Trinity Release 3.9.X

Command Line Reference Guide

Sales Office: **+1 (301) 975-1000**
Technical Support: **+1 (301) 975-1007**
E-mail: **support@patton.com**
WWW: **www.patton.com**

Part Number: **07MTrinityCLI, Rev. F**
Revised: **August 10, 2016**

Patton Electronics Company, Inc.
7622 Rickenbacker Drive, Gaithersburg, MD 20879 USA
Tel: +1 (301) 975-1000 • **Fax:** +1 (301) 869-9293 • **Support:** +1 (301) 975-1007
Web: www.patton.com • **E-mail:** support@patton.com

Summary Table of Contents

Table of Contents

List of Figures

List of Tables

About this guide

The objective of this *Trinity Release 3.0 Command Line Reference Guide* is to provide information concerning the syntax and usage of the command set.

This section describes the following:

- Who should use this guide (see "Audience")
- How this document is organized (see "Structure")
- Typographical conventions and terms used in this guide (see "Typographical conventions used in this document" on page 38)

Audience

This guide is intended for the following users:

- System administrators who are responsible for installing and configuring networking equipment and who are familiar with the Trinity.
- System administrators with a basic networking background and experience, but who might not be familiar with the Trinity.
- Operators
- Installers
- Maintenance technicians

How to read this guide

Trinity is a complex and multifaceted operating system. Without the necessary theoretical background you will not be able to understand and use all the features available. Therefore, we recommend reading at least the chapters listed below to get a general idea about Trinity and the philosophy of contexts used for IP and circuit switching related configuration.

- Appendix A, "Trinity Architecture Terms and Definitions" on page 673 contains the terms and their definitions that are used throughout this *Trinity Release 3.x Command Line Reference Guide.*
- Chapter 1, "System Overview" on page 41 provides an overview of the main elements of a Trinity system.

Structure

This guide contains the following chapters and appendices:

- Chapter 1, "System Overview" on page 41 provides an overview of the main elements of a Trinity system.
- Chapter 2, "Configuration Concepts" on page 47 introduces basic Trinity configuration concepts.
- Chapter 3, "Command Line Interface (CLI)" on page 52 gives an overview of the CLI and the basic features that enable you to navigate the CLI and edit commands effectively.
- Chapter 4, "Accessing the CLI" on page 57 describes the procedures for entering Trinity commands via the command line interface (CLI) to obtain help, to change operator mode, and to terminate a session.
- Chapter 5, "Creating CLI Action Scripts" on page 69 describes how to create CLI action scripts that execute on specific events (for example, link down on the IP interface).

- Chapter 6, "System Image Handling" on page 74 describes how to load and maintain system images and driver software.
- Chapter 7, "Configuration File Handling" on page 78 describes how to upload and download configuration files from and to a Patton device.
- Chapter 8, "System Licensing and Preferences" on page 88 describes how to configure system licensing in Trinity.
- Chapter 9, "AAA Configuration" on page 91 provides an overview of configuring TACACS+.
- Chapter 10, "Basic System Management" on page 97 describes parameters that report basic system information to the operator or administrator, and their configuration.
- Chapter 11, "Programmable System-Event Configuration" on page 111 describes how to use programmable system events in Trinity.
- Chapter 12, "Alarm Management" on page 144 describes how to configure the alarm subsystem.
- Chapter 13, "Auto Provisioning of Firmware and Configuration" on page 147 provides an overview of Trinity's Auto Provisioning capabilities and tasks involved to configure it.
- Chapter 14, "Ethernet Port Configuration" on page 156 provides an overview of Ethernet ports and describes the tasks involved in their configuration through Trinity.
- Chapter 15, "Cellular Modem" on page 164 describes how to configure a cellular modem.
- Chapter 16, "Hardware Switching" on page 168 provides an overview of devices with an internal switch for connecting with external ports.
- Chapter 17, "DSL Port Configuration" on page 203 provides an overview of the DSL ports, their characteristics and the tasks involved in the configuration.
- Chapter 18, "Context Bridge" on page 209 describes how to configure bridging with Ethernet ports.
- Chapter 19, "Spanning Tree Configuration" on page 213 provides an overview of how to configure classic, rapid, and multiple spanning tree protocol.
- Chapter 20, "PPP Configuration" on page 219 describes how to configure the point-to-point protocol over different link layers.
- Chapter 21, "IP Context Overview" on page 229 outlines Trinity *Internet protocol* (IP) context, together with its related components.
- Chapter 22, "IP Interface Configuration" on page 238 provides a general overview of Patton device interfaces and describes the tasks involved in their configuration.
- Chapter 23, "IP Routing" on page 248 provides an overview of IP routing and describes the tasks involved in configuring static IP routing in Trinity.
- Chapter 24, "Fast-Path" on page 262 explains how to use the new Fast Path feature to speed-up the routing performance of Trinity devices.
- Chapter 25, "NAT/NAPT Configuration" on page 265 provides a general overview of the network address port translation and describes the tasks involved in its configuration.
- Chapter 26, "DHCP Configuration" on page 273 provides an overview of the *Dynamic Host Configuration Control Protocol* (DHCP) and describes the tasks involved in its configuration.
- Chapter 27, "DNS Configuration" on page 279 describes how to configure the *domain name system* (DNS) component.
- Chapter 28, "SNTP Client Configuration" on page 283 describes how to configure a *simple network time protocol* (SNTP) client.

- Chapter 29, "SNMP Configuration" on page 286 provides overview information about the *simple network management protocol* (SNMP) and describes the tasks used to configure those of its features supported by Trinity.
- Chapter 30, "Public-Key Infrastructure (PKI)" on page 301 provides an overview on how to set up the public-key infrastructure (PKI) on a Patton device.
- Chapter 31, "Quality of Service (QoS) Overview" on page 315 provides an overview of the core principles of Trinity's Quality-of-Service architecture.
- Chapter 32, "Profile Service-Policy Configuration" on page 319 describes how to use and configure service-policy profiles.
- Chapter 33, "Access Control List Configuration" on page 331 provides an overview of IP *Access Control Lists* (ACLs) and describes the tasks involved in configuring them.
- Chapter 34, "Classifier Configuration" on page 341 provides an overview of Trinity's packet classifier and describes the tasks involved in its configuration.
- Chapter 35, "Service Policy Configuration" on page 349 describes how to use and configure service-policy profiles.
- Chapter 36, "Packet Matching" on page 353 lists the criteria that may be used to match packets and specifies the command line syntax.
- Chapter 37, "SIP Profile Configuration" on page 359 describes the SIP profile, which specifies disconnect cause mappings from SIP codes to Q.931 causes, and vice versa.
- Chapter 38, "VoIP Profile Configuration" on page 365 provides an overview of VoIP profiles, and describes how they are used and the tasks involved in VoIP profile configuration.
- Chapter 39, "PSTN Profile Configuration" on page 390 provides an overview of PSTN profiles, and describes how they are used and the tasks involved in PSTN profile configuration.
- Chapter 40, "CS Context Overview" on page 394 provides an overview of the circuit-switching (CS) context and associated components, and describes the tasks involved in its configuration.
- Chapter 41, "CS Interface Configuration" on page 416 provides an overview of interfaces in the CS context and describes the tasks involved in their configuration.
- Chapter 42, "Tone Configuration" on page 424 provides an overview of call-progress-tone profiles and tone-set profiles, and describes the tasks involved in their configuration.
- Chapter 43, "Authentication Service" on page 431 describes how to configure authentication services in Trinity.
- Chapter 44, "Location Service" on page 434 describes how to configure location services in Trinity.
- Chapter 45, "Call Router Configuration" on page 455 provides an overview of call router tables, mapping tables and call services and describes the tasks involved in configuring the call router in Trinity.
- Chapter 46, "Global SIP Configuration" on page 525 describes the Trinity commands available under the global SIP configuration mode.
- Chapter 47, "SIP Overload Configuration" on page 529 describes how to configure the Trinity device's behavior in case of an overload situation.
- Chapter 48, "SIP Interface Configuration" on page 538 provides an overview of SIP interfaces used by context SIP gateways and describes the specific tasks involved in their configuration.

- Chapter 49, "Secure SIP Applications" on page 555 discusses different Transport Layer Security (TLS) and Secure Real-Time Protocol (SRTP) scenarios and provides the configuration snippets to set up the Patton device for them.
- Chapter 50, "SIP Security" on page 564 provides an overview of Trinity's *Transport Layer Security* (TLS) capabilities for SIP and describes the tasks involved in configuring it.
- Chapter 51, "SIP Call-router Services" on page 595 contains the description of all SIP specific call router services, which are only available if the software includes the SIP component.
- Chapter 52, "Context SIP Gateway Overview" on page 601 provides an overview of the context SIP gateway.
- Chapter 53, "ISDN Overview" on page 617 provides an overview of ISDN ports and describes the tasks involved in configuring ISDN ports.
- Chapter 54, "ISDN Configuration" on page 622 describes the configuration of the Q.921 and Q.931 protocol and how to bind the ISDN protocol to an application like the Call Control.
- Chapter 55, "ISDN Interface Configuration" on page 631 provides an overview of ISDN interfaces, and the tasks involved in their configuration.
- Chapter 56, "PRI Port Configuration" on page 648 provides an overview of the PRI (Primary Rate Interface) ports, their characteristics and the tasks involved in the configuration.
- Chapter 57, "E1/T1 Port Configuration" on page 654 provides an overview of the E1/T1 ports, their characteristics and the tasks involved in the configuration.
- Chapter 58, "BRI Port Configuration" on page 656 provides an overview of the BRI (Basic Rate Interface) ports, their characteristics and the tasks involved in the configuration.
- Chapter 59, "Debug and Monitoring" on page 661 describes how to debug VoIP sessions, including the signaling part and the voice data path part (speech, fax, and modem connectivity).
- Appendix A, "Trinity Architecture Terms and Definitions" on page 673 contains terms and definitions that are used in this *Trinity Software Configuration Guide*.
- Appendix B, "Command summary" on page 679 is a command reference.
- Appendix C, "Glossary of Terms" on page 682 is the glossary of terms.

Precautions

The following are used in this guide to help you become aware of potential problems:

Note A note presents additional information or interesting sidelights.

IMPORTANT

The alert symbol and IMPORTANT heading calls attention to important information.

Typographical conventions used in this document

This section describes the typographical conventions and terms used in this guide.

General conventions

In this guide we use certain typographical conventions to distinguish elements of commands and examples. In general, the conventions we use conform to those found in IEEE POSIX publications. The procedures described in this manual use the following text conventions:

Table 1. General conventions

Convention	Meaning
Garamond blue type	Indicates a cross-reference hyperlink that points to a figure, graphic, table, or section heading. Clicking on the hyperlink jumps you to the reference. When you have finished reviewing the reference, click on the **Go to Previous View** button in the Adobe® Acrobat® Reader toolbar to return to your starting point.
Helvetica bold type	Commands and keywords are in **boldface** font.
Helvetica bold-italic type	Parts of commands, which are related to elements already named by the user, are in ***boldface italic*** font.
Italicized Helvetica type	Variables for which you supply values are in *italic* font
Garamond italic type	Indicates the names of fields or windows.
Garamond bold type	Indicates the names of command buttons that execute an action.
< >	Angle brackets indicate function and keyboard keys, such as **<shift>**, **<ctrl>**, **<c>**, and so on.
[ ]	Elements in square brackets are optional.
{a | b | c}	Alternative but required keywords are grouped in braces ({ }) and are separated by vertical bars (|)
device	The leading IP address or name of a Patton device is substituted with ***device*** in ***boldface italic*** font.
device	The leading device on a command line represents the name of the Patton device
#	An hash sign at the beginning of a line indicates a comment line.

Service and support

Patton Electronics offers a wide array of free technical services. If you have questions about any of our other products we recommend you begin your search for answers by using our technical knowledge base. Here, we have gathered together many of the more commonly asked questions and compiled them into a searchable database to help you quickly solve your problems.

Patton support headquarters in the USA

- Online support: Available at www.patton.com
- E-mail support: E-mail sent to support@patton.com will be answered within 1 business day
- Telephone support: Standard telephone support is available five days a week—from **8:00 am** to **5:00 pm EST** (**1300** to **2200 UTC/GMT**)—by calling **+1 (301) 975-1007**
- Support via VoIP: Contact Patton free of charge by using a VoIP ISP phone to call sip:support@patton.com
- Fax: **+1 (301) 869-9293**

Alternate Patton support for Europe, Middle East, and Africa (EMEA)

- Online support: Available at www.patton-inalp.com
- E-mail support: E-mail sent to support@patton-inalp.com will be answered within 1 business day
- Telephone support: Standard telephone support is available five days a week—from **8:00 am** to **5:00 pm CET** (**0900** to **1800 UTC/GMT**)—by calling **+41 (0)31 985 25 55**
- Fax: **+41 (0)31 985 25 26**

Warranty Service and Returned Merchandise Authorizations (RMAs)

Patton Electronics is an ISO-9001 certified manufacturer and our products are carefully tested before shipment. All of our products are backed by a comprehensive warranty program.

Note If you purchased your equipment from a Patton Electronics reseller, ask your reseller how you should proceed with warranty service. It is often more convenient for you to work with your local reseller to obtain a replacement. Patton services our products no matter how you acquired them.

Warranty coverage

Our products are under warranty to be free from defects, and we will, at our option, repair or replace the product should it fail within one year from the first date of shipment. Our warranty is limited to defects in workmanship or materials, and does not cover customer damage, lightning or power surge damage, abuse, or unauthorized modification.

Returns for credit

Customer satisfaction is important to us, therefore any product may be returned with authorization within 30 days from the shipment date for a full credit of the purchase price. If you have ordered the wrong equipment or you are dissatisfied in any way, please contact us to request an RMA number to accept your return. Patton is not responsible for equipment returned without a Return Authorization.

Return for credit policy

- Less than 30 days: No Charge. Your credit will be issued upon receipt and inspection of the equipment.
- 30 to 60 days: We will add a 20% restocking charge (crediting your account with 80% of the purchase price).
- Over 60 days: Products will be accepted for repairs only.

RMA numbers

RMA numbers are required for all product returns. You can obtain an RMA by doing one of the following:

- Completing a request on the RMA Request page in the *Support* section at www.patton.com
- By calling +1 (301) 975-1007 and speaking to a Technical Support Engineer
- By sending an e-mail to returns@patton.com

All returned units must have the RMA number clearly visible on the outside of the shipping container. Please use the original packing material that the device came in or pack the unit securely to avoid damage during shipping.

Shipping instructions

The RMA number should be clearly visible on the address label. Our shipping address is as follows:

Patton Electronics Company
RMA#: xxxx
7622 Rickenbacker Dr.
Gaithersburg, MD 20879-4773 USA

Patton will ship the equipment back to you in the same manner you ship it to us. Patton will pay the return shipping costs.

Chapter 1 System Overview

Chapter contents

Introduction

This chapter provides an overview of the main elements of a Patton device system.

