the programmer sets the capability to C_SV_REQ and the primitive_type is set to C_REQUEST.The programmer also sets the sao_id (also known as cluster_id).

The parameter sv_item defines the domain of the items to be set. This parameter can be C_SV_MWI (message waiting indicator) or C_SV_FLEX (flexible billing).

sv_mwi_val

When sv_item is set to C_SV_MWI, the sv_mwi_val union member indicates the specific item parameters. The called_num parameter is a pointer to a null-terminated string containing the extension of the end point for which the indicator is to be set. mwi_type indicates whether the light should be on or off (C_MWI_ON or C_MWI_OFF).

sv_flex

When the sv_item is set to C_SV_FLEX the sv_flex union member indicates the specific item parameters. Call_id is ECS-assigned identifier of the call to which the billing rate is to be applied. The value of this parameter is a character array of length id_length, pointed to by id_ptr. The bill_type parameter specifies the rate treatment for the call and can be one of NEW_RATE, FLAT_RATE, PREM_CHARGE, PREM_CREDIT, or FREE_CALL. The bill_rate parameter is a floating point number that specifies the rate according to the treatment indicated by bill_type.

NOTE:
The bill_rate is not sent to ECS when the bill_type is FREE_CALL. The pool parameter is a character array that provides buffer space for the ASAI library to store program-specific information. See “Pool’’ on page 8-15 in Chapter 8, “ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an asai_send(), the function returns a nonnegative value when successful and -1 on failure. A return value of 0 does not indicate an error, it merely indicates that there is no data to be sent.

Errors

A C_SV_MWI request will return C_BADPMATCH if the called_num parameter is not set. A C_SV_FLEX request will return C_BADVALUE if the bill_rate parameter is greater than 999.99 or a precision higher than the hundredths is specified (for example, 127.312). See Chapter 7, ''Error Messages'' for a list of error messages with brief explanations of their probable causes.

See Also

asai_send()

C_SV_CONF

C_VQ_CONF (3ASAI)

Name

C_VQ_CONF — Acknowledge and terminate a value query request

Type

Terminating — Unacknowledged

Usage

```
typedef struct{
        asai_common_t        asai_common;
        long                 vqc_type;
        long                 cause_value;
}vqc_cause_t;

typedef struct{
        asai_common_t        asai_common;
        long                 vqc_type;
        long                 num_login;
        long                 num_avail;
        long                 num_callinque;
}vqc_split_t;

typedef struct{
        asai_common_t        asai_common;
        long                 vqc_type;
        long                 num_avail;
        long                 num_inuse;
}vqc_cc_stat_t;

typedef struct{
        asai_common_t        asai_common;
        long                 vqc_type;
        long                 num_avail;
        long                 num_inuse;
}vqc_tg_stat_t;

typedef struct{
        asai_common_t        asai_common;
        long                 vqc_type;
        long                 year;
        long                 month;
        long                 day;
        long                 hour;
        long                 minute;
        long                 second;
```

```
}vqc_time_t;

typedef struct{
        asai_common_t           asai_common;
        long                    vqc_type;
        long                    talk_state;
        long                    work_mode;
        long                    reason_code;
}vqc_agt_stat_t;

typedef struct{
        asai_common_t           asai_common;
        long                    vqc_type;
        long                    ext_status;
}vqc_station_t;

typedef struct{
        asai_common_t           asai_common;
        long                    vqc_type;
        long                    num_call;
        stn_info_t              *stninfo_list;
        char                    pool[C_DATSZ];
}vqc_cal_stat_t;

typedef struct{
        asai_common_t           asai_common;
        long                    vqc_type;
        long                    num_party;
        party_ext_t             *party_list;
        char                    pool[C_DATSZ];
}vqc_pty_stat_t;

typedef struct{
        asai_common_t           asai_common;
        long                    vqc_type;
        long                    domain_type;
        long                    station_type;
        long                    domain2_type;
        char                    *extension;
        char                    pool[C_DATSZ];
}vqc_ext_stat_t;

typedef struct{
        asai_common_t           asai_common;
        long                    vqc_type;
        long                    mwi_status;
}vqc_mwi_stat_t;

typedef struct{
        asai_common_t           asai_common;
```

```
        long                    vqc_type;
        long                    sac_status;
}vqc_sac_stat_t;
typedef struct{
        asai_common_t           asai_common;
        long                    vqc_type;
        long                    scf_status;
        char                    *redirection;
        plan_type_t             redir_type;
        char                    pool[C_DATSZ];
}vqc_scf_stat_t;

typedef struct{
        asai_common_t           asai_common;
        long                    vqc_type;
        long                    domain_type;
        char                    *extension;
        char                    *name;
        wchar_t                 *w_name;
        char                    pool[C_DATSZ];
} vqc_aaid_t;

typedef struct{
        asai_common_t           asai_common;
        long                    vq_item;
        ucid_t                  ucid;
        char                    pool[C_DATSZ];
}vqc_ucid_t;                                   /* UCID G3V6 */

typedef struct{
        asai_common_t           asai_common;
        long                    vqc_type;
}vqc_common_t;

typedef union{
        vqc_common_t            vqc_common;
        vqc_cause_t             vqc_cause;
        vqc_split_t             vqc_split;
        vqc_cc_stat_t           vqc_cc_stat;
        vqc_tg_stat_t           vqc_tg_stat;
        vqc_time_t              vqc_time;
        vqc_agt_stat_t          vqc_agt_stat;
        vqc_station_t           vqc_station;
        vqc_cal_stat_t          vqc_cal_stat;
        vqc_pty_stat_t          vqc_pty_stat;
        vqc_ext_stat_t          vqc_ext_stat;
        vqc_mwi_stat_t          vqc_mwi_stat;
        vqc_sac_stat_t          vqc_sac_stat;
        vqc_scf_stat_t          vqc_scf_stat;
        vqc_aaid_t              vqc_aaid;
```

```
        vqc_ucid_t           vqc_ucid;      /* UCID 63V6 */
}vqc_buf_t;
```

Description

The client receives this capability as a positive or negative acknowledgement that terminates a value query request. The information needed to service the request is contained in the members of the union `vqc_buf.`

The data common to all termination requests is contained in `vqc_common`. Within `asai_common,` the `capability` is returned as `C_VQ_CONF` and the `primitive_type` as `C_POS_ACK` for a positive acknowledgment and `C_NEG_ACK` for as a negative acknowledgement. The association identified by the `sao_id` (also known as `cluster_id`) of the value query request is also returned.

In a positive acknowledgment, the capability returns information in one of the following `vqc_buf` members: `vqc_split, vqc_cc_stat, vqc_tg_stat, vqc_time, vqc_agt_stat, vqc_station, vqc_common, vqc_common, vqc_cal_stat, vqc_pty_stat, vqc_ext_stat, vqc_mwi_stat, vqc_sac_stat, vqc_scf_stat, vqc_aaid, or vqc_ucid_t.`

Within the `vqc_common` structure, `vqc_type` indicates the type of item to which the information supplied pertains. This parameter may assume the values of `C_ACDSPT_STAT, C_CLASS_STAT, C_TG_STAT, C_TIME_OF_DAY, C_AGT_STAT, C_STATION, C_AGTLOGIN_AUD C_PTY_STAT, C_EXT_STAT, C_MWI_STAT, C_SAC_STAT, C_SCF_STAT, C_AAID, or C_UCID`. (ACD split status, call classifier status, trunk group status, parties on a call, extension information, time of day, ACD agent status, station status, third party call audit, agent login audit, message waiting indication, send all calls, station call forwarding, name associated with the extension number, or Universal Call ID.)

For third party call audits and agent login audits, data comes in a variable number of `C_VQ_RESP` messages and this message is used to indicate the end of the list messages.

The following table shows the item type, associated `vqc_buf` member and a description of the information contained in the member. Mandatory information is indicated with an asterisk(*). Optional items show a value for `C_NUSE_LONG`.