A complete Patton device system or network, as installed in any of the application scenarios introduced in section "Applications" on page 43, is typically composed of the following main elements plus a third-party network infrastructure:

- The first and most obvious element is the *Patton* devices (also referred to as *hardware platforms* or *network devices*) that provide the physical connectivity, the CPU and DSP resources. Some Patton device models support packet-routed and circuit-switched traffic, others do not.
- The second element comprises the embedded software—called *Trinity*—running on the Patton device hardware platforms.
- Finally, a third-party IP network and transmission infrastructure provides IP connectivity between the above elements. This infrastructure can range from a simple Ethernet hub or switch to highly complex networks including multiple access technologies, backbone transmission, and service devices.

Figure 1 depicts the basic system model of a Patton device. Patton VoIP + Router devices have the following main components:

- 64k circuit switching between on-board ISDN ports and between ISDN and PSTN interface cards. The circuit switching engine uses dedicated hardware resources and therefore can bypass the VoIP gateway and packet routing engine.
- A gateway (GW) that converts telephone circuits into Internet protocol (IP) packet streams and vice versa. SIP-compliant and SIP Voice over IP (VoIP) is supported.
- An IP router with on-board ports and optional data interface cards is QoS enabled, thereby allowing classification, shaping, and scheduling of multiple service classes.

Patton Router-only devices have the following main components:

- Physical data ports: Ethernet, DSL, Wifi, Cellular Modem, etc.
- Routing Core
- Firewall
- NAT

For more detailed hardware information, refer to the getting started guide that came with your Patton system.

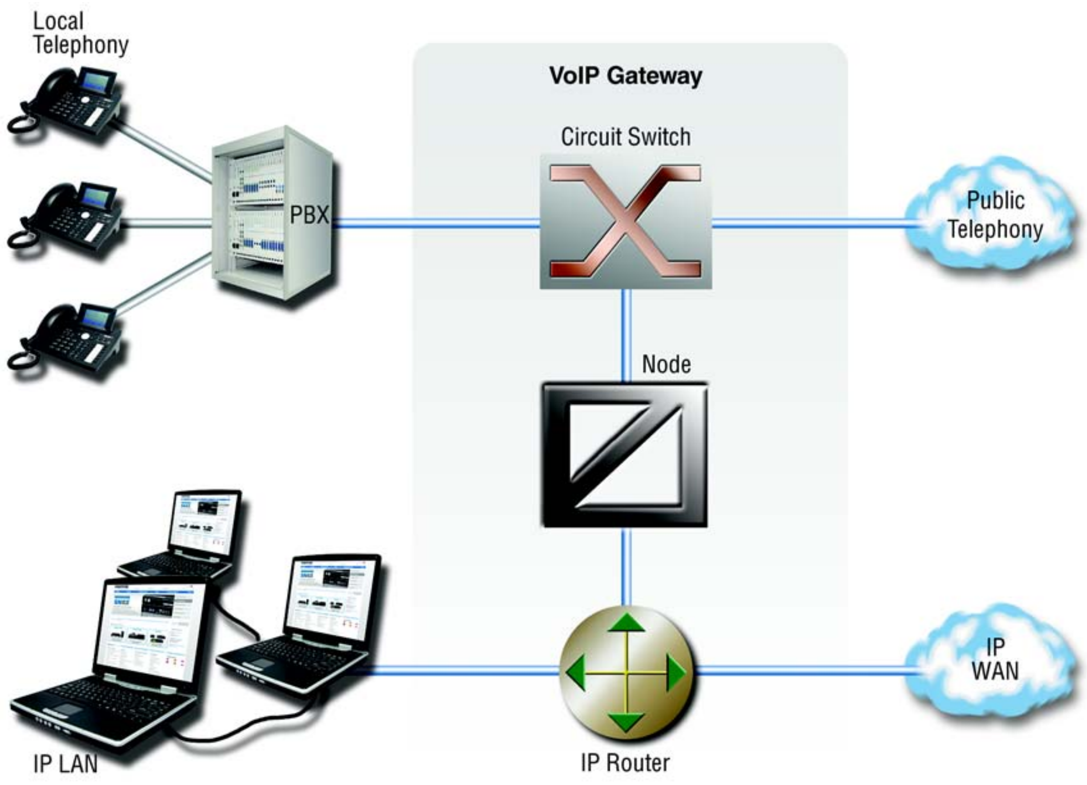


Figure 1. Basic system (abstract) model

Trinity embedded software

Trinity is the application software that runs on certain Patton device hardware platforms. Trinity is available in several releases.

A Trinity build is a binary image file. The download to the Patton device is packaged in a single TAR image that is uploaded onto the device. Refer to chapter 6, "System Image Handling" on page 74 for details on Trinity image downloads.

Applications

The Patton product family consists of highly flexible multi-service IP network devices, which fit a range of networking applications. This section provides an overview of the following Patton device applications and the main elements in a Patton network.

- Carrier networks—Patton devices are used as customer gateways or integrated access devices at the customer premises. These applications are also called Integrated Service Access (ISA).
- Enterprise networks—Patton devices are used as WAN routers and voice gateways for inter-site networking. These applications are also called *multiservice intranets* (MSI).
- LAN telephony—Patton devices serve as gateways between the LAN and the local PBX or PSTN access. These applications are also called LAN voice gateway (LVG).

Carrier networks

The network termination (NT) device in a multi-service IP based provider network plays a vital role. It provides the service access point for the subscriber with respect to physical connectivity and protocol interoperability.

Since the access bandwidth in most cases represents a network bottleneck, the NT must also ensure traffic classification and the enforcement of service level agreements (SLA) on the access link. In broadband access networks, this NT is also called an Integrated Access Device (IAD) or customer gateway.

Patton products offer unique features as customer gateways for business services. It provides amongst others full ISDN feature support, local switching and breakout options and mass provisioning support.

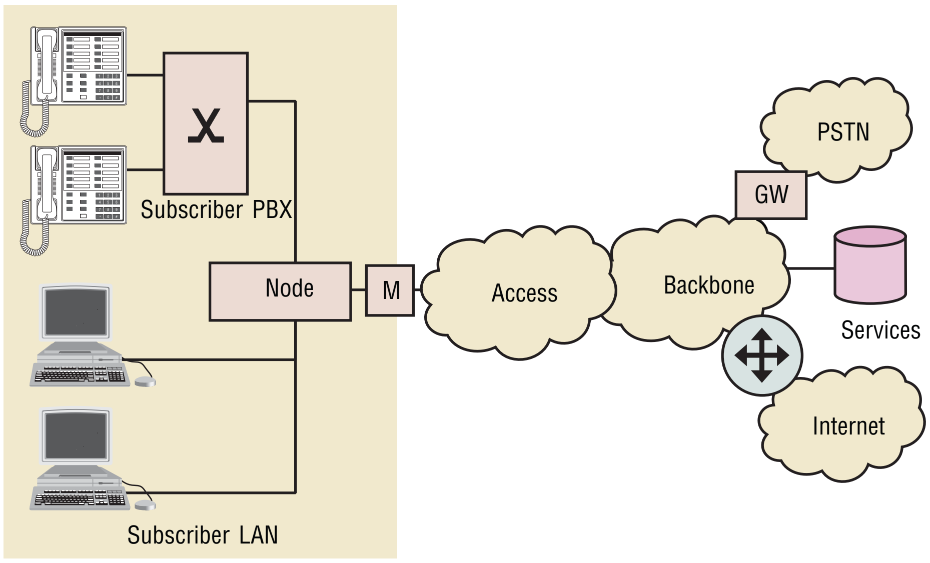


Figure 2. Typical carrier network application with a Patton device

Figure 2 shows the deployment of Patton devices in carrier networks. Each subscriber site is equipped with a Patton device that connects the subscriber LAN on one side with the provider network and services on the other.

Typical services in these networks are softswitch-based telephony, PSTN access through V5.2 gateways, PBX networking services, and LAN interconnection.

Typical access technologies for these networks include xDSL, WLL, PowerLine, cable and conventional leased lines. With the use of an external modem, the device can connect to leased lines or any bridged-Ethernet broadband access.

Enterprise networks

In company-owned and operated wide area networks, Patton devices can be used to converge voice and data communications on the same IP link. In combination with centralized services such as groupware and unified messaging, Patton devices provide migration and investment protection for legacy telephony systems.

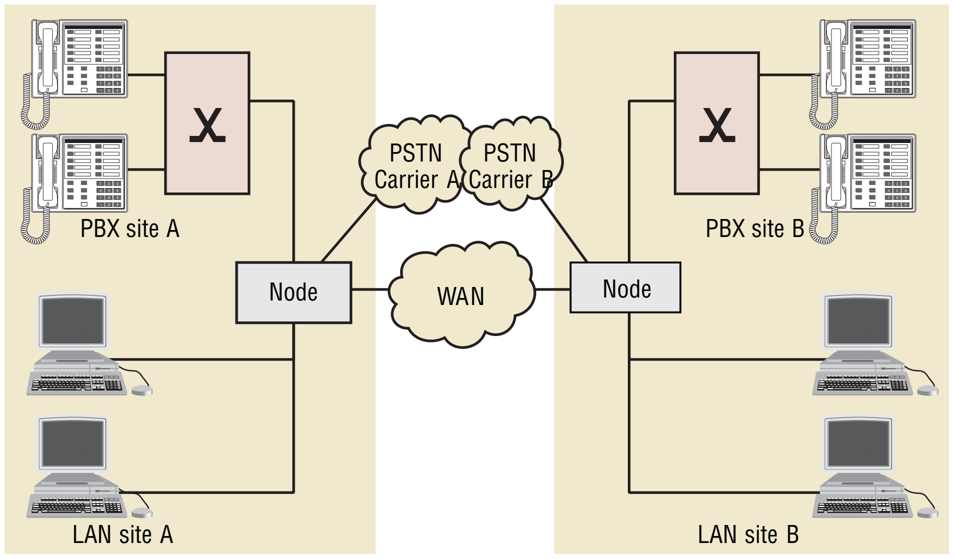


Figure 3. Typical enterprise network with a Patton device

Figure 3 shows the deployment of Patton devices in enterprise networks. Each site (headquarter, branch or home office) is equipped with a Patton device that connects the local LAN and telephony infrastructure with the IP WAN and the local PSTN carrier.

LAN telephony

With its voice-over-IP gateway features, the Patton device can be used as a standalone gateway for VoIP telephony (see figure 4).

A standalone gateway has performance reliability and scalability advantages compared with PC-based gateway cards. In this application, the Patton device also offers a migration path to enterprise or carrier networking.

Figure 4 shows the deployment of a Patton device as a LAN voice gateway.

The PSTN connections can be scaled from a single ISDN basic rate access to multiple primary rate lines. With Q.SIG, integration in private PBX networks is also supported.

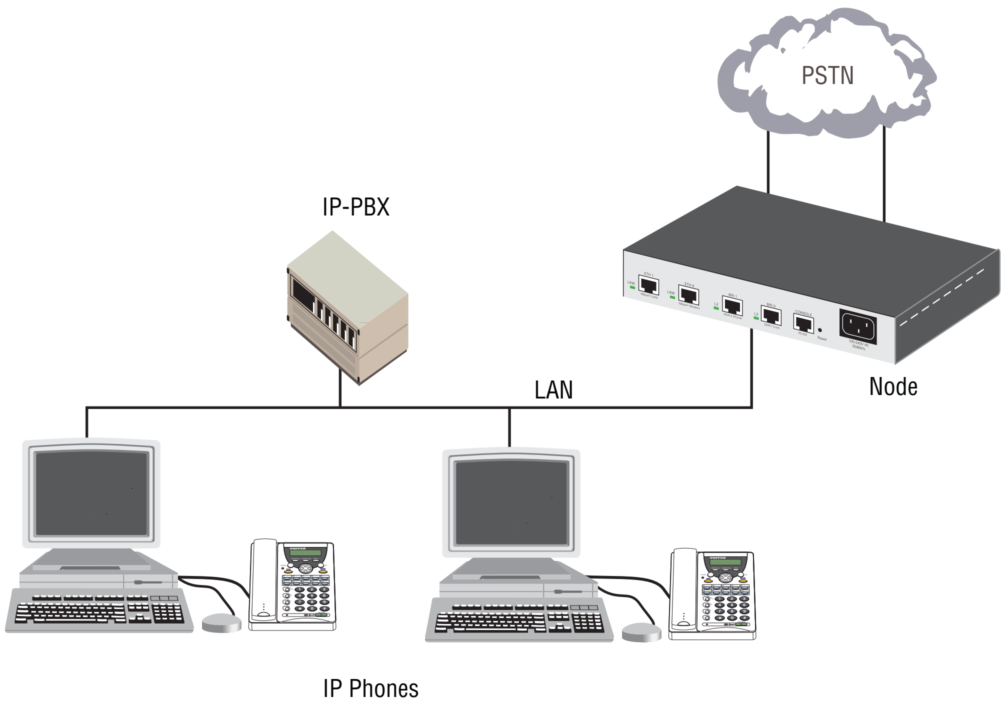


Figure 4. Typical LAN telephony system with a Patton device gateway

Chapter 2 Configuration Concepts

Chapter contents

Introduction

This chapter introduces basic Trinity configuration concepts. A good understanding of these concepts is vital for the configuration tasks explained in the remaining chapters of this guide.

Patton strongly recommends that you read through this chapter because it introduces the fundamental ideas behind the structure of the command line interface. Once you understand and know this structure, you will find it much more intuitive to navigate through the CLI and configure specific features.

This chapter includes the following sections:

- Contexts and gateways (see page 49)
- Interfaces, ports, and bindings (see page 50)
- Profiles and Use commands (see page 51)

Patton devices are multi-service network devices that offer high flexibility for the inter-working of circuit-switched and packet-routed networks and services. In order to consistently support a growing set of functions, protocols, and applications, Trinity configuration is based on a number of abstract concepts that represent the various Trinity components.

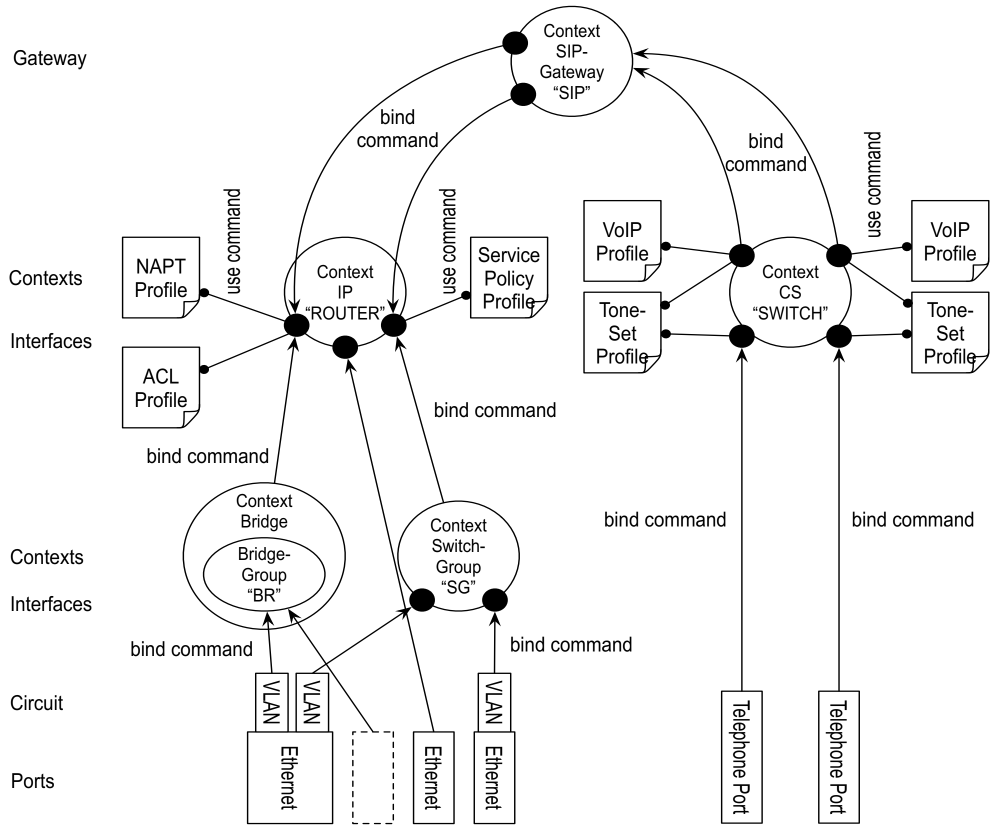


Figure 5. Configuration concept overview

Figure 5 shows the various elements of a complete Patton device configuration. Each of these elements implements one of the configuration concepts described in this chapter. The figure also shows the relationships and associations between the different elements. The relations are specified through *bind* (arrow) and *use* (bullet-lines) commands. For example, you need *bind* commands to bind a physical port to a logical interface, and *use* commands to assign profiles to contexts.