Table 9-4. Associated Items and Descriptions for C_VQ_CONF

Item Type	vqc_buf **Member**	Item and Description
C_ACDSPT_STAT	vqc_split	num_login is an integer value indicating the number of agents logged in (C_NUSE_LONG if not provided).
		num_avail is an integer value indicating the number of agents available.
		num_callinque is an integer value indicating the number of calls in the queue (C_NUSE_LONG if not provided).
C_CLASS_STAT	vqc_cc_stat	num_avail is an integer value indicating the number of idle call classifiers.
		num_inuse is an integer indicating the number of call classifiers in use (C_NUSE_LONG, if not provided).
C_TG_STAT	vqc_tg_stat	num_avail is an integer value indicating the number of available trunks in the group.
		num_inuse is an integer indicating the number of trunks in use (C_NUSE_LONG if not provided).
C_TIME_OF_DAY	vqc_time	year is specified by a binary quantity that is specified by 2 digits.
		month is specified by a binary quantity that is specified by 2 digits.
		day is specified by a binary quantity that is specified by 2 digits.
		hour is specified by a binary quantity that is specified by 2 digits.
		minute is specified by a binary quantity that is specified by 2 digits.
		second is specified by a binary quantity that is specified by 2 digits.
C_AGT_STAT	vqc_agt_stat	talk_state can be C_ON_CALL or C_IDLE.
		work_mode can be C_MANUAL_IN, C_AUTO_IN, C_AFTCAL_WK, or C_AUX_WORK. reason_code can be an integer from 1 to 9 indicating the reason why the agent is in auxiliary mode.

Table 9-4. Associated Items and Descriptions for C_VQ_CONF — *Continued*

Item Type	vqc_buf **Member**	Item and Description
C_AAID	vqc_aaid	Item and Description domain_type is an integer value indicating the type of the entity queried, (for example, Station, VDN, etc.).
		extension is a pointer to a null-terminated string indicating the name associated with the extension number supplied in the request.
		name is a point to a null-terminated string indicating the name associated with the extension in the Integrated Directory Database. This name will be from 1 to 27 ASCII characters in length. If the name is not found, any of the optional elements of the RElease COMplete query response message will not be returned. w_name points to a translation of the raw OPTREX data in the name into its locale equivalent ISO (four bytes) characters. (See optrex, locale, and special characters in the Glossary and Appendix A, "Special Characters" for more information.)
C_UCID	vqc_ucid	Item is an integer value indicating the type of entity queried.
		call_id is a structure in which the parameter id_length specifies the length of ECS-assigned call identifier and id_ptr points to an array of binary bytes that specifies that call identifier.
		pool is a character array that provides buffer space for the ASAI library to store program-specific information. See ‘‘Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.
C_CAL_STAT	vqc_cal_stat	stn_info is a pointer to a list of structures of type stn_info_t. Each structure contains a structure call_id that specifies ECS-assigned call identifier values and length and a structure party_id that specifies ECS-assigned party identifier values and length and the current call state of the party, pty_state. When no call information is returned for this capability, stninfo_list is a null pointer.
		num_call is the number of structures in the list pointed to by stn_info. Whenever no call identifiers are returned for this capability, it assumes the value of C_NUSE_LONG.
		pool is a character array that provides buffer space for the ASAI library to store program-specific information. See ‘‘Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

Table 9-4. Associated Items and Descriptions for C_VQ_CONF — *Continued*

Item Type	vqc_buf **Member**	Item and Description
C_PTY_STAT	vqc_pty_stat	party_list is a pointer to a list of structures associated with the same call identifier. Each structure contains the extension number and its associated structure, party_id_t. party_id_t contains the parameter id_ptr that points to a character array specifying ECS-assigned party identifier values. The length of each party identifier is given by id_length. When no party identifiers are returned for this capability, party_list is a null pointer.
		num_party is the number of structures specified in the list pointed to by party_list. When no party identifiers are returned for this capability, num_party assumes the value of C_NUSE_LONG.
		pool is a character array that provides buffer space for the ASAI library to store program-specific information. See ‘‘Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.
C_EXT_STAT	vqc_ext_stat	domain_type can be C_ACD_SPLIT, C_VDN, C_ANNOUNCE, C_DATA, C_VOICE, C_ASAI_LINK, C_LOGICAL, or, if no value is returned for this capability, C_NUSE_LONG.
		station_type can be C_ANALOG, C_BRI, C_PROPRIETARY, or C_NUSE_LONG.
		domain_type2 can be C_LOGICAL or 0.
		Extension — In an EAS environment, if an agent's login is queried and the agent is logged in, the extension will be returned. If a physical extension is queried and the agent is logged in, the extension will be returned.
		pool — See vqc_pty_stat.
C_MWI_STAT	vqc_mwi_stat	mwi_status can be C_MWI_ON or C_MWI_OFF.
C_SAC_STAT	vqc_sac_stat	sac_status can be C_SAC_ON or C_SAC_OFF.

Table 9-4. Associated Items and Descriptions for C_VQ_CONF — *Continued*

Item Type	vqc_buf Member	Item and Description
C_SCF_STAT	vqc_scf_stat	scf_status can be C_SCF_ON or C_SCF_OFF.
		redirection is a pointer to a null-terminated string specifying the redirection number if scf_status is C_SCF_ON.
		redir_type is an optional plan_type_t structure that supplies additional information about the structure field that immediately precedes it (providing that the preceding field is not NULL). plan_type_t has two fields addr_type and numb_plan. addr_type can have two possible values unknown(0) and international(1). Currently, numb_plan can have only one value, unknown(0). However, additional values for both fields may be coded in the future.
C_STATION	vqc_station_t	ext_status can be C_ON_CALL or C_IDLE.
		pool is a character array that provides buffer space for the ASAI library to store program-specific information. See ‘‘Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

cause_value

This parameter can be any of the following:

```
C_FACUNSUB
C_INVLDIE
C_INVLDNUM
C_MAND_INFO
C_NOLOGIN
C_NOSPLIT_MEM
C_PROTERR
C_RESUNAVL
C_SER_UNIMP
C_SERV_UNAVIL
```

See Table 8-1 on page 8-12 for further information on cause values.

Return Value

When this capability is used in an asai_rcv(), the function returns the size of the message upon success and -1 on failure. A return value of 0 does not indicate an error, but that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a server.

See Also

`asai_rcv()`

`C_VQ_REQ C_VQ_RESP`

C_VQ_REQ (3ASAI)

Name

C_VQ_REQ — Request information about status or value of a service or feature within the domain of the provider

Type

Initiating — Acknowledged

Usage

```
typedef struct{
      asai_common_t                 asai_common;
      long                          vq_item;
      char                          *split_ext;
      char                          pool[C_DATSZ];
}vq_split_stat_t;

typedefstruct{
      asai_common_t                 asai_common;
      long                          vq_item;
      char                          *tg_ext;
      char                          pool[C_DATSZ];
}vq_tg_stat_t;

typedef struct{
      asai_common_t                 asai_common;
      long                          vq_item;
      char                          *split_ext;
      char                          *agt_ext;
      char                          pool[C_DATSZ];
}vq_agt_stat_t;

typedef struct{
      asai_common_t                 asai_common;
      long                          vq_item;
      char                          *extension;
      char                          pool[C_DATSZ];
}vq_station_t;

typedef struct{
      asai_common_t                 asai_common;
      long                          vq_item;
      char                          *split_ext;
      char                          pool[C_DATSZ];
}vq_agt_aud_t;
```

```
typedef struct{
      asai_common_t                  asai_common;
      long                           vq_item;
      char                           *extension;
      char                           pool[C_DATSZ];
}vq_cal_stat_t;

typedef struct{
      asai_common_t            asai_common;
      long                     vq_item;
      call_id_t                call_id;
      char                     pool[C_DATSZ];
}vq_pty_stat_t;

typedef struct{
      asai_common_t            asai_common;
      long                     vq_item;
      char                     *extension;
      char                     pool[C_DATSZ];
}vq_ext_stat_t;

typedef struct{
      asai_common_t            asai_common;
      long                     vq_item;
      char                     *extension;
      char                     pool[C_DATSZ];
}vq_mwi_stat_t;

typedef struct{
      asai_common_t            asai_common;
      long                     vq_item;
      char                     *extension;
      char                     pool[C_DATSZ];
}vq_sac_stat_t;

typedef struct{
      asai_common_t            asai_common;
      long                     vq_item;
      char                     *extension;
      char                     pool[C_DATSZ];
}vq_scf_stat_t;

typedef struct{
      asai_common_t            asai_common;
      long                     vq_item;
      long                     domain_type;
      char                     *extension;
      char                     pool[C_DATSZ];
}vq_aaid_t;
```

```
typedef struct{
     asai_common_t          asai_common;
     long                   vq_item;
     call_id_t              call_id;
     char                   pool[C_DATSZ];
}vq_ucid_t;                                    /* 63V6 */

typedef struct{
     asai_common_t          asai_common;
     long                   vq_item;
}vq_common_t;

typedef union{
     vq_common_t            vq_common;
     vq_split_stat_t        vq_split_stat;
     vq_tg_stat_t           vq_tg_stat;
     vq_agt_stat_t          vq_agt_stat;
     vq_station_t           vq_station;
     vq_agt_aud_t           vq_agt_aud;
     vq_cal_stat_t          vq_cal_stat;
     vq_pty_stat_t          vq_pty_stat;
     vq_ext_stat_t          vq_ext_stat;
     vq_mwi_stat_t          vq_mwi_stat;
     vq_sac_stat_t          vq_sac_stat;
     vq_scf_stat_t          vq_scf_stat;
     vq_aaid_t              vq_aaid_req;
     vq_ucid_t              vqc_ucid_req;  /* G3V6 */
     }vq_buf_t;
```

Description

The client sends this capability to request information concerning a specified item. The information needed to fulfill the request is contained in the members of the `vq_buf` union.