The sections that follow refer to figure 5 on page 48 and describe the concepts and elements in more detail.

Contexts and Gateways

Context

A *context* represents one specific networking technology or protocol, namely IP (Internet Protocol) or CS (circuit-switching). A context can be seen as *virtual dedicated equipment* within the Patton device. For example:

- A CS context contains the circuit-switching functions of the Patton device. It can be thought of as an embedded multiplexer or cross-connect within the device
- An IP context contains the routing functions of the Patton device. It can be thought of as an embedded router within the device
- A Bridge context contains the bridging functions of the Patton device at the CPU layer. Software bridging and bridge management is configured within.
- A Switch-group context contains the bridging functions of the Patton device at the hardware layer.

The contexts are identified by a name and contain the configuration commands that are related to the technology they represent. A separate configuration can be built by means of the context concept for newly supported network layer technologies without complicating the configuration methods of existing features. For example, as ATM or FR switching becomes available so an ATM or FR context can be introduced.

Each context contains a number of *interfaces*, which build the connections to other Trinity elements and the outside world. Figure 5 on page 48 shows four contexts:

- one type IP named *router*
- one type CS named *switch*
- one type Bridging
- one type of Switch-group

Note Trinity currently supports only one instance of the CS and IP context types.

Example

The IP context named *router* can contain static routes, RIP, and NAT configuration parameters. The default circuit-switching context named *switch* can contain number translations, local breakout conditions, and least-cost routing parameters.

Gateway

The concept of a *gateway* is introduced for the communication between contexts of different types. A gateway handles connections between different technologies or protocols. For example, a VoIP gateway connects an IP context to a circuit-switching context.

The gateways are each of a specific type and are identified by a name. Each named gateway contains its configuration parameters. With this concept, multiple virtual gateways can be instantiated and used at the same time.

Interfaces, Ports, and Bindings

Interfaces

The concept of an interface in Trinity differs from that in traditional networking devices. Traditionally, the term *interface* is often synonymous with *port* or *circuit*, which are physical entities. In Trinity however, an interface is a logical construct that provides higher-layer protocol and service information, such as layer 3 addressing. Interfaces are configured as part of a context, and are independent of physical ports and circuits. The decoupling of the interface from the physical layer entities enables many of the advanced features offered by Trinity.

In order for the higher-layer protocols to become active, you must associate an interface with a physical port or circuit. This association is referred to as a *binding* in Trinity. Refer to the "Bindings" section for more information. In figure 5 on page 48, the IP context shows three interfaces and the CS context shows four interfaces. These interfaces are configured within their contexts. The bindings shown in the figure are not present when the interfaces are configured; they are configured later.

Ports and circuits

Ports and *circuits* in Trinity represent the physical connectors and channels on the Patton device hardware. The configuration of a port or circuit includes parameters for the physical and data link layer such as line clocking, line code, framing and encapsulation formats or media access control. Before any higher-layer user data can flow through a physical port or circuit, you must associate that port or circuit with an interface on a context. This association is referred to as a *binding*. Refer to the "Bindings" section for more information.

Examples of ports are: Ethernet or DSL. Ports are numbered according to the label (or abbreviation) printed on the hardware.

Example: Ethernet 0/1, BRI 3/2

Figure 5 on page 48 shows five ports. Three ports are bound directly to an IP interface. One port has a single circuit configured, which is bound to the IP context. Two ISDN ports are bound to CS interfaces.

Bindings

Bindings form the association between circuits or ports and the interfaces configured on a context. No user data can flow on a circuit or Ethernet port until some higher-layer service is configured and associated with it.

Bindings are configured statically in the port or circuit configuration. The binding is created bottom-up, that is from the port to the interface.

In the case of VoIP CS interfaces, bindings are configured statically in the CS interface configuration. The binding is created from the interface to the gateway.

Bindings from ports to interfaces shown in figure 5 on page 48.

Profiles and Use commands

Profiles

Profiles provide configuration shortcuts. They contain specific settings that can be used in multiple contexts, interfaces, or gateways. This concept allows to avoid repetitions of groups of configuration commands that are the same for multiple elements in a configuration.

Profiles used in the IP and CS contexts are shown in figure 5 on page 48.

Use Commands

Use commands form the association between profiles and contexts, gateways, or interfaces. For example, when a profile is *used* in a context, all the configuration settings in that profile become active within the context.

Chapter 3 Command Line Interface (CLI)

Chapter contents

Introduction

The primary user interface to Trinity is the command line interface (CLI). You can access the CLI via the Patton device console port or through a Telnet or SSH session. The CLI lets you configure the complete Trinity functionality. You can enter CLI commands online or as a configuration script in the form of a text file. The CLI also includes monitoring and debugging commands. CLI commands are simple strings of keywords and user-specified arguments.

This chapter gives an overview of the CLI and the basic features that allow you to navigate the CLI and edit commands effectively. The following topics are covered:

- Command Modes
- Command Editing (see page 54)

Command modes

The CLI is composed of modes. There are three *mode groups*: the operator, the *administrator mode* and the *configure mode*. The configuration mode group contains all of the remaining modes. A command mode is an environment within which a group of related commands is valid. All commands are mode-specific, and certain commands are valid in more than one mode. A command mode provides command line completion and context help within the mode. The command modes are organized hierarchically. The current working mode is indicated by the CLI prompt. Appendix B, "Mode summary" on page 546 contains a detailed overview of all command modes, and appendix B, "Command summary" on page 679 describes the commands that are valid in each mode.

CLI prompt

For interactive (online) sessions, the system prompt is displayed as:

```
devicename>
```

In the operator exec mode, the system prompt is displayed as:

```
devicename#
```

In the administrator exec mode and in the different configuration modes, the system prompt is displayed as:

```
devicename(mode)device#
```

Where:

- *devicename* is the currently configured name of the Patton device, the IP address or the hardware type of the device that is being configured
- *mode* is a string indicating the current configuration mode, if applicable.
- *name* is the name of the instance of the current configuration mode

Example: the prompt in **radius-client mode**, assuming the devicename *device* and the instance *deepblue* is:

```
device(cfg)[deepblue]#
```

The CLI commands used to enter each mode and the system prompt that is displayed when you are working in each mode is summarized in appendix B, "Mode summary" on page 546.

Navigating the CLI

Initial mode

When you initiate a session, you can log in with operator or administrator privileges. Whichever login you use, the CLI is always set to operator exec (non-privileged exec) mode by default upon startup. This mode allows you to examine the state of the system using a subset of the available CLI commands.

System changes

In order to make changes to the system, the administrator exec (privileged exec) mode must be entered. The **enable** user interface command is used for this purpose (the **enable** command is only accessible if you are logged in as an administrator). Once in administrator exec mode, all of the system commands are available to you.

Configuration

To make configuration changes, the configuration mode must be entered by using the **configure** command in the administrator exec mode.

Changing Modes

The **exit** command moves the user up one level in the mode hierarchy (the same command works in any of configuration modes).

The **exit** command terminates a CLI session when typed from the operator exec mode.

A session can also be terminated by using the **logout** command within any mode.

Command editing

Command help

To see a list of all CLI commands available within a mode, type a question mark **<?>** or the **<tab>** key at the system prompt in the mode of interest. A list of all available commands is displayed. Commands that have become available in the current mode are displayed at the bottom of the list, separated by a line. Commands from higher hierarchy levels are listed at the top.

You can also type the question mark or the **<tab>** key while in the middle of entering a command. Doing so displays the list of allowed choices for the current keyword in the command. Liberal use of the question mark functionality is an easy and effective way to explore the command syntax.

The No Form

Almost every command supports the keyword **no**. Typing the **no** keyword in front of a command disables the function or “deletes” a command from the configuration. For example, to enable the DHCP server trace tool, enter the command **debug dhcp-server**. To subsequently disable the DHCP server trace, enter the command **no debug dhcp-server**.

Command completion

You can use the **<tab>** key in any mode to carry out command completion. Partially typing a command name and pressing the **<tab>** key causes the command to be displayed in full up to the point where a further choice has to be made. For example, rather than typing **configure**, typing **conf** and pressing the **<tab>** key causes the

CLI to complete the command at the prompt. If the number of characters is not sufficient to uniquely identify the command, the CLI will provide a list with all commands starting with the typed characters. For example, if you enter the string *co* in the configure mode and press **<tab>**, the selections **configure**, **copy**, and **context** are displayed. The CLI may be configured to automatically complete commands without pressing the <tab> key. This will only happen if a unique completion option exists.

Command	Purpose
[no] cli auto-completion	Enable or disable CLI automatic command completion.

Command history

Trinity maintains a list of previously entered commands that you can go through by pressing the **<up-arrow>** and **<down-arrow>** keys, and then pressing **<enter>** to enter the command. The show history command displays a list of the commands you can go through by using the arrow keys.

Command Editing Shortcuts

Trinity CLI provides a number of command shortcuts that facilitate editing of the command line. Command editing shortcuts are summarized below. The syntax <**Ctrl**>-<**p**> means press the <**p**> key while holding down the keyboard's control key (sometimes labeled *Control, Ctl*, or *Ctrl*, depending on the keyboard and operating system of your computer). <**Esc**>-<**f**> is handled differently; press and release the escape key (often labeled *Esc* on many keyboards) and then press the <**f**> key.

Keyboard	Description
<Ctrl>-<p> or **<up-arrow>**	Recall previous command in the command history.
<Ctrl>-<n> or **<down-arrow>**	Recall next command in the command history.
<right-arrow>	Move cursor forward one character.
<left-arrow>	Move cursor backward one character.
<Esc>-<f>	Move cursor forward one word.
<Esc>-<b>	Move cursor backward one word.
<Ctrl>-<a>	Move cursor to beginning of line.
<Ctrl>-<e>	Move cursor to end of line.
<Ctrl>-<k>	Delete to end of line.
<Ctrl>-<u>	Delete to beginning of line.
<Ctrl>-<d>	Delete character.
<Ctrl>-<c>	Quit editing the current line.
<Ctrl>-<v>	Insert a code to indicate to the system that the keystroke immediately following should be treated as normal text, not a CLI command. For example, pressing the question mark **<?>** character in the CLI prints a list of possible tokens. If you want to use the "?" in a configuration command, e.g. to enter a regular expression, press **Ctrl-v** immediately followed by the question mark <**?**>.

Timed Execution of CLI Command

The command timer allows the timed execution of CLI commands. The timer command is incremental; this means for each time it is entered, a new timer is created. All timers appear in the running-configuration, except if they have been created with the volatile option. It is possible to specify for each timer the start time and the reoccurrence. Use the CLI help (tab completion) for detailed description of all configuration options.

Example:

```
timer FIRMWARE_UPDATE now + 2 minutes every 10 minutes “provisioning execute FIRM-
WARE”
```

Starts a timer named FIRMWARE_UPDATE, whose first execution time is 2 minutes after the command is entered (2 minutes after device startup if the command is in the startup-configuration), and is executed every 10 minutes afterwards. This timer does not expire. The executed CLI command is provisioning execute FIRM-WARE.

As there are many possibilities to configure the timer time specification, here are some practical examples:

```
timer MYTIMER every day "command"
```

Will execute the command "command" every day at the same time.

```
timer MYTIMER oct 9th 2019 "command"
```

Will execute the command "command" on October 9th 2019 at midnight.

```
timer MYTIMER now + 2 minutes every month "command"
```

Will execute the command "command" in 2 minutes and then every month at the same hour.

Mode: configure

Step	Command	Purpose
1	**node (cfg)#timer (volatile)**<name> <time specification>	Enter the timer

Chapter 4 Accessing the CLI

Chapter contents

Introduction

Patton products are designed for remote management and volume deployment. The management and configuration of Patton devices is therefore based on IP network connectivity. Once a Patton device is connected to, and addressable in, an IP network, you can remotely perform all configuration, management, and maintenance tasks.

This chapter describes the procedures for entering Trinity commands via the command line interface (CLI), to obtain help, to change operator mode, and to terminate a session. You can access a Patton device as follows:

- Directly, via the console port (if available)
- Remotely, via the IP network (by using a Telnet or SSH application)

The ports available for connection and their labels are shown in the getting started guide that came with your unit. Remember that the CLI supports a command history and command completion. By scrolling with the **up** and **down** arrow keys, you can find many of your previously entered commands. Another time-saving tool is command completion. If you type part of a command and then press the **<tab>** key, the Trinity shell will present you with either the remaining portion of the command or a list of possible commands. These features are described in Chapter 3, "Command Line Interface (CLI)" on page 52. The telnet and SSH server can be disabled if desired.

IMPORTANT

Although Trinity supports concurrent sessions via SSH or the console port, we do not recommend working with more than one session to configure a specific Patton device. However, using one session for configuration and another for debugging is a good idea.

Accessing the Trinity CLI task list

The following sections describe the basic tasks involved in accessing the Trinity command line interface. Depending on your application scenario, some tasks are mandatory while others could be optional.

- Accessing via the console port (see page 59)
- Accessing via a SSH session (see page 60)
- Using an alternate TCP listening port for the SSH server (see page 60)
- Disabling the SSH server (see page 60)
- Logging on (see page 60)
- Selecting a secure password (see page 61)
- Configuring operators and administrators (see page 62)
- Displaying the CLI version (see page 65)
- Switching to another log-in account (see page 65)
- Checking identity and connected users (see page 65)
- Ending a SSH or console port session (see page 68)

Accessing via the console port

If a console port is available, the host computer can be connected directly to it with a serial cable (see Figure 6). The host must use a terminal emulation application that supports serial interface communication.