The data common to all requests is contained in `vq_common`. The programmer sets the `capability` to `C_VQ_REQ` and the `primitive_type` to `C_REQUEST`. The programmer also sets the `sao_id` (also known as `cluster_id`).

The parameter `vq_item` identifies the particular item for which information is requested. It can be `C_ACDSPT_STAT, C_CLASS_STAT, C_TG_STAT, C_TIME_OF_DAY, C_AGT_STAT, C_STATION, C_AGTLOGIN_AUD, C_CAL_STAT, C_PTY_STAT, C_EXT_STAT, C_MWI_STAT, C_SAC_STAT , C_SCF_STAT, C_AAID,` or `C_UCID.`(ACD split status, call classifier status, trunk group status, time of day, ACD agent status, station status, agent login audit, call at a station, parties on a call, extension status, message waiting indication, send all calls status, station call forwarding, name associated with the extension number, or Universal Call ID.)

The parameters contained in the vq_buf union members vq_split_stat, vq_common,vq_tg_stat,vq_common,vq_agt_stat,vq_station, vq_agt_aud,vq_cal_stat,vq_pty_stat,vq_ext_stat,vq_mwi_stat, vq_sac_stat, and vq_scf_stat supply mandatory and optional information concerning the requested item.

The following table shows the vq_item value, the associated vq_buf union member, the information contained in the member, and a description of the information. Mandatory information is indicated with an asterisk (*).

Table 9-5. Associated Items and Descriptions for C_VQ_REQ

Item Type	vq_buf Member	Description
C_ACDSPT_STAT	vq_split_stat	split_ext is a pointer to a null-terminated string specifying the ACD split extension number or the split number.
		pool is a character array that provides buffer space for the ASAI library to store program-specific information. See "Pool" on page 8-15 in Chapter 8, "ASAI Capability Primitives" for more information.
C_TG_STAT	vq_tg_stat	tg_ext is a pointer to a null-terminated string specifying the trunk group access code or extension.
		pool. See C_ACDSPT_STAT.
C_AGT_STAT	vq_agt_stat	split_ext. See C_ACDSPT_STAT.
		agt_ext is a pointer to a null-terminated string specifying the extension of the ACD agent.
		pool. See C_ACDSPT_STAT.
C_STATION	vq_station	extension is a pointer to a null-terminated string specifying the end point extension number.
		pool. See C_ACDSPT_STAT.
C_AGTLOGIN_AUD	vq_agt_aud	split_ext. See C_ACDSPT_STAT.
		pool. See C_ACDSPT_STAT.
C_CAL_STAT	vq_cal_stat	extension. See C_STATION.
		pool. See C_ACDSPT_STAT.
C_PTY_STAT	vq_pty_stat	call_id is a structure in which the parameter id_length specifies the length of ECS-assigned call identifier and id_ptr points to a character string that specifies that call identifier.
		pool. See C_ACDSPT_STAT.
C_EXT_STAT	vq_ext_stat	extension. See C_STATION.
		pool. See C_ACDSPT_STAT.
C_MWI_STAT	vq_mwi_stat	extension. See C_STATION.
		pool. See C_ACDSPT_STAT.

Table 9-5. Associated Items and Descriptions for C_VQ_REQ — *Continued*

Item Type	vq_buf Member	Description
C_SAC_STAT	vq_sac_stat	extension. See C_STATION.
		pool. See C_ACDSPT_STAT.
C_SCF_STAT	vq_scf_stat	extension. See C_STATION.
		pool. See C_ACDSPT_STAT.
C_CLASS_STAT	none	(no additional items)
C_TIME_OF_DAY	none	(no additional items)
C_AAID		domain_type is a pointer to a null-terminated string specifying the ACD split extension number or the split number.
		extension is a pointer to a null-terminated string specifying the end point extension number.
		pool. See C_ACDSPT_STAT.
C_UCID		call_id is a structure in which the parameter id_length specifies the length of ECS-assigned call identifier and id_ptr points to a character string that specifies that call identifier.
		pool. See C_ACDSPT_STAT.

Return Value

When this capability is used in an asai_send(), the function returns a nonnegative value when successful and -1 on failure. A return value of 0 indicates that no data has been sent.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

See Also

asai_send()

C_VQ_RESP C_VQ_CONF

C_VQ_RESP (3ASAI)

Name

C_VQ_RESP — Respond to a value query request

Type

Continuing — Unacknowledged

Usage

```
typedef struct{
      asai_common_t          asai_common;
      long                   vqr_item;
      char                   *agtext_list;
      plan_type_t            agtext_type;
      long                   num_ext;
      char                   pool[C_DATSZ];
}vqr_agtaud_t;

typedef struct{
      asai_common_t          asai_common;
      long                   vqr_item;
}vqr_common_t;

typedef union{
      vqr_common_t           vqr_common;
      vqr_agtaud_t           vqr_agtaud;
}vqr_buf_t;
```

Description

The client receives this capability in response to a value query request.The requested information is returned in the members of the union `vqr_buf`.

The information common to value query responses is contained in `vqr_common`. Within `asai_common`, the `capability` is returned as `C_VQ_RESP` and the `primitive_type` as `C_REQUEST`. The association identified by the `sao_id` (also known as `cluster_id`) of the request is also returned.

The parameter `vqr_item` indicates the type of item to which the information supplied pertains. It can be `C_AGTLOGIN_AUD` (agent login).The remaining `vqr_buf` union members supply specific information based on the item identified in the response by `vqr_item`. The following table shows the item type, associated `vqr_buf` union member, the information contained in the member, and a description of the information. All items are mandatory.

Table 9-6. Associated Items and Descriptions for C_VQ_RESP

Item Type	vqr_buf **Member**	Description
C_AGTLOGIN_ AUD	vqr_agtaud	agtext_list is a pointer to a list of agent extension numbers where each extension is specified by a null-terminated character string.
		num_ext specifies the number of extensions in the list pointed to by agtext_list.
		pool is a character array that provides buffer space for the ASAI library to store program-specific information. See ‘‘Pool’’ on page 8-15 in Chapter 8, ‘‘ASAI Capability Primitives’’ for more information.

Return Value

When this capability is used in an asai_rcv(), the function returns the size of the message when successful and -1 on failure. A return value of 0 indicates that no data has been received.

Errors

See Chapter 7, ‘‘Error Messages’’ for a list of error messages with brief explanations of their probable causes.

Notes

This capability can be issued only by a server.

See Also

asai_rcv()

C_VQ_REQ C_VQ_CONF

Testing

10

Integration Test Tool

This Integration Test Tool (ITT) is designed to help test the ASAI library by exercising its function calls and optionally, by comparing the expected results to the actual results for received messages. See Figure 10-1 on page 10-1 for ITT structure.

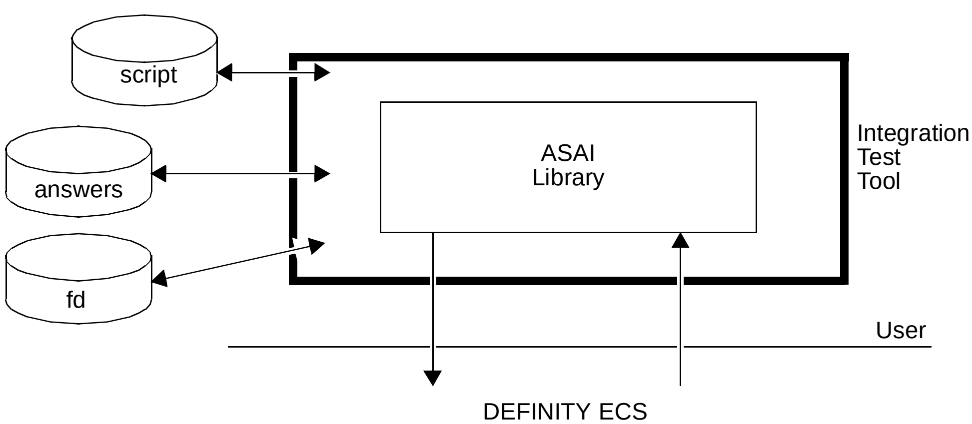


Figure 10-1. Integration Test Tool — Overview

Terminology

The following terms are used extensively throughout this chapter and are unique to the ITT:

test script — Runs the test by telling the tool which ASAI library functions to execute and which capabilities they should invoke.

capability info script — If the test tool needs additional information about the selected capability, it asks the user for that input. Alternatively, the script may contain the name of an input file that would answer all the specifics about the selected capability.

fd file — Used to convey the return value from the ASAI open function. The information is stored in a separate file, so that it can be passed to subsequent calls (represented by command lines in the test script file).

Executing the ITT Program

The following instructions assume that the installation steps have been completed successfully for the CallVisor PC ISDN, ASAI and ITT packages. Run `asai_test` to verify that the link(s) are working between the adjunct and the ECS.

The program, `asai_test`, sends an ASAI heartbeat and waits for a reply on each administered link. `asai_test` reports success if a positive acknowledgment is received.

The command, **mkclid**, creates cluster id, file descriptor and input/output files used by the script files. These files are created with the users login id and shell umask file permissions.

Enter the following commands:

cd /usr/adm/asai_itt

The ECS sends a heartbeat to the adjunct every two minutes and, after doing that three times with no response, tears down and then restarts the link. During this time `C_ABORTS` are sent to the adjunct. Scripts can be run without the `C_ABORTS` appearing by running the **asai_hb** command. Refer to the manual page for more information about the **asai_hb** command.

Enter the following command for link 1:

nohup /usr/bin/asai_hb 1 &

The following command line illustrates how to execute the ITT:

```
itt scriptfile
```

Figure 10-2. Execution of the ASAI integration test tool

Here `scriptfile` represents the name of the main script file to be interpreted by the ITT.

Scripts

This section describes the format of script and capability-info files.

The script test file is an ASCII file (created using a text editor) that tells the test tool which functions to execute and gives it the information that will allow it to gather the proper information to execute that function.

Test script files can have embedded comments, but these must appear on lines by themselves. Comment lines should start with a *sharp* (#) symbol, followed by a space and then the comments. Comment lines will appear in the trace but will not be interpreted by the test tool.

Figure 10-3 shows embedded comments:

```
#
#      Copyright (c) 1996 Lucent Technologies
#        All Rights Reserved
#
#     THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF Lucent Technologies
#      The copyright notice above does not allow for any
#      actual or intended publication of such source code.
#
#
# FILE Name:test.event
#    AUTHOR:Robert Hutchison
#   PURPOSE:this script tests the event notification features
#         of the ASAI library.
#
```

Figure 10-3.   Script File — Comments

Each command (noncomment) line in the test script file contains the following parts:

library function name	The name of the ASAI library function to be executed; (for example, `asai_open` is used to open communication with the ASAI stack and, the ECS).
return value filename	The name of a file where the return value of the function is to be stored. This is especially important for the `asai_open` call because we need to save its return value to be used as input to subsequent calls. If the return value is to be ignored, a reference to `/dev/null` will be sufficient to throw away its value.
capability-specific arguments	Other arguments on the command line are function specific and are described in detail with the particular command.

stall

This script command temporarily stops the test tool and gives instructions to the person running the test. This command can be used in demos, where you want to stop and tell about what is happening, or to give the operator instructions like “pick up extension 1600” or “dial 1601 from extension 1600.” These instructions permit the operator to test functionality such as event reports.

The format for a `stall` command:

stall instructions

`instructions` tells the operator what to do. When these instructions are printed on the screen, the operator is prompted to press the RETURN key.

asai_open

This script command causes the `asai_open()` library function to be executed. Its command line is as follows:

```
asai_open return_filename asai_special_file flags node_id server_num server_type
```

The arguments to `asai_open` are as follows:

`return_filename`	The filename of the file where the return value of the `asai_open` (a file descriptor) will be stored. The value will be put in the file in ASCII so that it will be editable as necessary. This file will probably be passed to subsequent commands wishing to refer to this stream (file descriptor).
`asai_special_file`	The filename of the ASAI provider (usually `/dev/asai/asai`), or the machine name in the case of CV/LAN.
`flags`	The numeric value of flags to be forwarded to `asai_open()`.
`node_id`	A character string referring to the node being opened. In the default environment (where the administrator uses the suggested values on IPCI installation), this would be `signal01`.
`server_num`	Informs the ASAI library software of the number of services that this open (if successful) will serve. If this open does not intend to serve anything, then both this and the next fields should be ignored.
`server_type`	Informs the ASAI library software of the service type that this open (if successful) will serve. If this open intends to serve more than one type of service, then each `server_type` should be separated by a blank and the `server_num` should match the number of service types that you entered here.

After the `asai_open()`, the node id will automatically be set by the test tool through `asai_set_env()` using the node id specified on the command `asai_open` command line. An example of an `asai_open` command line follows:

```
asai_open fd_file /dev/asai/asai 0 signal01 1 C_RT_SER
```

Here `/dev/asai/asai` is the pathname of the ASAI provider, 0 denotes the open flags passed to the open, `signal01` represents the node name (or special file name of the ISDN connection to the ECS), `1` indicates that this open (if successful) will serve one type of service, and `C_RT_SER` is the service type (routing an incoming call).

asai_close

This script command tells the integration test tool to close an open stream through the `asai_close()` library function. The format of the command line is as follows:

```
asai_close close_ret fd_value
```

This command expects two arguments:

`close_ret` — The name of a file where the return value from the `asai_close()` will be stored. If the return value is not needed or desired, then `/dev/null` would be sufficient.

`fd_value` — The name of the file that contains the file descriptor that we want to close. It would probably be the return value file of the `asai_open` command line.

asai_send

Requests the integration test tool to use the supplied information to issue an `asai_send()` library function call. Its format is as follows:

```
asai_send send_ret fd_file CAPABILITY CAP_Type clid_file cap_info
```

This command line expects six arguments described as follows:

`send_ret` — The name of a file where the return value from the `asai_send()` will be stored.

`fd_file` — The name of the file where the file descriptor is stored. This file was probably created as the return-value file of the `asai_open` command line.

`CAPABILITY` — A character string that describes the capability being requested. The value corresponds to the capability names listed in the `asaiDefs.h` header file.

`CAP_Type` — A character string describing the type of capability. Possible values are:

```
C_REQUEST
C_POS_ACK
C_NEG_ACK
```

`clid_file` — The name of the file that contains the cluster identification number for this send. This is an ASCII file that can be created through an editor and may or may not contain a terminating newline character.

`cap_info` — A file that contains capability-specific information. If this argument is omitted and the capability needs other information from the user, it will prompt him or her for that information. If this filename is specified but the file does not exist, the function will use its interactive mode, ask the questions, and record the user's responses in the specified file. If the filename is specified and the file exists, the information contained therein will be used as the detailed information for the capability.

The format of this file is capability-specific. It will, therefore, be described with the individual capabilities.

Three examples are provided in the following three subsections.

Heartbeat Request — C_HB_REQ

This capability requires no additional information. The `cap_info` argument to the `asai_send` command line is ignored if it is there.

Set Value Request — C_SV_REQ

This capability allows the user to either:

— Turn on or off a message waiting lamp for a particular phone
or

— Send a Flexible Billing rate change request. If used in the interactive mode, the conversation might look as follows:

```
      field name: sv_common.sv_item
 symbolic values: C_SV_MWI
                  C_SV_FLEX
    enter choice: C_SV_MWI
        --- message waiting indicator ---
      field name: sv_mwi_val.mwi_type
 symbolic values: C_MWI_ON
                  C_MWI_OFF
    enter choice: C_MWI_ON
      field name: sv_mwi_val.clled_num
     enter value: 5767162
```

Figure 10-4. Set Value Request

Here the emboldened values represent values that the user typed in. If recorded in a file (to be used later in regression tests), it would be stored as follows:

```
sv_common.sv_item = 131
sv_mwi_val.mwi_type = 144
sv_mwi_val.clled_num = 5767162
```

Note that although the symbols were used in the interactive mode, the actual values are stored in the `cap_info` file. The `asaidefs.h` header file stores symbolic values and their associated definitions.