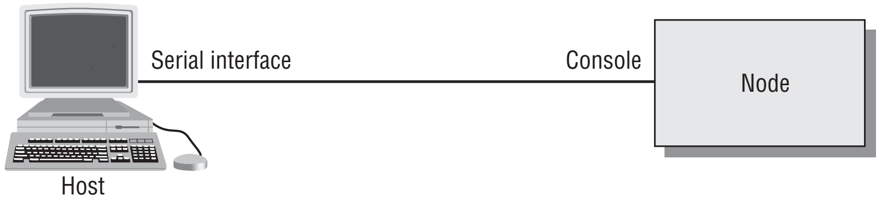


Figure 6. Setup for initial configuration via the console port

Note You do not need to configure IP settings if you access the Patton device via the console port.

Console port procedure

Before using the CLI to enter configuration commands, do the following:

1. Set up the hardware as described in the getting started guide.
2. Configure your serial terminal as described in the getting started guide.
3. Connect the serial terminal to your Patton device. Use a serial cable according to the description in the getting started guide included with your Patton device.
4. Power on your device. A series of boot messages are displayed on the terminal screen. At the end of the boot sequence, press the **<return>** key and the login screen will be displayed. Proceed with logging in.

Accessing via a secure configuration session over SSH

SSH is the most commonly used and recommended method for connecting to a Patton device. A partial implementation of secure shell according RFC 4251, RFC 4252, RFC 4253 and RFC 4254 is provided. It is possible to open a secure configuration session over SSH to a Patton device.

Note The copy tftp and http functions are still insecure!

The SSH Transport Layer supports the following Algorithms: "ssh-rsa" or 'ssh-dsa" public key for signing, "diffie-hellmann-group1-sha1" and "diffie-hellmann-group14-sha1" for key exchange, "3des-cbc", "aes256-cbc" and "aes128-cbc" for encryption, "hmac-sha1" and "hmac-md5" for data integrity. For user authentication, only the method "password" is supported. On the Connection Layer, only the request for an interactive command shell is supported. After the first startup of Trinity, the RSA or DSA server host key is going to be calculated. The RSA or DSA server host key is calculated only once and always remains the same.

Mode: Configure

Step	Command	Purpose
1	device(cfg)#terminal ssh use auth <AAA profile name>	Set the AAA profile which is going to be used for user authentication. The AAA profile "default" is used when another profile is not specified.

Accessing via a Telnet session

It is way faster than console access. The Telnet host accesses the Patton device via its network interface.

> **Note** If the IP configuration of the Ethernet port (LAN port) is not known or is incorrectly configured, you will have to use the console interface.

Telnet Procedure

Before you begin to use the CLI to input configuration commands, do the following:

1. Set up the Patton device as described in the getting started guide included with your device.
2. Connect the host (PC) or hub to the Patton device as described in the getting started guide.
3. Power on your device and wait until the *Run* LED lights.
4. Open a Telnet session to the IP address shown in the getting started guide.
5. Proceed with logging in.

Using an alternate TCP listening port for the Telnet or SSH server

The following command defines an alternate listening port for the telnet or SSH server.

Mode: Configure

Step	Command	Purpose
1	telnet-server port <port> or ssh-server port <port>	Uses TCP port <port> for accepting telnet or SSH connections

Disabling the Telnet or SSH server

The telnet or SSH server can be disabled using the following command.

Mode: Configure

Step	Command	Purpose
1	device(cfg)# no terminal [telnet | ssh]	Disables the telnet or SSH server

Logging on

Accessing your Patton device via the local console port or via a Telnet session opens a login screen. The following description of the login process is based on a Telnet session scenario but is identical to that used when accessing via the local console port.

The opening Telnet screen you see resembles that shown below. The window header bar shows the IP address of the target Patton device.

A factory preset superuser account with name *admin* and an empty password is available when you first access the unit. For that reason, use the name *admin* after the login prompt and simply press the **<enter>** key after the password prompt.

```
$ telnet 172.16.54.79
Trying 172.16.54.79…
```

```
Connected to 172.16.54.79.
Escape character is '^]'.

Patton Electronics Company FF3310RC
Release: 3.1.0 2013/01/20

Trinity login: admin
Password:

Trinity >
```

Upon logging in you are in operator execution mode, indicated by the “>” as command line prompt. Now you can enter system commands.

Note Details on the screen, such as the IP address in the system prompt and window header bar, may be different on your unit.

IMPORTANT You are responsible for creating a new administrator account to maintain system security. Patton Electronics accepts no responsibility for losses or damage caused by loss or misuse of passwords. Please read the following sections to secure your network equipment properly.

Selecting a secure password

It is not uncommon for someone to try to break into (often referred to as *hacking*) a network device. The network administrator should do everything possible to make the network secure. Carefully read the questions below and see if any applies to you:

- Do your passwords consist of a pet’s name, birthdays or names of friends or family members, your license plate number, social security number, favorite number, color, flower, animal, and so on?
- Do you use the same password repeatedly? (Example: Your ATM PIN, cell phone voice mail, house alarm setting code, etc.)
- Could your password or a portion thereof be found in the dictionary?
- Is your password less than six characters long?

To prevent unauthorized access, you should select passwords that are not dictionary words or any of the above-mentioned examples. Every password should be at least 6 characters long and include at least one capital letter, one number, and one lowercase letter.

A good example of a password is: *3Bmshtr*

You are probably asking yourself, “How am I going to remember that?” It’s easy, the password above is an acronym taken from: “three blind mice, see how they run.” Making a good password is that easy—but please, don’t use the above example password for your Patton device!

Password encryption

Unencrypted passwords can be stolen by hackers using protocol analyzers to scan packets or by examining the configuration file—to protect against that type of theft, Trinity encrypts passwords by default. Encryption prevents the password from being readable in the configuration file.

- Plain text
- Encrypted text (for example, the password mypassword always appears in encrypted form as *HUAvCYeILW-Zz3hQvS0IEpQ== encrypted* when doing a **show** command)

The command **show running-config** always displays the passwords in encrypted format. To encrypt a password, enter the password in plain format and retrieve the encrypted format from the running-config or store it permanently into the startup-config (with the command **copy running-config startup-config**).

Factory preset superuser account

Trinity contains a factory preset superuser account with the name *admin* (no passwords). When a new superuser account has been defined in the configuration, the preset admin account will delete after reboot. You can create more than one superuser account, but there has to be at least one superuser account defined. If, for some reason, the last superuser account is deleted, the factory preset administration account with the name *admin* and an empty password is automatically recreated.

Configuring operators, administrators, and superusers

Creating an operator account

The operator can only show a small set of states for supervising the functionality of a device. He does not configure or change anything on the device.

- **Default not allowed:**
  - Config write commands
  - Exec commands
  - Copy config commands
  - Software upgrade
- **Default allowed:**
  - Debug/trace commands
  - Show commands (not all)
  - Show running config (incomplete)
- **Exceptions to default behavior**: show accounts (and related) is not allowed.

Creating a new operator account is described in the following procedure:

Mode: Operator execution

Step	Command	Purpose
1	*device*>enable	Enters administration execution mode

Step	Command	Purpose
2	*device*#configure	Enters configuration mode
3	*device(cfg)#* operator *name* password *password*	Creates a new operator account *name* and password *password*
4	copy running-config startup-config	Saves the change made to the running configuration of the Patton device, so that it will be used following a reload

Example: Create an operator account

The following example shows how to add a new operator account with a login name *support* and a matching password of *s4DF&qw*. The changed configuration is then saved.

```
    device>enable
    device#configure
    device(cfg)#operator support password s4DF&qw
    device(cfg)#copy running-config startup-config
```

Creating an administrator account

The administrator can configure and debug a device in operation. The main exception is he is not allowed to configure user accounts. Therefore all configuration which allows to remove accounts from a device in an easy way are not allowed. Remark: We cannot prevent a physical present administrator/operator to gain supervisor access to a device via erasing the config from uboot.

- **Default not allowed:**
  - Any related to Provisioning
  - Any related to AAA
  - Any related to User accounts
- **Default allowed:**
  - Software upgrade
  - Copy config commands
  - Config write commands
  - Exec commands
  - Debug/trace commands
  - Show commands
  - Show running config (complete)

Creating a new administrator account is described in the following procedure:

Mode: Operator execution

Step	Command	Purpose
1	*device*>enable	Enters administration execution mode
2	*device*#configure	Enters configuration mode

Step	Command	Purpose
3	*device(cfg)#* administrator *name* password *password*	Creates a new administrator account *name* and password *password*
4	*device(cfg)#*copy running-config startup-config	Permanently stores the new administrator account parameters.

Example: Create an administrator account

The following example shows how to add a new administrator account with a login name *super* and a matching password *Gh3*Ke4h*.

```
device>enable
device#configure
device(cfg)#administrator super password Gh3*Ke4h
device(cfg)#copy running-config startup-config
```

The web user interface can be accessed through two different modes. The enhanced or the basic GUI. In the enhanced GUI, all the configuration options are present. In the basic, only the wizards. The basic GUI is enabled per user via the CLI.

Mode: Configure

Step	Command	Purpose
1	*device(cfg)*#administrator <username> password <user-password> terminal-type http web-basic-only	Add a new user who can only access and configure the device via the basic web GUI.

Creating a superuser account

The superuser is the main administrator which can configure the device and manages user accounts.

- **Default allowed**: Everything
- **Exceptions to default behavior**: engineer exec commands are not allowed

Creating a new superuser account is described in the following procedure:

Mode: Operator execution

Step	Command	Purpose
1	*device*>enable	Enters administration execution mode
2	*device*#configure	Enters configuration mode
3	*device(cfg)#* superuser *name* password *password*	Creates a new superuser account *name* and password *password*
4	*device(cfg)#*copy running-config startup-config	Permanently stores the new superuser account parameters.

Example: Create a superuser account

The following example shows how to add a new superuser account with a login name *super* and a matching password *Gh3*Ke4h*.

```
device>enable
device#configure
device(cfg)#superuser super password Gh3*Ke4h
device(cfg)#copy running-config startup-config
```

Displaying the CLI version

This procedure displays the version of the currently running CLI.

Mode: Operator execution

Step	Command	Purpose
1	*device*>show version cli	Displays the CLI version

Example: Displaying the CLI version

The following example shows how to display the version of the current running CLI on your device, if you start from the operator execution mode.

```
device>show version cli
CLI version: 3.00
```

Displaying account information

You can use the **show** command to display information about existing administrator and operator accounts. This command is not available for an operator account.

The following procedure describes how to display account information:

Mode: Administrator execution

Step	Command	Purpose
1	*device*#show accounts	Displays the currently-configured administrator and operator accounts.

Example: Display account information

The following example shows how to display information about existing administrator and operator accounts.

```
device#show accounts
# UserName AccessLevel Status
0 super superuser (logged out:0)
1 admin administrator (logged out:0)
2 op operator (logged out:0)
```

Checking identity and connected users

The **who** command displays who is logged in or gives more detailed information about users. Depending on the execution mode, the command displays varying information. In administrator execution mode, the command output is more detailed and shows information about the ID, user name and location. In operator execution mode, only the user name being used at the moment is reported, which helps checking the identity.

Mode: Administrator or operator execution

Step	Command	Purpose
1	*Trinity(cfc)*#who	Shows more detailed information about the users ID, name, state, idle time and location
or		
	Trinity>who	Shows the user login identity

Example: Checking identity and connected users

The following example shows how to report who is logged in or more detailed information about users, depending on the execution mode in which you are working.

Used in administrator execution mode:

```
Trinity(cfg)#who
# User Name          Login Time                 Location
0 admin                 01/01/2000 00:08:59     console
1 admin                 01/01/2000 00:11:36     telnet 172.16.54.135:55404
```

Used in operator execution mode:

```
Trinity>who
You are operator support
```

Command index numbers

A command index number (indicated by the boldface **1**, **2**, and **3** index numbers in the example below) indicates the position of a command in a list of commands (that is, a command with index *1* will appear higher in the configuration file than one with index *3*).

```
192.168.1.1(pf-voip)[default]#show running-config
...
profile voip DEFAULT
codec 1 g711ulaw64k rx-length 20 tx-length 20
codec 2 g711alaw64k rx-length 20 tx-length 20
codec 3 g723-6k3 rx-length 30 tx-length 30
dejitter-max-delay 200
...
```

Commands that make use of index numbers always show the index in the running config. However, the index can be omitted when entering the command. If you enter such a command with an index, it is inserted into list at the position defined by the index. If you enter such a command without an index, it is placed at the bottom of the list. Also, you can change a commands position in a listing (moving it up or down in the list) by changing its index number.

Example 1: Moving the G.723 codec from position *3* in the list to position *1* at the top of the list.

Listing before changing the G.723 codec index number:

```
profile voip DEFAULT
codec 1 g711ulaw64k rx-length 20 tx-length 20
codec 2 g711alaw64k rx-length 20 tx-length 20
codec 3 g723-6k3 rx-length 30 tx-length 30
dejitter-max-delay 200
```

```
...
```

Listing after changing index number:

```
192.168.1.1(pf-voip)[default]#codec 3 before 1
192.168.1.1(pf-voip)[default]#show running-config
...
profile voip DEFAULT
codec 1 g723-6k3 rx-length 30 tx-length 30
codec 2 g711ulaw64k rx-length 20 tx-length 20
codec 3 g711alaw64k rx-length 20 tx-length 20
dejitter-max-delay 200
...
```

Note Succeeding indexes are automatically renumbered.

Example 2: Moving the G.723 codec back position 3
This command moves the G.723 codec from the top to third place. As a result, the other two codecs move up in the list as their indexes are automatically renumbered to accommodate the new third-place codec.

```
192.168.1.1(pf-voip)[default]#codec 1 after 3
192.168.1.1(pf-voip)[default]#show running-config
...
profile voip DEFAULT
codec 1 g711ulaw64k rx-length 20 tx-length 20
codec 2 g711alaw64k rx-length 20 tx-length 20
codec 3 g723-6k3 rx-length 30 tx-length 30
dejitter-max-delay 200
...
```

Example 3: Inserting a codec at a specific position in the list.
This command assigns the G.729 codec the index number 1 so the codec appears at the top of the list.

```
192.168.1.1(pf-voip)[default]#codec 1 g729 tx-length 30 rx-length 30 silence-sup-
pression
192.168.1.1(pf-voip)[default]#show running-config
...
profile voip DEFAULT
codec 1 g729 rx-length 30 tx-length 30 silence-suppression
codec 2 g711ulaw64k rx-length 20 tx-length 20
codec 3 g711alaw64k rx-length 20 tx-length 20
 codec 4 g723-6k3 rx-length 30 tx-length 30
dejitter-max-delay 200
...
```

Ending a Telnet, SSH or console port session

Use the **logout** command in the operator or administration execution mode to end a Telnet or console port session. To confirm the **logout** command, you must enter `yes` on the dialog line as shown in the example below.

Mode: Operator execution

Step	Command	Purpose
1	*device*>logout	Terminates the session after a confirmation by the user.

Example: End a Telnet or console port session

The following example shows how to terminate a session from the administrator execution configuration mode.

```
device>logout
Press 'yes' to logout, 'no' to cancel:
```

After confirming the dialog with "yes", the Telnet session is terminated.

Note Using the command **exit** in the operator execution mode also terminates a Telnet or console port session, but without any confirmation dialog.

Step	Command	Purpose
1	*device*>enable	Enters administration execution mode
2	*device*#configure	Enters configuration mode
3	*device(cfg)#* superuser *name* password *password*	Creates a new superuser account *name* and password *password*
4	device(cfg)# cli config defaults	Generate a command even if it reflects the default setting (Default: Disabled)

Chapter 5 Creating CLI Action Scripts

Chapter contents

Introduction

Trinity's event-driven user-programmed hook system allows users to create a specific CLI script to execute on a specific event (for example, link down on the IP interface).

Action Script Task List

To configure Action Scripts, perform the tasks in the following sections:

- Creating an Action Script
- Conditions
  - Events
- Actions

Creating an Action Script

There are several parameters that need to be configured in order for the Action Script to properly function. Those steps are listed below.

M ode:configure

Step	Command	Purpose
1	**device(cfg)#[no] actions**	Enters the actions configurations
2	**device(act)[<rule-name>]#[no] rule <name>**	Creates/deletes a rule

Conditions

There can be more than one condition per rule. The rule will be executed each time one of the conditions match the event. This gives the possibility to trigger an Action Script by several events. All events are deferred until the startup-config is parsed to avoid any error due to partial configuration parsing.

M ode: actions

Step	Command	Purpose
1	**device(act)[<rule-name>]#[no] condition <group-name> <source-name> <event-name> [initial]**	Add or remove a condition to the rule. The rule is executed as soon as one of the conditions matched an event. The *initial* option means that the condition will only be matched by the first event (until next reboot). This can be used for if you want to only match the first link up event after boot up.

Context CS Events:

Triggers an action script based on ISDN Interface state change, i.e when ISDN interface is up or down.

Step	Command	Purpose
1	**device(act)[<rule-name>]#[no] condition cs isdn:<if-name> {LINKUP | LINKDOWN} [initial]**	Add or remove a condition to the rule based on events within the context cs ISDN interfaces.

Context IP Events:

Triggers an action script based on IP address state change, i.e when IP address/interface is up or down.

Step	Command	Purpose
1	**device(act)[<rule-name>]#[no] condition ip {address | interface}:<if-name>.<label> {LINKUP | LINKDOWN} [initial]**	Add or remove an IP address condition to the rule, where <if-name> is the IP interface name and <label> is the IP address label.

SIP Gateway Events:

Triggers an action script based on the SIP Registration status, i.e when device is Registered.

Step	Command	Purpose
1	**device(act)[<rule-name>]#[no] condition sip gateway:<gw-name> {event-name} [initial]**	Add or remove SIP Gateway condition to the rule, where <gw-name> is the Context SIP-Gateway name. Possible events are: **NOTIFY_CHECK_SYNC_NORELOAD** - See Chapter 49, section "SIP Notify Check-Sync Event" for more information **NOTIFY_CHECK_SYNC_RELOAD** - See Chapter 49, section "SIP Notify Check-Sync Event" for more information **NOT_REGISTERED** - Event will come if the last registration unregistered or expired for the specified gateway **REGISTERED** - Event will come if the first registration was successful for the specified gateway

System NTP Events:
Triggers an action script based on NTP events, i.e when the system NTP server initially sets the time.

Step	Command	Purpose
1	**device(act)[<rule-name>]#[no] condition system ntp { TIME_INITIALIZED | TIME_SYNCHRONIZED } [initial]**	Add or remove a timer condition to the rule. The first time the NTP server sets the time, the system ntp **TIME_INITIALIZED** is triggered. System ntp **TIME_SYNCHRONIZED** is then triggered when ntp reaches, for the first time, the synchronized state.

System Timer Events:
Triggers an action script using the system timer.

Note A system timer must already be configured in order to execute the commands below. See **Chapter 7, section "Timed Execution of CLI Command"** for more information.

Mode: actions

Step	Command	Purpose
1	**device(act)[<rule-name>]#[no] condition system timer:<timer-name> TIMEOUT [initial]**	Add or remove a timer condition to the rule.

Mode: configure

Step	Command	Purpose
1	**device(cfg)#[no] timer <name> ... trigger-event-timeout**	Create a timer that generates an action TIMEOUT event.

Actions
Defines a CLI Command script that will be executed when the condition events occur.

Mode: actions

Step	Command	Purpose
1	**device(act)[<rule-name>]# [no] action [<index>] <CLI_script>**	• Can be multiline. ([<index>] defines line number). For more help on indexing and list operations, see ACL help. • <CLI_script> commands will be executed as if they were typed one-by-one in configuration mode.

Example:

```
[node](act)[rule1]# action “port bri 0 0”
[node](act)[rule1]# action shutdown
```

The example above will have the same effect as if user typed the command below in configuration mode.

```
port bri 0 0
shutdown
```

Note Observe the use of quotes to write multi-word commands.

Chapter 6 System Image Handling

Chapter contents

Introduction

System image handling management is a complex and feature rich system allowing a user to perform various upgrades on the devices. It allows a user to perform full upgrades and partial upgrades. It allows you to upgrade system configuration(s) seamlessly. The upgrades tasks are supported both from the CLI and WMI. You can copy files to flash from TFTP and local flash space. You can also upgrade from HTTP.

System image handling task list

To load and maintain system images, perform the tasks described in the following sections:

- Displaying system image information
- Displaying Update Status Information
- Copying system images from a network server to the Flash memory
- Erase inactive image on dual-image system
- Switch to the inactive image

Displaying system image information

This procedure displays information about system images and driver software.

Mode: Administrator execution

Step	Command	Purpose
1	**device# show system image**	Lists the system software release version, information about optional interface cards mounted in slots and other information that is the currently running system software. If you have just completed a download of new system software from the tftp server, you must execute the reload command in order to be running with the new system software. This applies equally to driver software. In some cases, the device may reboot itself.

```
device(cfg)#show system image
Software Image #1
===============================================
Image State : active, next
Build Version : 3.5.7-15061
Build Date : 2015/05/01
Build Number : 15061
Build Type : Release
Software Image #2
===============================================
Image State : inactive
Build Version : 3.4.2
Build Date : 2014/03/26
Build Number : 1
Build Type : Release
```

Step	Command	Purpose
1	**device#show system version** **or** **device#show version**	Show Software and Hardware versions. “show version” is a shortcut command for “show system version”

```
device(cfg)#show system version
Software Version     : 3.9.2-15121
Hardware Version     : 2
Hardware Revision    : 1

device(cfg)#show version
Software Version     : 3.9.2-15121
Hardware Version     : 2
Hardware Revision    : 1
```

Displaying Update Status Information

Mode: Operator/Admin execution

Step	Command	Purpose
1	**device>show update status [continuously]**	Shows the current update status.
2	**device>show update progress**	Shows the current update progress. This command is only available if the upgrade was manually started (e.g. by entering copy tftp://.. flash:command). Enter CTRL-C to break the command.

```
device>show update status
System update status
====================

  Status:                                 not updated
  Last state:                             done



deice>show update progress
% No upgrade in progress
```

Copying system images from a network server to flash memory

As mentioned previously, the system image file contains the application software that runs Trinity; it is loaded into the flash memory at the Patton Electronics Co. factory. Since most of the voice and data features of the Device are defined and implemented in the application software, upgrading to a new release might be necessary if you want to have additional voice and data features available. A new system image file must be stored permanently into the flash memory of your Device to be present when booting the device. Since the system image file is preloaded at the Patton Electronics Co. factory, you will have to download new Trinity application software only if a major software upgrade is necessary or if recommended by Patton Electronics Co. Under normal circumstances, downloading a system image file should not be needed.

Downloading a new system image file means storing it permanently at a defined location within the Patton device flash memory. To store the system image file, you must use a special download image bundle file. This bundle file contains directions for the system that describe how to handle the system image file and where to store it. The direction for the system upgrade contained in a file called manifest which is a part of the upgrade image.

Mode: Administrator execution

Step	Command	Purpose
1	**device(cfg)# copy tftp://<hostname>/<file> flash: [reload|no-image-switch]**	Downloads the image file from the TFTP server at address <hostname> and starts the system image download process. This progress is visualized with a progress bar, printing dots according to the time elapsed since the start of each upgrade operation. "**reload**" parameter performs a reboot of the system when the upgrade is successful and the image is switched "**no-image-switch**" will only perform an upgrade without switching the image (old copy to flash command "*flash-cfg:*" behavior)

Switch to the inactive image

This new command switches to the inactive image (only available on dual image system).

Step	Command	Purpose
1	**device(cfg)# system image switch**	Switches to the inactive image upon a reload.

Erase inactive image on dual-image system

This new command will erase the inactive partition of a dual-image system. The status of the erased image is set to "invalid".

Step	Command	Purpose
1	**device# erase system image inactive**	Erases the inactive image

Chapter 7 Configuration File Handling

Chapter contents

Introduction

This chapter describes how to upload and download configuration files to and from a Trinity device. This chapter also describes some aspects of configuration file management. Refer to chapter 6, “System Image Handling” on page 74 for more information.

This chapter includes the following sections:

- Shipping configuration (see page 80)
- Configuration file handling task list (see page 80)

All Patton devices are shipped with a configuration file installed in the factory, which is stored in their flash memory.

A configuration file is like a script file containing Trinity commands that can be loaded into the system. Configuration files may also contain only partial configurations. This allows you to keep a library of command sequences that you may want to use as required. By default, the system automatically loads the shipping configuration from the flash memory if no user-specific configuration is defined as the startup configuration.

Changing the current running configuration is possible as follows:

You may change the running configuration interactively. Interactive configuring requires that you access the CLI by using the **enable** command to enter administrator execution mode. You must then switch to the configuration mode with the command **configure**. Once in configuration mode, enter the configuration commands that are necessary to configure your Patton device.

- You can also create a new configuration file or modify an existing one offline. You can copy configuration files from the flash memory to a remote server. Transferring configuration files between the flash memory and a remote system requires the Trivial File Transfer Protocol (TFTP). The TFTP server must be reachable through one of the Patton device network interfaces.

See Chapter 4, "Accessing the CLI" on page 57 for information concerning access to the CLI.

The following sections focus on Trinity memory regions and software components that can be copied within the memory or uploaded/downloaded between a TFTP server and the memory of the Patton device. Since Trinity uses a specific vocabulary in naming those software components, refer to appendix “Trinity Architecture Terms and Definitions” on page 673 to ensure that you understand the concepts. Refer to chapter 6, “System Image Handling” on page 74 for a brief description of how Trinity uses system memory.

Understanding Configuration Files

Configuration files contain commands that are used to define the functionality of Trinity. During system startup, the command parser reads the factory or startup configuration file command-by-command, organizes the arguments, and dispatches each command to the command shell for execution. If you use the CLI to enter a command during operation, you alter the running configuration accordingly. In other words, you are modifying a live, in-service system configuration.

```
bind interface LAN router
  no shutdown

port ethernet 0 0
medium auto
encapsulation ip
bind interface LAN router
no shutdown

port ethernet 0 1
medium 10 half
encapsulation ip
bind interface WAN router
no shutdown
```

Figure 7. Sample configuration file

Each configuration file stored in the flash memory needs a unique name. The user has to assign a file name to any user-specific configuration. Trinity predefines some names for configuration files. These are the shipping configuration (*shipping-config*), startup configuration (*startup-config*), minimal configuration (*minimal-config*) and running configuration (*running-config*) file names. Refer to appendix A, "Trinity Architecture Terms and Definitions" on page 673 to learn more about configuration file types.

Shipping Configuration

Patton devices are delivered with a *shipping configuration* in the logical region *config:*. This shipping configuration initially parameterizes the most useful network and component settings of Trinity.

Once a user-specific configuration is created and stored as the startup configuration, the shipping configuration is no longer used, but still remains in the persistent memory. It is possible to switch back to the shipping configuration at any time during the operation of a Patton device configuration. The getting started guide included with your Patton device describes the restoration procedure for restoring the default settings.

Configuration File Handling Task List

This section describes how to create, load, and maintain configuration files. Configuration files contain a set of user-configured commands that customize the functionality of your Patton device to suit your own operating requirements.

The tasks in this chapter assume that you have at least a minimal configuration running on your system. You can create a basic configuration file by using the **configure** command; see section "Modifying the Running Configuration at the CLI" on page 85 for details.

To display, copy, delete, and download or upload configuration files, perform the tasks described in the following sections:

- Copying configurations within the local memory (see page 81)
- Replacing the startup configuration with a configuration from the Flash memory (see page 82)
- Copying configurations to and from a remote storing location (see page 83)
- Replacing the startup configuration with a configuration downloaded from the TFTP server (see page 83)

- Displaying configuration file information (see page 84)
- Modifying the running configuration at the CLI (see page 85)
- Modifying the running configuration offline (see page 86)
- Deleting a specified configuration (see page 87)

Copying Configurations Within the Local Memory

Configuration files may be copied into the local memory in order to switch between different configurations. Remember the different local memory regions in Trinity as shown in figure 8.

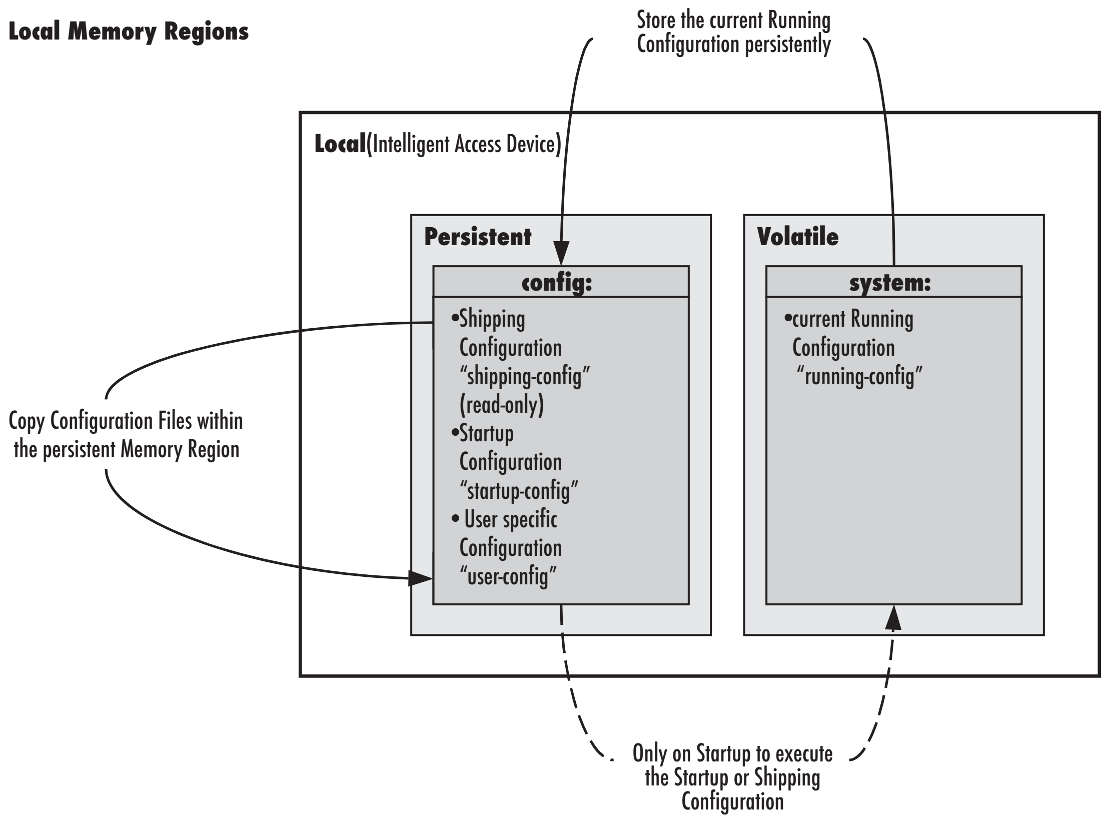

Local Memory Regions
Store the current Running Configuration persistently
Local (Intelligent Access Device)
Persistent
config:
•Shipping Configuration "shipping-config" (read-only)
•Startup Configuration "startup-config"
• User specific Configuration "user-config"
Volatile
system:
•current Running Configuration "running-config"
Copy Configuration Files within the persistent Memory Region
Only on Startup to execute the Startup or Shipping Configuration

Figure 8. Local memory regions

In most cases, the interactively modified running configuration known as the *running-config*, which is located in the volatile memory region *system:*, is copied into the persistent memory region *config*. This running config is stored under the name *startup-config* and replaces the existing startup configuration.