NOTE:
Due to the changes in `t_sv_req` for the addition of Flexible Billing *Set Value* capability, all previous ITT set value request input scripts will have to be updated with `sv.common.sv_item` as the first line in the file. This was optional prior to this release of CallVisor.

Event Notification Request — C_EN_REQ

This capability allows the user to ask the ECS to send event reports for a specified domain or vector directory number. In its interactive mode, the interaction for sending this capability looks like:

```
    --- EVENT NOTIFICATION REQUEST ---
       asai_common.capability = 23
       asai_common.primitive_type = 0
       asai_common.cluster_id = 10
      field name: en_common.domain_type
 symbolic values: C_ACD_GROUP
                  C_CALL_VECTOR
    enter choice: C_ACD_GROUP
      field name: acd_grp_info.acd_group
     enter value: 2015551212
```

Figure 10-5. Event Notification Request

The test tool prints the information as it extracts it from the command line:

```
asai_send send_ret fd_value C_EN_REQ C_REQUEST cl_id en_parms
```

and then prompts the user for the proper capability-specific values. It then stores the answers in the file `en_parms` (the last argument on the command line) as follows:

```
en_common.domain_type = 129
acd_grp_info.acd_group = 2015551212
```

Note that although the symbol `C_ACD_GROUP` was used in the interactive mode, the actual value was stored in the file.

asai_rcv

This script command allows the user to receive messages from the ASAI provider (that is, the ECS). The script command line's format is as follows:

```
asai_rcv rcv_ret fd_file CAP CAP_Type clid_fn [ output  expected  timer]
```

The command line expects six, seven, or eight arguments described as follows:

`rcv_ret`	The name of the file where the return value from the `asai_rcv()` will be stored. If the return value is not important, you can specify `/dev/null`.
`fd_file`	The name of a file that contains the file descriptor value. It should probably be the same file that kept the return value from the call to `asai_open()`.
`CAP`	The symbolic name of the capability that you expect to receive. If it is not important which capability is to be received, then a value of * will tell the test tool to skip its validity checking of this field.
`CAP_Type`	The value of capability type (also known as `primitive_type`) is taken from the expected file.
`clid_fn`	The name of a file where the cluster identifier will be stored. If this information is not needed later, a file name `/dev/null` can be specified.
`output`	The name of a file where the output of the `asai_rcv()` will be stored. It will be an ASCII file consisting of name-value pairs. See capability-specific sections of this document for more detailed information on these files. If the `output` parameter is omitted, the information received from the ASAI provider will only be printed on the screen and will not be stored in any file.
`expected`	An optional field contains the file name that contains name-value pairs that tell the test tool what you expect to receive from the library. These name-value pairs must be in the same order and are formatted like the output file described previously. The test tool will compare these name-value pairs against the name-value pairs received from the library. If a single capability description match is not found in the ''expected'' file, an exception is printed. If multiple description matches exist and a value cannot be found, an exception is printed. If there is no "expected" file, enter `NULL` and the test tool will ignore the ''expected'' field.
`timer`	A default timer of 120 seconds will cause the test tool to exit with an `Alarm call` message displayed. To change the default value enter an integer value.

Heartbeat Confirmation — C_HB_CONF

This message is received from the ECS confirming a heartbeat request. Its output looks like the following:

```
--- HEARTBEAT CONFIRMATION ---
asai_common->capability = 24
asai_common->primitive_type = 1
asai_common->cluster_id = 10
```

Figure 10-6. Heartbeat Confirmation Message

The contents of the output file look like the following:

```
asai_common.capability = 24
asai_common.primitive_type = 1
asai_common.cluster_id = 10
```

Figure 10-7. Heartbeat Confirmation Output

Set Value Confirmation — C_SV_CONF

This message confirms a set value request. Its screen output resembles the following:

```
--- SET VALUE CONFIRMATION ---
asai_common->capability = 31
asai_common->primitive_type = 1
asai_common->cluster_id = 10
```

Figure 10-8. Set Value Confirmation Message

Event Report — C_EN_REP

This message contains event reports sent to the adjunct (PC) by the ECS.

Incoming Call Event Report

The following event report is generated when an external caller places a call to an ASAI-controlled domain.

```
     --- EVENT REPORT ---
asai_common->capability = 22
asai_common->primitive_type = 0
asai_common->cluster_id = 10
erep_common.event_name = 11
--- INCOM_CAL REPORT ---
incall_list.clling_num = 5767907
incall_list.clled_num = 1234567890
incall_list.trk.direct = 0
incall_list.trk.reserved = 0
incall_list.domain_type = 1
incall_list.val_length = 10
incall_list.domain_val = 1234567890
incall_list.leng_udata = 23
incall_list.udata_type = 5
incall_list.incomg_udata = display stuff goes here
incall_list.priority_level = 3
incall_list.interflow_type = 2
incall_list.hour = 0
incall_list.minute = 0
incall_list.second = 0
incall_list.dnis_chars =
```

Figure 10-9. Incoming Call Event Report

For explanations of each field's meaning, see Chapter 9, ''Programming Manual Pages.''

```
     --- EVENT REPORT ---
asai_common->capability = 22
asai_common->primitive_type = 0
asai_common->cluster_id = 10
ev_rpt_ptr->erep_common.event_name = 0
--- ALERTING REPORT ---
ev_rpt_ptr->alert_list.connect_num = 1234567890
ev_rpt_ptr->alert_list.clling_num = 1234567890
ev_rpt_ptr->alert_list.dial_num = 1234567890
```

Figure 10-10. Call Alerting

```
     --- EVENT REPORT ---
asai_common->capability = 22
asai_common->primitive_type = 0
asai_common->cluster_id = 10
ev_rpt_ptr->erep_common.event_name = 1
--- CONNECTED REPORT ---
ev_rpt_ptr->connect_list.connect_num = 1234567890
ev_rpt_ptr->connect_list.clled_num = 1234567890
```

Figure 10-11. Connected Report

```
     --- EVENT REPORT ---
asai_common->capability = 22
asai_common->primitive_type = 0
asai_common->cluster_id = 10
ev_rpt_ptr->erep_common.event_name = 4
--- DROP REPORT ---
ev_rpt_ptr->drop_list.drop_cause = 25
ev_rpt_ptr->drop_list.drop_type = 1
ev_rpt_ptr->drop_list.droped_num = 1234567890
```

Figure 10-12. Drop Report

```
     --- EVENT REPORT ---
asai_common->capability = 22
asai_common->primitive_type = 0
asai_common->cluster_id = 10
ev_rpt_ptr->erep_common.event_name = 12
--- CALLEND REPORT ---
ev_rpt_ptr->calend_list.calend_cause = 25
```

Figure 10-13. Call End Report

Manual Pages

The following manual pages describe the utilities available to test the tool for the programmers.

close_dialog_out(3)

Name

`close_dialog_out`

Synopsis

`close_dialog_out()`

Description

Closes the temporary dialog-recording file opened by `open_dialog_out`. It knows the **FILE** pointer used for the file because the open function stored it in a static variable.

Return Value

Returns SUCCESS or causes the program to exit with a status of 16.

get_long(3)

Name

`get_long`

Synopsis

```
long get_long( fieldname, cap_info )
char                *fieldname;
struct cap_info     *cap_info;
```

Description

Obtains the current value of the specified field from the capability information structure if that value is of type `long`. The `cap_info` argument can either be the address of the start of the table or the address of the proper entry itself.

Return Value

Returns the long value requested or exits with a status of 19 if the value stored in the table is not a long. This indicates that the capability information structure was incorrectly set up.

get_string(3)

Name

`get_string`

Synopsis

```
char *get_string( fieldname, cap_info )
char                *fieldname;
struct cap_info     *cap_info;
```

Description

Obtains the current value of the specified field from the capability information structure if that value is a string. The `cap_info` argument can either be the address of the start of the table or the address of the proper entry itself.

Return Value

Returns the pointer to the string requested or exits the program with a status of 20 if the value stored in the table is not a string. This indicates that the capability information structure was incorrectly set up.

open_dialog_out(3)

Name

`open_dialog_out`

Synopsis

```
open_dialog_out(filename)
char        *filename;
```

Description

Opens the file whose name is pointed to by its argument. If unsuccessful, it causes the program to exit with a value of 15. It stores the FILE pointer in a static variable, allowing `append_dialog()` and `close_dialog_out()` to access the file directly.

Return Value

Returns the defined value of `SUCCESS`.

parse(3)

Name

parse

Synopsis

```
int parse( prim, nid, clid, batch_ans, ans_len, ans_filename
)
long        *prim;
char        *nid;
long        *clid;
char        *batch_ans;
int         ans_len;
char        *ans_filename;
```

Description

Parses the remainder of a script file command line and returns the operation mode of the function based on the information on the command line. It stores the various information elements in the variables (and arrays) whose addresses are supplied to the function.

The meanings of its arguments are defined in the following table:

Table 10-1. Argument Types and Descriptions

ARGUMENT	TYPE	MEANING
prim	long *	address of long where primitive type is stored
nid	char *	address of char array where node id is copied
clid	long *	address of long where clid is stored
batch_ans	char **	address of table of pointers to character strings where batch answers will be stored
ans_len	int	length of previous table
ans_filename	char *	address of char array where batch answers may be written if we are in INTERACTIVE mode

Return Value

Returns the operation mode of the capability-specific function based on how much information was given on the command line. The return values are defined as follows:

Table 10-2. Return Value Descriptions

Return Value	Meaning
`INTERACTIVE`	the user should be prompted for capability-specific information
`RECORD_DIALOG`	a file name has been specified to record the dialog
`SCRIPT_MODE`	a file name was supplied that contained answers to the dialog-specific questions

print_common(3)

Name

```
print_common
```

Synopsis

```
print_common( common_ptr )
asai_common_t        *common_ptr;
```

Description

`printf` the values of each of the elements of the ASAI common message header to ***stdout***.

Return Value

Returns the defined value of `SUCCESS`. This return value is usually ignored by the calling function.

t_input(3)

Name

`t_input`

Synopsis

```
t_input( fieldname, cap_info, mode, batch_value )
char                 *field_key;
struct cap_info      *cap_info;
int                  mode;
char                 *batch_value;
```

Description

If the mode is `INTERACTIVE`, this function displays the name of the field, gives choices (if they were defined in the choices tables set up by the programmer) and accepts input.

If the `RECORD_DIALOG` bit is on (in the mode), it will append the results to the current dialog file. If the mode is equal to `SCRIPT_MODE`, it is assumed that the value of `batch_value` is the ASCII representation to be used. The ASCII string is converted as necessary and stored in the table's `value` field.

Arguments to this function are described as follows:

Table 10-3. Argument Types and Descriptions

Argument	Meaning
`field_name`	name of the field being entered
`cap_info`	starting address of the `cap_info` table for this capability
`mode`	operation mode - returned by call to `C_W parse()`
`batch_value`	if the mode is `SCRIPT_MODE`, then this is a pointer to a character string containing the ASCII value of the parameter

Return Value

Returns the address of the capability information structure for this field.

t_output(3)

Name

`t_output`

Synopsis

```
t_output( fieldname, expected_filename, type, value, mode )
char        *fieldname;
char        *expected_filename;
int         type;
char        *value;
int         mode;
```

Description

`t_output` is called by read-side capability-handling functions. It serves three purposes:

1. prints received value to standard output;
2. records name-value pair in specified file;
3. compares received value with the one recorded in the expected answers file.

The `mode` field is used to determine which of the above three is appropriate.

Return Value

The return value is normally `SUCCESS`.

CV/LAN Testing

Overview

A modified version of ITT called `itt` is provided to test the CV/LAN client API, the CV/LAN server and the ASAI library.

After installation of the CV/LAN client software for Solaris X86 and UnixWare, `itt` can be found in the `/usr/adm/asai_itt directory`. For SPARC Solaris, it is in `/usr/adm/asai/asai_itt` directory. For Windows NT 4.0 system, it is under the CV/LAN client program group.

Scripts

The ITT scripts can be used with `itt`. These scripts will require only a slight modification at the beginning of `asai_open` statement. The statement beginning ITT scripts is as follows:

```
asai_open return_filename asai_special_file flags node_id
server_num server_type
```

All the arguments to `asai_open` remain the same except for `asai_special_file`. Since the CV/LAN client needs to establish connection to the ECS through the CV/LAN server, the `asai_special_file` must point to the server (its IP address) and the ASAI link to the ECS. The format used by CV/LAN client API is: IP address or machine name.

Example:

```
WWW.XXX.YYY.ZZZ
```

Troubleshooting

11

This chapter presents a variety of issues or problems that you may encounter during installation, administration, maintenance, or application programming phases of ASAI application development using the CallVisor ASAI product.

PC/ISDN Platform Installation and Reference is the document accompanying the ISDN personal computer interface — the BRI interface board (IPCI board). Some of the sections are very DOS oriented. However, the document is helpful for specific installation instructions such as:

- How to install multiple IPCI cards
- IPCI-card ECS settings for addresses in PC memory space that may be used for the card
- Modifications that are required for the installation of additional RAM on the IPCI card

ECS Administration

When administering the DEFINITY ECS BRI line to be used for ASAI connectivity between the ECS and the adjunct computer, enter the command **add station *ext_num***, where ***ext_num*** is the extension number associated with the ASAI link. Then enter the precise values listed in Chapter 1, ‘‘Installation’’ of this document.

Other Release 6 administration commands that may be useful include:

list station	To find an extension number that has been administered as ‘‘type” ASAI. For example, the extension number of the ASAI link may be needed for a ‘‘route to adjunct” step in a vector.
change vector *vect_num*	To administer vector steps for routing applications where ***vect_num*** is the number of the vector being administered.
display cor *rstrict_num*	The Direct Agent Calling field must be set to **y** on the class of restriction (COR) form for a station’s COR for that station to originate or receive direct agent calls through an adjunct. ***rstrict_num*** is the COR number.

Message Trace Capability

Use `isdn_trace` to write the actual messages (in hexadecimal format) that are coming from or going to the IPCI device to `stdout`.

IPCI_ON

If `ipci_on` fails, an error message is written to `stderr`. A list of the error codes with explanations is in `/usr/adm/isdn/log_msgs`.

Application Programming

Cause Values

CallVisor ASAI product must be installed to access the complete set of correct cause values. For a complete description of the DEFINITY ECS and ASAI cause values, see the section “Cause” in Chapter 4, “Information Elements” in *DEFINITY Enterprise Communications Server Release 6 CallVisor ASAI Protocol Reference*; also see ‘‘cause_value’’ on page 8-12 in Chapter 8, ‘‘ASAI Capability Primitives’’ of this document.

Nonblocking I/O

Some applications must be implemented as ‘‘nonblocking I/O.” For nonblocking I/O in the library, an application must call `asai_open` with `ndelay_flag` set to `O_NDELAY`. It is also necessary to check error returns on every library function call. See `open(2)` in the *UNIX User Reference Manual* for more details.

The `oflag` that is passed to the library is `O_NDELAY` for nonblocking reads and writes. Since the library is not reentrant, an interrupt service routine (signal catcher) cannot call `asai_rcv` if the main application has already done so.

It is beyond the scope of this document to discuss the requirements and techniques of nonblocking I/O. Since the implementation of nonblocking I/O is an intricate matter, it is strongly recommended that such an undertaking should not be attempted before careful planning is conducted by an experienced developer.

Integration Test Tool

The integration test tool (ITT) is included only as a very basic example of an application.

Responding to Heartbeat

It is the responsibility of the CallVisor PC adjunct to respond to the ASAI heartbeat messages that are periodically sent from the ECS. This is the case for every CallVisor PC adjunct and for each ASAI link that is active.

The ECS sends a register message (with the operation value set to heartbeat) to the adjunct every two minutes. In order to respond to this request, the application must have previously called the function `asai_set_env` with `server_type` set to `C_MAINT_SER` for the application to be considered a maintenance server. As a maintenance server, the application must respond to every `C_HB_REQ` capability it receives from the ECS with a `C_HB_CONF`. If the adjunct fails to respond to three consecutive heartbeat requests from the ECS, the ECS takes down Layer 1 for five seconds and all active associations are aborted. If the application program itself has to perform the maintenance server role, great care must be taken to ensure that the application does not block waiting on other events which would cause it to miss a heartbeat request. A separate process should be used to perform the maintenance server role.

Beginning with Release 2.2 of the CallVisor ASAI product, an OA&M process, `asai_hb`, is available to accomplish this maintenance task.

Link Startup Failure

The QP module logs link startup failures to both the system console and the crash buffer. Use crash (1M) to examine the crash buffer. If the QP module has failed it is probably because of ECS and adjunct version incompatibility. If the link failure was because of incompatible parameters, correct the parameters as described in “Starting Up the System’’ on page 1-23 in Chapter 1, “Installation.’’

Detecting Failed ASAI Links

Some applications are required to detect and report a failed link within a certain time.