You can copy the current running configuration into the persistent memory region *config*: under a user-specified name, if you want to preserve that configuration.

In addition, an already existing configuration is usually copied into the persistent memory region *config:* by using a user-specified name, for conservation or later activation.

As shown in figure 8 the local memory regions are identified by their unique names, like *config:,* which is located in flash memory, and *system*:, which is the system RAM, i.e. the volatile memory. As already mentioned, configuration files in the same memory region need a unique name. For example, it is not possible to have two configuration files with the name *running-config* in the memory region *config:*.

As you might expect, the **copy** command does not move but replicates a selected source to a target configuration file in the specified memory region. Therefore the source configuration file is not lost after the copy process. There are four predefined configuration file names for which it is optional to specify the memory region, namely *shipping-config*, *startup-config*, *minimal-config* and *running-config*.

Mode: Administrator execution

Step	Command	Purpose
1	***device*#copy {shipping-config | startup-config | minimal-config | running-config | config:** *source-name* } **config:** *target-name*	Copies the selected source configuration file *source-name* as target configuration file *target-name* into the local memory.

Example: Backing up the startup configuration

The following example shows how to make a backup copy of the startup configuration. It is copied under the name backup into the flash memory region *config*:.

```
device#copy startup-config config:backup
```

Replacing the Startup Configuration with a Configuration from Flash Memory

It is possible to replace the startup configuration by a configuration that is already present in the flash memory. You can do so by copying it to the area of the flash memory where the startup configuration is stored.

Mode: Administrator execution

Step	Command	Purpose
1	***device*# copy config:*backup* startup-config**	Replaces the existing persistent startup configuration with the startup configuration *backup* already present in flash memory.

Note The configuration *backup* can be a previously backed up configuration or previously downloaded from a TFTP server.

Copying Configurations To and From a Remote Storage Location

Configuration files can be copied from local memory (persistent or volatile region) to a remote data store. From within Trinity, the remote TFTP server is represented by the memory region *tftp:* in combination with the IP address of the TFTP server and the name and path of the configuration file. We will explain the usage of the remote memory region *tftp:* in the following section more detailed. Another typical task is uploading the current running configuration to the remote data store for backup purpose, or if an extensive configuration file is to be edited on the remote host. In this case the running configuration, named *running-config*, which is to be found in the volatile memory region *system:* is transferred to the TFTP server. On the TFTP server the running configuration is stored to a file whose name is defined as one of the arguments of the **copy** command.

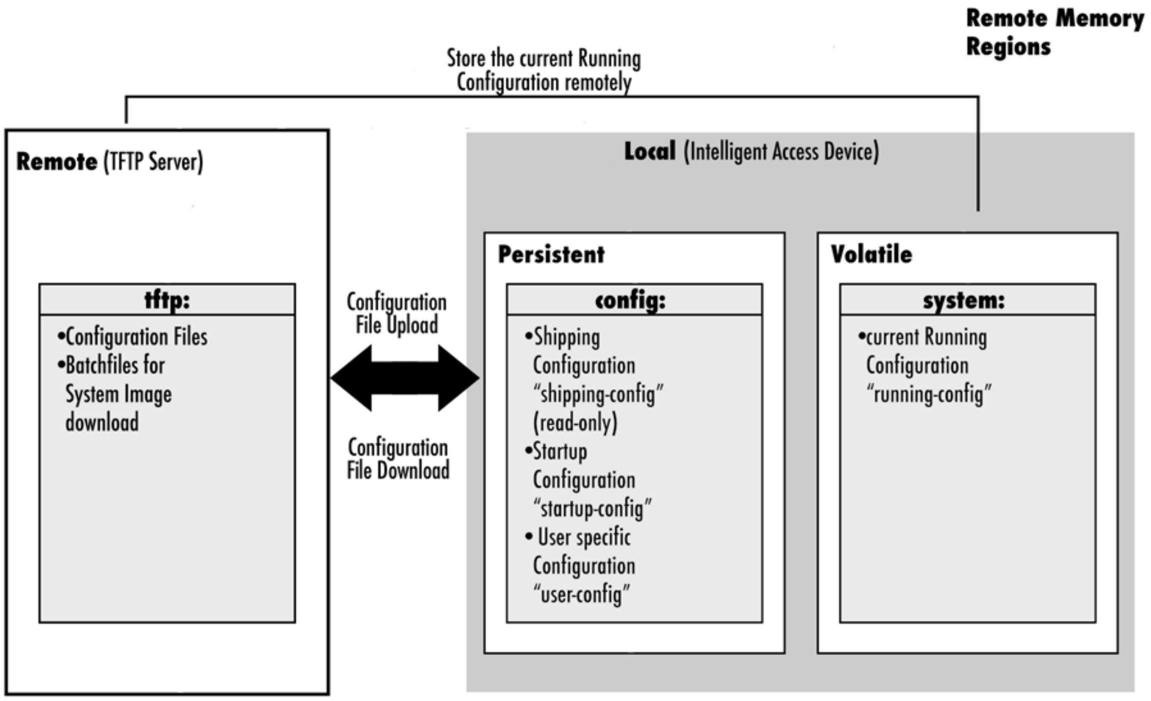

Remote Memory Regions
Store the current Running Configuration remotely
Remote (TFTP Server)
tftp:
•Configuration Files
•Batchfiles for System Image download
Configuration File Upload
Configuration File Download
Local (Intelligent Access Device)
Persistent
config:
•Shipping Configuration "shipping-config" (read-only)
•Startup Configuration "startup-config"
• User specific Configuration "user-config"
Volatile
system:
•current Running Configuration "running-config"

Figure 9. Remote memory regions for Trinity

Finally, configuration files, i.e. the startup configuration or a user-specific configuration that is stored in the persistent memory region *config:* are often uploaded to the remote data store for backup, edit or cloning purposes. The latter procedure is very helpful when you have several Patton devices, each using a configuration which does not greatly differ from the others, or which is the same for all devices. During the configuration of the first Paton device according to your requirements, the running configuration of this device, named *running-config* and located in the volatile memory region *system:,* is edited. Next, the configuration is tested and if everything is as required, the running configuration is copied as startup configuration, named *startup-config*, into the persistent memory region *config:* of the target device. After this, the startup configuration is transferred to the TFTP server, where it can be distributed to other Patton devices. These devices therefore get clones of the starting system if the configuration does not need any modifications.

Replacing the Startup Configuration with a Configuration Downloaded from TFTP Server

From within the administration execution mode, you can replace the startup-configuration by downloading a configuration from the TFTP server into the flash memory area where to store the startup configuration.

Mode: Administrator execution

Step	Command	Purpose
1	***device***(cfg)# copy tftp://*server-ip-address/ [:port]/new-startup* **config:startup-config**	Downloads the configuration file *new-startup* from the TFTP server at address *ip-address* replacing the existing persistent startup configuration. Optionally you can enter the UDP *port* where the TFTP server listens. If the port is not specified, the default port 69 is used. This progress is visualized with a counter, counting up from 0 to 100% according to the downloaded amount of the file size. Should the download fail, an error message *% File Transfer - Get failed* is displayed.

Example: Sample configuration download from the TFTP server

The following example shows how to replace the persistent startup configuration in the flash memory of a Patton device by overwriting it with the configuration contained in the file *new-startup* located on the TFTP server at IP address 172.16.36.80.

1. Download the startup configuration with the **copy** command into the flash memory area where to store the startup configuration.

```
device>enable
device#configure
device(cfg)#copy tftp://172.16.36.80/user/new-startup config:startup-config
Download...100%
device(cfg)#
```

2. Check the content of the persistent startup configuration by listing its command settings with the **show** command.

```
device#show config:startup-config
```

Displaying Configuration File Information

This procedure describes how to display information about configuration files

Mode: Administrator execution

Command	Purpose
show config:	Lists all persistent configurations
show running-config	Displays the contents of the running configuration file
show startup-config	Displays the contents of the startup configuration file
show running-config current-mode	Displays only the running-config of the current mode.
show running-config "<some mode>"	Displays the running-config of any named mode

IMPORTANT

It is recommended that you *never* save a configuration in startup-config or a user-specific configuration with the **cli config defaults** command because the additional list of default commands consumes significant portions of the *config:* memory.

Note Application files can be very long when displayed (by using the **show** command). To make them easier to read, many default commands are not displayed when executing the **show running-config** command. However, the administrator may want to see the entire configuration, including these normally “hidden” default commands. To see all commands, execute the **cli config defaults** command. By issuing a **show running-config** command afterwards, you will see all the commands, a list which is significantly longer. To hide these hidden commands again, issue the **no cli config defaults** command.

Modifying the Running Configuration at the CLI

Trinity accepts interactive modifications on the currently running configuration via the CLI. Interactive configuring needs access to the CLI. Use the **enable** command to enter administrator execution mode, and then switch to the configuration mode by typing the command **configure**. Once in configuration mode, you can enter the configuration commands that are necessary to your Patton device’s operation. When you configure Trinity by using the CLI, the shell executes the commands as you enter them.

When you log in using the CLI, all commands you enter directly modify the running configuration located in the volatile memory region *system:* (or RAM) of your device. Because it is located in volatile memory, to be made permanent, your modifications must be copied to the persistent (non-volatile) memory. In most cases you will store it as the upcoming startup configuration in the persistent memory region *config:* under the name *startup-config*. On the next start-up the system will initialize itself using the modified configuration. After the startup configuration has been saved to persistent memory, you have to restart the device by using the **reload** command to cause the system to initialize with the new configuration.

The execution command **reload** accepts with the following option:

- forced—reloads the system without prompting for confirmation or for saving the running-configuration (no need to type *yes* or *no*). The question whether to save the running-configuration is automatically answered with *no*, the question whether to reload or not with *yes*.
- graceful—reloads the system only if no voice calls are ongoing. If there are voice calls, the system waits until they all are closed to reload.

Mode: Administrator execution

Step	Command	Purpose
1	***device*#configure**	Enters administrator configuration mode
2	Enter all necessary configuration commands.	
3	***device*(cfg)#copy running-config startup-config**	Saves the running configuration file as the upcoming startup configuration
4	***device*(cfg)#reload** [graceful | forced]	Restarts the system

Example: Modifying the running configuration at the CLI

The following example shows how to modify the currently running configuration via the CLI and save it as the startup configuration.

```
device#configure
device(cfg)#…
device(cfg)#copy running-config startup-config
device(cfg)#reload
Press 'yes' to restart, 'no' to cancel: yes
The system is going down
```

Modifying the Running Configuration Offline

In cases of complex configuration changes, which are easier to do offline, you may store a configuration on a TFTP server, where you can edit and save it. Since the Patton device is acting as a TFTP client, it initiates all file transfer operations.

First, upload the running configuration, named *running-config*, from the Patton device to the TFTP server. You can then edit the configuration file located on the TFTP server by using any regular text editor. Once the configuration has been edited, download it back into the device as upcoming startup configuration and store it in the persistent memory region *config:* under the name *startup-config*. Finally, restart the Patton device by using the **reload** command to activate the changes.

Mode: Administrator execution

Step	Command	Purpose
1	***device#*copy running-config tftp://** *device-ip-address[:port]***/*current-config***	Uploads the current running configuration as file current-config to the TFTP server at address *device-ip-address*. Optionally you can enter the UDP *port* where the TFTP server listens. If the port is not specified, the default port 69 is used. This progress is visualized with a counter, counting up from 0 to 100% according to the downloaded amount of the file size. If the upload should fail an error message “% File Transfer - Put failed” is displayed.
2		Offline editing of the configuration file current-config on the TFTP server using any regular text editor.
3	***device#*copy tftp://***device-ip-address/ current-config* **config: *startup-config***	Downloads the modified configuration file current-config from the TFTP server at address device-ip-address into the persistent memory region config: by using the name startup-config. This progress is visualized with a counter, counting up from 0 to 100% according to the downloaded amount of the file size. Should the download fail, an error message “% File Transfer - Get failed” is displayed.
4	***device#*reload**	Restarts the system

Example: Modifying the running configuration offline

The following example shows how to upload the running configuration from the Patton device to the file *current-config* on a TFTP server at IP address 172.16.36.80. The uploaded configuration file is written into the root directory specified by the TFTP server settings, and overwrites any existing file with the same name. Read your TFTP server manual to get a thorough understanding of its behavior. After this, the configuration file is available for offline editing on the TFTP server. Once the configuration file *current-config* has been modified, it is downloaded from the TFTP server, at IP address 172.16.36.80, into the persistent memory region *config:* using the name *startup-config*. It will become active after a reload.

```
device#copy running-config tftp://172.16.36.80/user/current-config
Upload...100%
```

At this point in time, the offline editing of the configuration file *current-config* on the TFTP server takes place.

```
device#copy tftp://172.16.36.80/user/ current-config config:startup-config
Download...100%
device#reload
Press 'yes' to restart, 'no' to cancel: yes
The system is going down
```

Deleting a Specified Configuration

This procedure describes how to delete configuration files from the Patton device flash memory region *config:*.

Mode: Administrator execution

Step	Command	Purpose
1	***device*#show config:**	Lists the loaded configurations
2	***device*#erase config:name**	Deletes the configuration *name* from the flash memory.

Example: Deleting a specified configuration

The following example shows how to delete a specific configuration from among a set of three available configurations in Flash memory. The configuration named *minimal* is to be deleted, since it is no longer used.

1. Use the command **show config:** to list all available configurations.

```
device#show config:
Persistent configurations:
backup
minimal
startup-config
shipping-config
```

2. Delete the configuration named *minimal* explicitly.

```
device#erase config:minimal
```

3. Enter again the command **show config:** to check if the selected configuration was deleted successfully from the set of available configurations.

```
device#show config:
Persistent configurations:
backup
startup-config
shipping-config
```

Chapter 8 System Licensing and Preferences

Chapter contents

Introduction

This chapter describes how to configure system licensing in Trinity.

Managing Feature License Keys

Several features of the firmware require a system specific license key to be installed to enable the feature. The license key can be manually typed (or copied and pasted) in a console or Telnet window. The procedures are described below.