If the link failure detection time is more than three or four seconds, a good way to detect a failed link is to send a heartbeat request to the ECS at twice the rate of the requirement. If the heartbeat acknowledgment has not been received from the ECS when it is time to send the next heartbeat request, the link can be assumed to have failed. If the link failure detection time is fewer than two seconds, a more sophisticated algorithm might be needed. In constructing such an algorithm, the programmer should allow at least 100 milliseconds for the ECS to acknowledge the heartbeat. The programmer must also consider the delays due to any time-sharing activity that may be present.

The rate at which heartbeat requests are sent should take the link capacity and overall message load into consideration as well as the detection time requirement.

The link capacity is about 40 messages per second. The exact figure depends on the length and content of the messages. A heartbeat request and acknowledgment count as one message each. Sending a heartbeat once a second will consume somewhat less than 5 percent of the link capacity.

Version Selection

The ECS now supports version selection. The ISDN protocol stack, specifically the QP module, supports this feature. Three new tunable parameters are provided to allow selecting the desired version and allowable alternatives. QP_DESIRED_VER is set to 3. Version 1 corresponds to G3V3, Version 2 corresponds to G3V4 and Version 3 corresponds to release 6[1]. For release 6, the default also is 3. QP_HIGHERVER and QP_LOWERVER are provided to allow higher version and/or lower version operation. The tunable parameters are located in `/etc/conf/cf.d/stune`. The defaults are:

QP_DESIRED_VER is set to 3 (Release 6)
QP_HIGHERVER is set to 3 to allow higher (later) versions
QP_LOWERVER is set to 1 to allow lower (earlier) versions

To change the values, edit the `/etc/conf/cf.d/stune` file. Follow the standard tuning procedures and rebuild the kernel, then reboot.

1. The default is 3 which supports Release 5 and 6.

Inoperable System Errors

These procedures, if not done correctly with the correct values, can result in an inoperable system. Two cases are possible:

1. Defective kernel
2. Incompatible version parameters

If the kernel is defective, the system will not boot or you will be unable to log on. To recover from this, use the reboot with optional kernel procedure as described in your UNIX documentation.

If the version parameters are incompatible with the ECS version, the BRI link will not start up. The QP module will log this error to the system console and the crash buffer.

CV/LAN Troubleshooting

Client Side OA&M Utilities

asai_test

`asai_test` tests connections with the ECS. It has been modified to accept a new option “-t” with the IP address of the server. When it is run from the client, it will test the connection from the client through the specified server to the ECS.

asai_ver

This function returns the library version of the client library. This is needed to ensure that the client library and the server library versions are same.

Other OA&M and Test Programs

For the following OA&M programs and ITT to run, the host name or (IP address) of the server to which the program is connected has to be administered in `/usr/adm/asai/server`.

Trouble Connecting to Server

Use `ping` to check the connectivity to the CV/LAN server. If you cannot `ping` to the server, check your `/etc/hosts` file to ascertain that there is an entry for your server.

CVLN_ITT - ECS Connection Failure

Check the `asai_open` statement at the beginning of your script file for the node_id argument. This argument must be of the form:

```
WWW.XXX.YYY.ZZZ:signal01
```

Here `WWW.XXX.YYY.ZZZ` is the IP address of the server and `signal01` designates the ASAI link to the ECS.

Special Characters

A

Starting with Release 5, CallVisor ASAI supports a subset of European and Japanese Katakana characters that can be displayed by DEFINITY ECS on 85xx telephones. These European and Japanese characters are displayed when the Integrated Directory Database is queried for any name that contains all or some of these special characters.

ASAI parses the queried directory name and translates it according to the language that the application supports. An ASCII character preceded by a tilde (~) maps to a specific European or Katakana character that allows the presented name to be translated for the application. If a name is composed of special European characters (or the Japanese Katakana characters) as well as ASCII characters, the initial tilde combined with the subsequent ASCII character(s) toggles to the appropriate character map to translate the character(s); the second or next tilde turns off the toggle. If four tildes appear in a name sent by ECS, then the characters preceding the first and third tildes will toggle to the character map so that the characters can be translated; the second and fourth tildes discontinue character mapping and the presentation of ASCII characters within the name resumes. To illustrate the use of tildes for character mapping and translation, the Spanish name, "Pe~n~a" is received from ECS by ASAI and this name is subsequently translated by the ASAI adjunct to "Peña." The first tilde in this name turns on character mapping and the second tilde turns it off. It should be noted that the application can only present the translated name to the user if the operating system supports the European and Katakana character sets and fonts.

- For details on character mapping, see the "Enhanced Voice Terminal Display" section in Chapter 3 of the *DEFINITY Enterprise Communications Server Release 5 Feature Description*.
- For information on searching for a name in the Integrated Directory Database, see ''Value Query'' section in Chapter 5, ''ASAI Capabilities'' of this document.

Glossary

A

ACD
Automatic Call Distribution, a distribution method used to direct calls to a group of agents called ("splits"), using the ''most idle agent" or the ''hot seat" routing algorithm.

ACD agent/extension
A station extension that is a member of an Automatic Call Distribution (ACD) split/skill/hunt group.

ACD call
A call directed to an ACD split/skill/hunt group either directly or through vector processing.

Active
One of the five possible states for a party on a call. Active means that the party has been connected to the call.

Active call
For the Send DTMF Signals feature, a call that has received answer supervision, either network or timed (that is, resulting from elapse of a software timer), and has assigned listen and talk time slots. Therefore, for Send DTMF Signals purpose, an answered call on hold is an active call.

Active-Notification Association
A unique CRV/link number combination assigned by an application via the Event Notification Request capability.

Active-Notification Call
A call for which event reports are being sent over an active-notification association (communication channel) to the adjunct. Sometimes referred to as a "monitored call."

Active-Notification Domains
Group domains are Vector Directory Numbers or ACD split extensions for which Event Notification has been requested.

Adjunct
An independent processor connected to ECS.

Adjunct-Controlled Association
A unique CRV/link number combination assigned by an application via the Third Party Make Call or the Third Party Take Control capability.

Adjunct-Controlled Call
A call controlled by an adjunct-controlled association. The call must have originated via the Third Party Make Call capability or by a call with an active Third Party Take Control capability.

Adjunct-Controlled Split
An ACD split administered to be under adjunct control. Agents logged into such splits must do all telephony and ACD login/logout and change work mode functions through the adjunct.

Adjunct-Monitored Call
An adjunct-controlled call, an active-notification call, or a call that provides event reports over a domain-controlled association.

Adjunct Routing
A vector command/step that allows the switch to request a route/destination, from an ASAI adjunct, for the call executing the vector command/step. When an adjunct routing vector command is encountered, the switch disconnects any tone detector/call prompter connected to the call and discards any dial-ahead digits that had been collected. Collected digits are retained with the call and sent to the adjunct in the Route Request message.

Agent
A member of an ACD split or a hunt group. An agent can be a person at a terminal or a port on an adjunct processor. If the agent is a member of an ACD split, that agent is referred to as an ‘‘ACD agent.”

Alerting
One of the five possible states for a party on a call. Alerting means that the party is being alerted for the call.

AOC
Advice of Charge

AP
Adjunct Processor (Application Processor)

API
Application Program Interface

Application
An adjunct entity that requests and receives ASAI services or capabilities. The terms “application” and “adjunct” are used interchangeably.

ASAI
Adjunct Switch Application Interface. A messaging interface between the switch and an Adjunct Processor that allows the AP to perform call monitoring and control functions.

ASAI Application
An application running on an ASAI adjunct by making calls to a library written to meet the ASAI specification.

ASAI_E
See MFB. It is a label at the front of the DEFINITY LAN Gateway.

ASAI Host/Adjunct Processor
A computer processor that communicates with the switch via an ASAI link.

ASAI link
An ISDN BRI or Ethernet interface configured to support ASAI.

ASE
Application Service Element, a well-defined set of services available to other ASEs and to the application.

ASE instance
Active conversation between client and server using operations contained in the corresponding ASE.

ASEID
ASE identifier

Association
An association is a single invocation of an ASE between an application and ECS. An association is represented by a unique CRV/link number combination. This term is synonymous with the

following terms: ASE instance, cluster instance, cluster identifier, and single application object (SAO). An active association is one which applies to an existing call on ECS or to an extension on the call. SAO is preferred.

Automatic Call Distribution
See ACD.

B

BRI
Basic Rate Interface, referring to the basic ISDN physical interface providing two 64K bps full duplex B (bearer) channels and one 16K bps D channel for message-oriented signaling.

Busy/Unavailable
An event sent when a call cannot be completed because the party on the call is busy or the connection to the called endpoint cannot be established.

C

`call_id`
An identifier assigned by the node that processes the call (usually ECS).

Call Management System (CMS)
An application that collects, stores, analyzes, displays and reports ACD information provided by the switch. CMS enables customers to monitor and manage telemarketing centers by generating reports on the status of agents, splits, trunk groups, vectors, and vector directory numbers, and enables customers to partially administer the ACD features on the switch.

Call Reference Value
See CRV.

Call Vectoring
A method that manages inbound calls, using routing tables to uniquely define treatments for each call type. The call type is based on the dialed number or trunk group termination to a vector via vectoring directory numbers. The vectors are customer-programmable using commands that resemble a high-level programming language to specify what kind of treatment the call should be given.

Capability
A request or in indication of an operation.

Capability Groups
Sets of capabilities that denote association types. Each capability group may contain capabilities from several capability groups. Groups are provisioned through switch administration, and can be requested by an application. These are also called Application Service Elements (ASEs).

Cause Value
A cause value is returned in response to requests or in event reports when a denial occurs or an unexpected condition is encountered. ASAI cause values fall into two coding standards: coding standard 0 includes any cause values which are part of Lucent Technologies and ITU ISDN specifications; coding standard 1 includes any other ASAI cause values.

Client
As used in this guide, the initiator of a request for service; usually, but not always, the adjunct.

Cluster
Synonym for capability group or ASE. See also Association.

Controlled Extension
A station extension that is being monitored and controlled via domain-control association.

CPN
Calling Party Number

CRV
Call Reference Value a number used to identify the logical channels on the ISDN D channel.

CV/LAN
CallVisor LAN application running on Multi-Application Platform for DEFINITY (MAPD).

D

Daemon
Administrative process started when a UNIX system goes into a specified state that controls the activities of a system or peripheral device.

Denying a Request
Equivalent to sending a negative acknowledgement (NAK). This is done by sending an FIE with a return error component. A cause value is also provided. It should not be confused with the denial event report which applies to calls.

DLG
DEFINITY LAN Gateway is an application running on MAPD, or DEFINITY LAN Gateway Board (MFB).

Domain
Available domains are vector directory numbers, ACD splits and extensions. The VDN and ACD split domains are only used for active-notification associations. The extension domain is used only for the domain-control associations.

Domain-Control Association
A third party domain control request capability initiates a unique CRV/link number combination, which is referred to as a domain-control association.

Domain-Controlled Extension on a Call
An extension active on a call which provides event reports over one or two domain-control associations.

E

Expert Agent Selection
An optional feature that allows call center agents to have assigned skills and to receive calls based on their skill. EAS adds flexibility to ACD. Each agent assigned to a station at login time. This makes it possible to use the same physical station for a variety of skills and agents.

It is also an ACD feature in which calls can be directed to specialized pools of agents who possess the correct skills to handle the call.

Event Report
An activity within ECS which can cause notification to be sent to an adjunct.

F

Flexible Billing
A feature that allows ASAI to change the rate at which an incoming 900-type call is billed.

H

Held
One of the five possible states for a party on a call. Held means that the party has placed the call on hold and is not connected to the call.

Hunt Group
A group of stations which answer calls that are distributed using a routing algorithm. An ACD split is a special type of hunt group.

I

IE
Information Element

II-Digits
Information Identifier Digits. Information sent from the originating network switch that identifies the type of originator (cellular, prison, etc.) for the call.

IPCI
ISDN-PC (personal computer) Basic Rate Interface (BRI)

ISDN
Integrated Services Digital Network, the ITU project for standardization of operating parameters and interfaces for a network that allows a variety of mixed digital transmission services to be accommodated.

ISO characters
This is an international character encoding standard established by the International Standards Organization. It is a fixed-width encoding of 32 bits which provides unique codes for all the languages and scientific symbols.

ITU
International Telecommunications Union (ITU) is a division of the United Nations. It produces ‘‘recommendations” which, because of ITU’s global coverage, have more impact than ISO/IEC standards.

L

LED

Light Emitting Diode, a type of electronic device, a small lamp often used to indicate the state of the hardware.

LCRV

Local Call Reference Value, a number that uniquely identifies each Q.931/2 call controlled on a single stream. The Q.931/2 provider maps the LCRV to the CRV used in Q.931/2 messages.

Locale

Locale is an internationalization feature. Locale can be set to different regions (such as French, German, Japanese, etc.). It causes certain functions to act in the appropriate language or culture-dependent ways by providing information such as character representation, date and time representation, currency representation, etc.

M

MAPD

Multi-Application Platform for DEFINITY ECS, currently supporting DEFINITY LAN Gateway and CV/LAN applications.

MFB

Multi-Function Board for DEFINTY LAN Gateway application.

MWI

Message Waiting Indication

Manual Answer

An operation in which an Automatic Call Distribution (ACD) agent is on-hook and available to receive an ACD call, the call comes via ringing on the station set, and the agent goes off-hook on the ringing appearance to answer the call.

Monitored Call

A call that provides ASAI Event Reports over Event Notification associations.

Monitored domain

The VDN, split, or agent specified in an ASAI Event Notification Request. ASAI messaging to support this feature only applies to monitored domains.

N

Node

Processor reachable through a network.

Null

One of the five possible states for a party on a call. Null means that the party is not a participant on the call.

O

OPTREX

This set is a Lucent Technologies proprietary standard that consists of ASCII and 128 other special characters. The ASCII character tilde, “~” is an indicator to CallVisor PC that it has received an OPTREX sequence. Tilde is a nonprintable toggle character between ASCII and OPTREX. The high bit is set for the characters following an odd number of ~. These 8-bit OPTREX characters are mapped to their corresponding ISO characters. (See description of ISO characters above for more information.)

P

Party/Extension Active on a Call

A party is on the call if it is actually connected to the call (in active talk or in held state). An originator of a call is always a party on the call. Alerting parties, busy parties and tones are not parties on a call.

`party_id`

An abstraction of ECS/endpoint-dependent information. The party_id provides a way of referring to an endpoint without dealing with the specifics of the way ECS tracks calls, users, or ports and without dealing with the device used by the connected party. party_id is unique within call_id and is assigned by the node that processes the call.

Peer

Destination for ASAI requests or the source of ASAI indications.

PRI

Primary Rate Interface, referring to the ISDN physical interface providing 23 64K bps full duplex B channels and 1 64K bps D channel for message-oriented signaling.

Pump

The action of downloading software to an intelligent peripheral card.

Q

Q.931

ITU recommendation Q.931 which describes basic call control on ISDN BRI or PRI.

Q.932

ITU recommendation Q.932 which describes mechanisms for gaining supplementary services on an ISDN BRI or PRI.

R

RELease COMplete Message

The ISDN message type used to complete the ASAI Set Value and Value Query capabilities.

S

SACF
Single Association Control Function, that portion of an SAO that controls the interaction between two different ASEs.

SAO
Single Application Object, group of ASEs and an SACF that is used to communicate with a peer process. Preferred term for association. See also Association.

SAOID
SAO identifier, preferred synonym for cluster ID and association ID.

Seized
One of the five possible states for a party on a call. Seized indicates that the party is external to ECS and is represented by a trunk.

Selective Listening
A feature that allows an adjunct to disconnect a party (endpoint) from listening to other parties on an active call. A disconnected party can also be reconnected to the active call.

Send DTMF Signals
A feature that allows the DEFINITY ECS switch to generate DTMF tones on a talk path when requested to do so by the adjunct application.

Server
As used in this guide, the responder to a request for service; usually, but not always, ECS.

Service
A process that performs a specific system function and often provides an application programming interface (API) for other processes to call.

Single-Step Conference
A feature that allows a device to be added into an existing call without placing any parties on hold.

Split
A group of ACD agents organized to receive similar calls.

State
The five possible states for a call are null, alerting, active, held, and seized.

U

Unavailable
An event equivalent to ‘‘busy.”

UCID
Universal Call Identifier.

V

VDN

Vector Directory Number, an extension that provides access to the vectoring feature on ECS. Vectoring allows a customer to specify the treatment of incoming calls based on the dialed number.

Vector-Controlled Split

A hunt group or ACD split administered with the vector field enabled. Access to such split is only possible by dialing a VDN extension.

W

wide characters

These characters point to a translation of the raw OPTREX data into its equivalent ISO (four byte) characters.

Index

Numerics

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W