Mode: Configure

Step	Command	Purpose
1	**[*device*](cfg)#install license** <*license*>	Installs the specified license permanently

Example: Installing license keys from the console

The following example shows the command used to install license keys manually on the console.

```
device(cfg)#install license 10011002R1Ws63yKV5v28eVmhDsVGj/JwKqIdpC4Wr1BHaN-
tenXUYF/
2gNLoihifacaTPLKcV+uQDG8LJis6EdW6uNk/
GxVObDEwPFJ5bTV3bIIfUZ1eUe+8c5OpCCd7PSAe83Ty2c/
CnZPSlEjIrVlJrr8VhOr1DYxkEV9evBp+tSY+y9sCeXhDWt5Xq15SAPlznTLQmym7fDakvm+zltzswX/
KX13sdkR0ub9IX4Sjn6YrvkyrJ2dCGivTTB3iOBmRjV1u
```

After installing license keys, you can check if the license keys have been added successfully to your system using the following:

Mode: Configure

Step	Command	Purpose
1	**[*device*](cfg)#show system licenses**	Shows installed licenses

Dynamic licenses can now be removed in the CLI. This completely removes the license and license key from the licenses file. A reboot is required before the change is applied.

Mode: Configure

Step	Command	Purpose
1	**node> erase license <name>**	Erase the given license

Example:

```
06D609(cfg)#show system licenses
Software Licenses
===============================================
                       IP Tunnels
                       Routing
                       Development
                       Time Slot Interchanger
```

```
Clock Source Control
Routing
DHCP Server
Network Address Translation
RIP
DS0 Mapping
```

Chapter 9 AAA Configuration

Chapter contents

Introduction

This chapter provides an overview of the AAA (Authentication, Authorization, and Accounting) component and describes how to configure the TACACS+ client, a subpart of the AAA component. It is important to understand how AAA works before configuring the TACACS+ client. This chapter also describes the local database accounts configuration, which is another subpart of AAA.

To use the authentication and authorization service on Trinity, you have to configure the AAA component, the TACACS+ component and the local database accounts.

This chapter includes the following sections:

- The AAA component
- TACACS+ configuration
- Configuration of the local database accounts

The AAA Component

Authentication, authorization, and accounting is a term for controlling access to client resources, enforcing policies, auditing usage, and providing information necessary to invoice users for services.

Authentication provides a way of identifying a user (usually in the form of a login window where the user is expected to enter a username and password) before allowing access to a client. The AAA component compares the user's authentication login information with credentials stored in a database. If the information is verified, the user is granted access to the network. Otherwise, authentication fails and network access is denied.

Following authentication, authorization determines the activities, resources, or services a user is permitted to access. For example, after logging into a system, a user may try to issue commands, the authorization process determines whether the user has the authority to issue such commands.

Accounting, which keeps track of the resources a user consumes while connected to the client, can tally the amount of system time used or the amount of data transferred during a user's session. The accounting process records session statistics and usage information that is used for authorization control, billing, and monitoring resource utilization.

AAA information can be stored in a local database or in a database on a remote server. A current standard by which network access servers interface with the AAA server is the TACACS+ Service (Terminal Access Controller Access-Control System Plus).

Figure 10 on page 93 illustrates the authentication procedure for a user logging into a SmartNode that is configured to use RADIUS as authentication method.

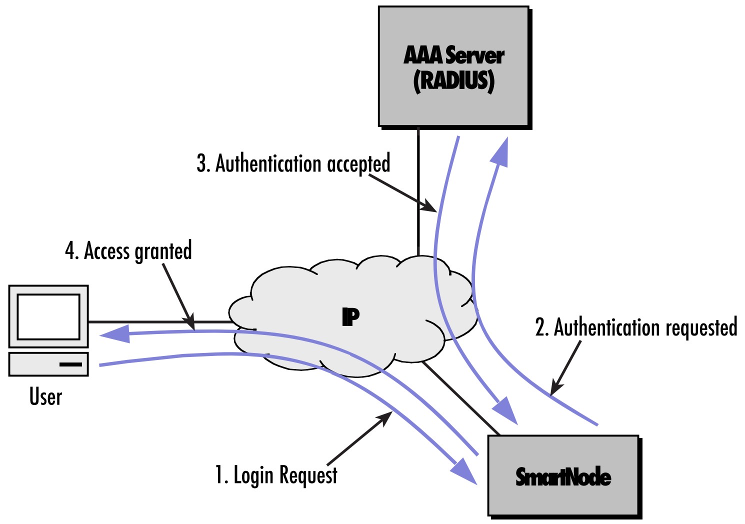


Figure 10. Authentication procedure with a TACACS+ server

General AAA Configuration

The AAA component consists of AAA profiles and AAA methods. A service (e.g. Telnet) has to specify a profile it wants to apply to all login requests. The profile then specifies the sequence in which methods are applied to obtain AAA information. Figure 11 illustrates the correlation between the Telnet login and console login services.

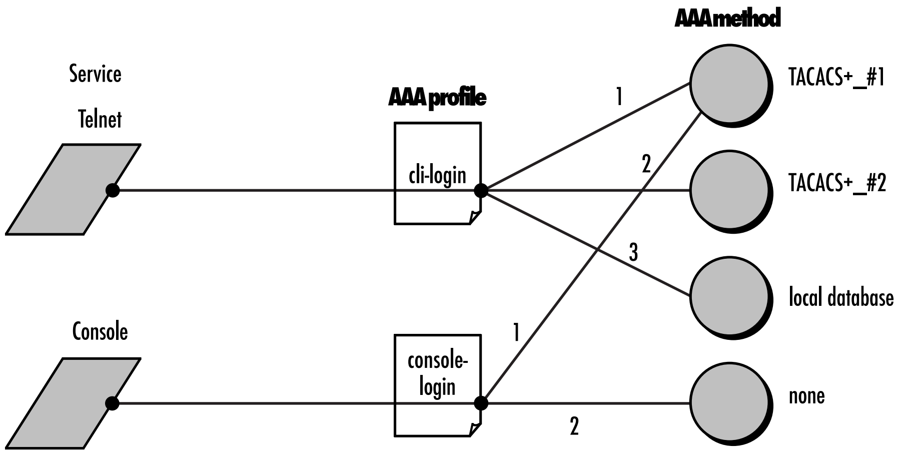


Figure 11. How to use AAA methods and AAA profiles

The Telnet service uses an AAA profile called *cli-login*. This profile specifies that the following methods are used in the order they appear in the configuration:

1. Query TACACS+ server TACACS+_#1.
2. Query TACACS+ server TACACS+_#2.

3. Query the local database (see the "Configuring operators, administrators, and superusers" section for information on how to configure the local database).

If, e.g. TACACS+_#1 is not available, TACACS+_#2 will be queried after a timeout. But if TACACS+_#1 gives an answer that rejects the login request, the remaining methods are not used and the login is denied. The same applies to the console service, which uses the profile console-login. This profile uses the following sequence of methods:

1. Ask TACACS+ server TACACS+_#1.
2. Ask predefined method none. This method always grants access as system operator.

If TACACS+_#1 is not available, access will be granted by the method none. If TACACS+_#1 rejects the login request, console access is denied. If TACACS+_#1 confirms the request, console access is granted.

Configuring TACACS+ client

If the AAA profiles you have defined make use of the TACACS+ AAA method, you must configure the corresponding TACACS+ clients. To configure TACACS+ client, do the following steps:

Mode: Configure

Step	Command	Purpose
1	***node*(cfg)# tacacsplus-client** <name>	Adds a TACACS+ client with the name *ame-* and enters TACACS+-client configuration mode.
2	***node*(tacplus)**[device]**# server** host-name> [<port>]	Sets the host name (or IP address) of the remote TACACS+ server. If no port is specified then the default port 49 is automatically set.
3	***node*(tacplus)**[device]**# [no] shared-secret** <key>	Sets the password shared between the TACACS+ client and the remote TACACS+ server. When no password is set, then an empty password is sent during authentication.

Configuring TACACS+ server

Authentication

During authentication process the client send a standard ASCII authentication request to the server containing the user name and password. Other authentication type (PAP, CHAP, ARAP and MSCHAP) are not supported by the client. Users with password should be configured on the server using the standard ASCII authentication. An authentication request is usually followed by an Authorization request.

Authorization

Authorization request are usually sent after an authentication request and are used to ask the server if the user is allowed to use the requested service. An authorization request always contains the requested service. The client supports only the following services:

Service	Description
telnet	User telnet login request.
ssh	User ssh login request.
fcgi	User webpage login request.
console	User console login request.
call	User call request.

For authorization login request (telnet, ssh, fcgi and console) the server must response with the following attribute-value pair:

Attribute Name	Attribute Value	Description
patton-priv-lvl	operator	The user as operator privilege
	administrator	The user as administrator privilege
	superuser	The user as superuser privilege

For call authorization request, the user field in the request is filled with the e-164 called number. The following attribute-value pairs are added to the request if they are available:

Attribute Name	Attribute Value
nas-identifier	<id>
calling-station-id	<id>
called-station-id	<id>
calling-ip-address	<ip-address>
called-ip-address	<ip-address>

All attributes present on the request must be configured on the server to allow a call! The server can repeat an attribute-value pair, to allow more than one call ID/IP.

Server configuration example

Below is an example of configuration file for the TACACS+ F4.0.4.10 Linux server:

```
# tacacs+ configuration file
# /etc/tac_plus.conf

# set the shared secret key between client and server
key = cle_tacacs

# set the accounting file, where all accounting request are logged
accounting file = /var/log/tac_plus.acct

# users accounts

user = johndoe {
 login = cleartext "normal"    # enter login password in clear text
 name = "John Doe"             # Name description (not used by trinity)

 # You must now list all service that the user is allowed to connect with
 # ( console | telnet | ssh | fcgi ):

 service = telnet {
   patton-priv-lvl = operator  # set the privilege level of the
                               # user (operator|administrator|superuser)
   }

 service = ssh {
   patton-priv-lvl = administrator  # set the privilege level of the user
                                    # (operator|administrator|superuser)
   }
}

# Example with encrypted password

user = johndoe {
 login = des "yrVMIa532Sy.2"        # enter login password encrypted with
                                    # tac_pwd program
name= "John Doe"

 service = telnet {
   patton-priv-lvl = administrator  # set the privilege level of the user
                                    # (operator|administrator|superuser)
   }
 service = ssh {
   patton-priv-lvl = superuser      # set the privilege level of the user
                                    # (operator|administrator|superuser)
   }
}

# call accounts to authorize call

user = 100 {                          # user is the e-164 called number
 service = call {
   nas-identifier     = MyNas
   calling-station-id = 100
   called-station-id  = 200          # Allow to call 200 and 300
   called-station-id  = 300
   calling-ip-address = 192.168.0.1
   called-ip-address  = 192.168.0.2  # Allow called IP address 192.168.0.2 and
192.168.0.3
   called-ip-address  = 192.168.0.3
 }
}
```

Chapter 10 Basic System Management

Chapter contents

Introduction

This chapter describes parameters that report basic system information to the operator or administrator, and their configuration. The following are basic parameters that can be established when setting up a new system (see section "Setting System Parameters" on page 100):

- Defining the system's hostname
- Setting the location of the system
- Providing reference contact information
- Setting the clock

Additionally, the following tasks are described in this chapter:

- Setting the system banner (see section "Setting the System Banner" on page 102)
- Enabling the embedded web server (see section "Configuring and starting the web server" on page 104)
- Using the Trinity Performance Tracker (see section "Trinity Performance Tracker" on page 109)

Basic System Management Configuration Task List

All tasks in the following sections are optional, though some such as setting time and calendar services and system information are highly recommended.

To configure basic system parameters, perform the tasks described in the following sections.

- Managing feature license keys (see page 99)
- Setting system information (see page 100)
- Setting the system banner (see page 102)
- Setting time and date (see page 102)
- Displaying clock information (see page 104)
- Displaying time since last restart (see page 104)
- Configuring and starting the secure web server (see page 105)
- Restarting the system (see page 105)
- Displaying the System Logs (see page 106)
- Displaying Reports
- Exporting System Logs and Reports
- Identifying a unit by flashing all LED's (see page 107)

Managing Feature License Keys

Several features of the firmware require a system specific license key to be installed to enable the feature.

This section describes how to install the feature license keys on your equipment.

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#install license** *license-key*	Install the license key
2		Repeat step 1 for any additional license keys

Example: Installing license keys from the console

The following example shows the command used to install license keys manually on the console.

```
device(cfg)#install license 10011002R1Ws63yKV5v28eVmhDsVGj/JwKqIdpC4Wr1BHaN-
tenXUYF/2gNLoihifacaTPLKcV+uQDG8LJis6EdW6uNk/GxVObDEwPFJ5bTV3bIIfUZ1eUe+8c5OpC-
Cd7PSAe83Ty2c/
CnZPSlEjIrVlJrr8VhOr1DYxkEV9evBp+tSY+y9sCeXhDWt5Xq15SAPlznTLQmym7fDakvm+zltzswX/
KX13sdkR0ub9IX4Sjn6YrvkyrJ2dCGivTTB3iOBmRjV1u
```

After installing license keys, you can check if the license keys have been added successfully to your system using the following command.

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#show system licenses**	Displays all dynamic and static licenses

Example: Displaying installed licenses

The following example shows the command used to display all installed licenses on a system and a sample of its output.

```
device(cfg)#show system licenses
Dynamic Licenses
================

  Name: sip-tls-srtp
  ------------------

    ID:                                 54
    Description:                        SIP TLS and SRTP

  Name: sip-registrar
  -------------------

    ID:                                 55
    Description:                        SIP Registrar

Static Licenses
```

Showing System Resources

The following command will display all available resources including Voice Ports and total number of SIP Legs. The system information includes the following parameters:

- Contact
- Hostname
- Location
- Provider
- Subscriber
- Supplier

Mode: all

Step	Command	Purpose
1	**node#show system resources**	Displays available resources including Voice Ports and total number of SIP Legs

Example:

```
(cfg)#show system resources
Ports
=====

  BRI Ports:                         0
  E1T1 Ports:                        4
  FXS Ports:                         0
  FXO Ports:                         0

VoIP Resource
=============

  Number of DSPs:                    2
  DSP Firmware:                      0
  DSP Channels:                      120
  SIP Legs:                          180
  Transcoding:                       no
  Border Controller:                 no
```

Setting System Parameters

The system information includes the following parameters:

Note By default there is no information specified for any of the following parameters.

- **Contact**: System contact information tells the user how to contact the information service, e.g. the help line of the service provider. The contact information may be any alphanumeric string, including spaces, that is no longer than one line. This entry corresponds to the MIB II system sysContact object.

- **Hostname**: The system name, also called the hostname, is used to uniquely identify the Patton device in your network. The selected name should follow the rules for ARPANET hostnames. Names must start with a letter, end with a letter or digit, and have as interior characters only letters, digits, and hyphens. Names must be 63 characters or fewer. For more information, refer to RFC 1035. This entry corresponds to the MIB II system sysName object. After setting the hostname of the Patton device the CLI prompt will be replaced with the chosen name.
- **Location**: Assigning explanatory location information to describe the system physical location of your device (e.g. server room, wiring closet, 3rd floor, etc.) is very supportive. This entry corresponds to the MIB II system sysLocation object.
- **Provider**: The system provider information is used to identify the provider contact for this Patton device, together with information on how to contact this provider. The provider is a company making services available to subscribers. The provider information may be any alphanumeric string, including spaces, that is no longer than one line. This entry corresponds to the Patton Electronics enterprise-specific MIB provider object.
- **Subscriber**: The system subscriber information is used to get in touch with subscriber for this Patton device, together with information on how to contact this subscriber. The subscriber is a company or person using one or more services from a provider. The subscriber information may be any alphanumeric string, including spaces, that is no longer than one line. This entry corresponds to the Patton Electronics enterprise-specific MIB subscriber object.
- **Supplier**: The system supplier information is used to get in touch with the supplier for this Patton device, together with information on how to contact this supplier. The supplier is a company delivering Patton devices to a provider. The supplier information may be any alphanumeric string, including spaces, that is no longer than one line. This entry corresponds to the Patton Electronics enterprise-specific MIB supplier object.

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#system contact** *information*	Sets the contact information to *information*
2	***device*(cfg)#system hostname** *information*	Sets the hostname to *information*
3	***device*(cfg)#system location** *information*	Sets the location information to *information*
4	***device*(cfg)#system provider** *information*	Sets the provider information to *information*
5	***device*(cfg)#system subscriber** *information*	Sets the subscriber information to *information*
6	***device*(cfg)#system supplier** *information*	Sets the supplier information to *information*

Note If the system information must have more than one word, enclose it in double quotes.

Example: Setting system information

The following example shows the commands used to configure the contact information for your device, if you start from the operator execution mode.

```
device(cfg)#system contact "Bill Anybody, Phone 818 700 1504"
device(cfg)#system hostname device
device(cfg)#system location “Wiring Closet, 3rd Floor”
device(cfg)#system provider “Best Internet Services, contact@bis.com, Phone 818 700
2340”
device(cfg)# system subscriber “Mechanical Tools Inc., jsmith@mechtool.com, Phone
818 700 1402”
device(cfg)# system supplier “WhiteBox Networks Inc., contact@whitebox.com, Phone
818 700 1212”
```

Setting the System Banner

The system banner is displayed on all systems that connect to your Patton device via Telnet, SSH, or a serial connection. It appears at login and is useful for sending messages that affect administrators and operators, such as scheduled maintenance or system shutdowns. By default no banner is present on login.

To create a system banner use the **banner** command followed by the message you want displayed. If the banner message has to be formed out of more than one word the information is enclosed by double quotes. Adding the escape sequence “\n” to the string forming the banner creates a new line on the connected terminal screen. Use the **no banner** command to delete the message.

```
Mechanical Tools Inc.
jsmith@mechtool.com
Phone 818 700 1402

login:
```

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#banner** *message*	Sets the message for the system banner to *message*

Example: Setting the system banner

The following example shows how to set a message for the system banner for your device, if you start from the configuration mode.

```
device(cfg)#banner \n#\n# The password of all operators has changed\n# please con-
tact the administrator\n#"
```

Setting Time and Date

All Patton devices provide time-of-day and date services. These services allow the products to accurately keep track of the current time and date. The system clock specifies year, month, day, hour, minutes, and optionally seconds. The time is in 24-hour format *yyyy-mm-ddThh:mm:ss* and is retained after a reload.

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#clock set** *yyyy-mm-ddThh:mm:ss*	Sets the system clock to *yyyy-mm-ddThh:mm:ss*

Note The integrated SNTP client allows synchronization of time-of-day and date to a reference time server. Refer to chapter 28, “SNTP Client Configuration” on page 283 for more details.

Example: Setting time and date

The following example shows the commands used to set the system clock of your device to August 6, 2001 at 16:55:57, if you start from the operator execution mode.

```
device(cfg)#clock set 2001-08-06T16:55:57
```

Configuring Daylight Savings Time Rules

Trinity allows configuring daylight saving time rules, which affect the local clock offset without changing the configuration. After booting up and loading the configuration, the daylight saving rules are checked and applied automatically. The rules consist of a default-offset and one or multiple dst-rules. The offset of a dst-rule is active if the local clock is between the specified start and stop time of the rule. If the local clock is outside the specified start and stop time of all specified rules, then the default-offset is active.

Note When the DST rule is active, its offset is added to the local default offset. Therefore, the DST offset defines how much time is "shifted" during summer (e.g. 1h).

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#clock local default-offset (+hh:mm | -hh:mm)**	Configures the offset of your time zone from GMT. This offset is used if no other dst rule is currently active. Default: +00:00

Mode: Configure

Step	Command	Purpose
1	**node(cfg)]#clock local dst-rule <name> (+hh:mm | -hh:mm) from <month> <counter> <day-of-week> hh:mm [<year>] until <month> <counter> <day-of-week> hh:mm [<year>]**	Configures a DST rule that enables summer time on a specific day of the week (e.g. last Sunday of March)

Mode: Configure

Step	Command	Purpose
2	**node(cfg)]#clock local dst-rule <name> (+hh:mm | -hh:mm) from <month> <day-of-month> hh:mm [<year>] until <month> <day-of-month> hh:mm [<year>]**	Configures a DST rule that enables summer time on a specific day of the month (e.g. 23rd of September)

Possible parameters for the daylight savings time configuration commands:

Parameter	Possible Values
<month>	(jan I feb I mar I apr I may I jun I jul I aug I sep I oct I nov I dec)
<day-of-week>	(monday I tuesday I wednesday I thursday I friday I saturday I sunday)

Parameter	Possible Values
<counter>	(first | second | third | fourth | last | 1st | 2nd | 3rd | 4th | 5th | 6h | 7th | 8th | 9th | 10th | 11st | 12nd | 13rd | 14th | 15h | 16th | 17th | 18th | 19th | 20th | 21st | 22nd | 23rd | 24th | 25h | 26th | 27th | 28th | 29th | 30th | 31st

Display Clock Information

This procedure describes how to display the current date and time

Mode: Both in operator and administrator execution

Step	Command	Purpose
1	***device*>show clock**	Display the local time.

Example: Display clock information

The following example shows the commands used to display the time and date settings of your device in local time, if you start from the operator execution mode.

```
device>show clock
2001-08-06T16:55:57
```

Display Time Since Last Restart

This procedure describes how to display the time since last restart

Mode: Operator execution

Step	Command	Purpose
1	***device*>show uptime**	Display the time since last restart.

Example:

The following example shows how to display the uptime of your device, if you start from the configuration mode.

```
device>show uptime
The system is up for 54 days, 23 hours, 44 minutes, 18 seconds
```

Configuring and starting the web server

The embedded web server has multiple parameters that are configurable.

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#web-server http**	Enters into the web server configuration mode.
2	***device*(http)# display [advanced|basic]**	Configures how the web-server is displayed. basic - Simplistic view of the GUI with limited configuration parameters. Can view and download/upload config files, firmware and wizards. advanced - Shows all available configuration parameters.
3	***device*(http)#port <port>**	Selects what port the web-server will be accessible on. Default is port 80.

Step	Command	Purpose
4	***device*(http)#[no] shutdown**	Enables/Disables the web-server.
5	***device*(http)# webrefresh <refresh rate (s)>**	This command configures the WEB interface refresh rate.

Configuring and starting the secure web server

HTTPS is now supported using a default SSL certificate that is unique to each device.

Note A web browser will warn when it connects to the web site. This is because the default SSL certificate is self-signed and is not assigned to a specific host-name, so the browser cannot use it to authenticate (i.e. to ensure that it is connected to the web site it thinks it is). Nevertheless, the connection will be encrypted.

Mode: configure

Step	Command	Purpose
1	**node(cfg)#web-server https**	Enters the HTTPS web-server configuration mode.
2	**node(https)#port <port>**	Configures the HTTPS server to listen on TCP port port (default 443).
3	**node(https)#use profile aaa <name>**	Configures HTTPS server to authenticate logins using AAA profile name.
4	**node(https)#webrefresh <seconds>**	Configures status webpages to refresh every seconds seconds.
5	**node(https)#[no] shutdown**	Start or stop the HTTPS server.

Restarting the system

In case the Patton device has to be restarted, the **reload** command must be used. The reload command includes a two-dialog, where the user is allowed to store any unsaved configuration data and finally confirms the system restart.

IMPORTANT

Restarting the system interrupts running data transfers and all voice calls.

Mode: Administrator execution

Step	Command	Purpose
1	***device*#reload**	Restarts the system

The execution command **reload** has been enhanced with the following options:

•**cancel** - Cancels reload/halt

•**forced** - Reloads the system without prompting for confirmation.

•**if-needed** - Only reloads if the system knows that a reload is needed.
•**in <seconds>** - Sets the time in which to perform the reload/halt action.

The following example shows how to restart the running system, if you start from the administrator execution mode.

```
device#reload
Type 'yes' to restart/halt, anything else to cancel:
```

Displaying the System Logs

The system logs contain warnings and information from the system components of Trinirty3. In case of problems it is often useful to check the event or the supervisor logs for information about malfunctioning system components. The event log stores general events such as flash full, DSP failed etc., comparable with the event log on Windows NT. The supervisor log stores information from the system supervisor such as memory full, task failed etc.

System resets may have a number of reasons, the most prominent being a manual reset issued on the Telnet/ console ('reload'). Other reset reasons include power off failures and system failures. In order to pinpoint the problem, the reset log contains the reset cause.

The show log command offers a new argument to suppress color information to be printed. Since Trinity 3.7 some log messages are printed in color, e.g. error messages in red. If log dumps are copied into text files, some editors have problems to render the color codes (ESC sequence) correctly. The 'unformatted' parameter removes these color codes before printing the log.

Mode: Operator execution

Step	Command	Purpose
1	**node>show log boot [debug] [unformatted]**	Displays the console and log messages captured during startup of the unit; the 'debug' option print more details.
2	**node>show log error [unformatted]**	Displays all error messages.
3	**node>show log event [unformatted]**	Shows important events such as link up/down.
4	**node>show log file-transfer [unformatted]**	Displays provisioned file transfers.
5	**node>show log performance {hourly|daily|monthly}**	Displays performance statistics of the device within the specified range.
6	**node>show log reset [unformatted]**	Outputs a list of reset reasons (with date and time).
6	**node>show log supervisor [unformatted]**	Shows a dump of the system supervisor, used, for example, to get information about an unexpected reboot.

Displaying the System Logs

The show reports command provides the 'unformatted' argument as well to suppress color codes in the output. This command is used to dump combined system information. We also changed the order in which the following logs are concatenated:

```
show log reset [unformatted]
```

```
    show log error [unformatted]
    show log event [unformatted]
    show log file-transfer [unformatted]
    show log supervisor [unformatted]
    show log boot debug [unformatted]
    show running-config
```

Mode: Operator execution

Step	Command	Purpose
1	**node>show reports [unformatted]**	Dumps the combined system information with (default) or without color highlighting ('unformatted' parameter).

Exporting System Logs and Reports

The copy command now contains a new "log:" source which allows for exporting all logs and reports to an external TFTP server. By default the log files are exported without color information, but you can include color with the optional 'formatted' argument.

Mode: Operator execution

Step	Command	Purpose
1	**node>copy log:boot tftp://<server-ip>/<path> [formatted]**	Copies the console and log messages captured during startup of the unit to text *filepath* on TFTP server *server-ip*.
2	**node>copy log:boot-debug tftp://<server-ip>/<path> [formatted]**	Copies the detailed console and log messages captured during startup of the unit to text *filepath* on TFTP server *server-ip*.
3	**node>copy log:error tftp://<server-ip>/<path> [formatted]**	Copies all error messages to text file path on TFTP server *server-ip*.
4	**node>copy log:event tftp://<server-ip>/<path> [formatted]**	Copies all important events such as link up/down to text file *path* on TFTP server *server-ip*.
5	**node>copy log:file-transfer tftp://<server-ip>/<path> [formatted]**	Copies performance statistics of the device to tape archive *filepath* on TFTP server *server-ip*.
6	**node>copy log:performance.tar tftp://<server-ip>/<path>**	Copies performance statistics of the device to tape archive *filepath* on TFTP server *server-ip*.
7	**node>copy log:reports tftp://<server-ip>/<path> [formatted]**	Copies combined system information to text file *path* on TFTP server *server-ip*.
8	**node>copy log:reports.tar tftp://<server-ip>/<path> [formatted]**	Copies combined system information to tape archive *filepath* on TFTP server *server-ip*. The 'formatted' argument specifies whether the log files within the generated tape archive contain color codes. **This method (without the 'formatted' argument) is the preferred file format to be sent to Patton support.**
9	**node>copy log:reset tftp://<server-ip>/<path> [formatted]**	Copies a list of reset reasons to text file *path* on TFTP server *server-ip*.

Step	Command	Purpose
10	**node>copy log:supervisor tftp://<server-ip>/<path> [formatted]**	Copies a dump of the system supervisor to text file *path* on TFTP server *server-ip*.

Configuring the blink interval

When there are many Trinity devices in the same location, use this command to flash all the LED's on a specific unit for a specified period of time. This makes identification of the physical unit very easy.

Step	Command	Purpose
1	***device* #blink <seconds>**	Enter an integer for the period of time you want the LED's to flash on the physical unit.

Configuring the Syslog Client

Syslog is a protocol for sending event notification messages across IP networks to message collectors (Syslog server). It uses transport protocol UDP on port 514. A syslog-message exits on the three main part Priority, Header and Message whereas the header is split into Facility and Severity and the header into Timestamp and Hostname. The whole syslog-message (Priority, Header and Message) contains only printable characters and the maximum length is 1024 bytes.

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#syslog-client**	Enters syslog client configuration mode.

Mode: Syslog Client

Step	Command	Purpose
1	***device*(syslog-client)#[no] remote { <ipv4 host> | <ipv6 host> } [tcp | udp] [<port>]**	Creates a new remote destination and enters its configuration mode. The 'no' form of the command removes an existing remote destination. The protocol type and port are optional. If not included, the default UDP port 512 will be used.

Mode: Remote

Step	Command	Purpose
1	***device*(syslog-client)(remote)#[no] facility <service name> <severity>**	Creates a new log expression for a remote destination. It exists on a facility that determines from which source messages must be accepted and a severity that defines up to which level the messages of the given facility must be sent. The 'no' form of the command disables sending of messages from the given facility.

Factory Reset

This command performs the same action as the reset button. It will currently remove the configurations, the logs, the preferences files and the installed TLS keys and certificates present in the flash.

Mode: administrator execution

Step	Command	Purpose
1	***device#* factory-reset**	Reset to factory state

Reset Button

The reset button can be used as follows:

1. Hold the reset button during boot
2. The power LED flashes quickly for 2 seconds, during which time the reset button must remain pressed
3. The power LED will begin a blink pattern (described below)
4. Pressing the reset button will change the blink pattern
5. 10 seconds after the last reset button press, an action will be performed based on the selected pattern

Pattern	Action
1-blink, pause	Boot normally
2-blink, pause	Switch to backup image, then boot (Boot normally if the device only has a single image).
3-blink, pause	Erase all configuration and licenses, then boot.

Trinity Performance Tracker

The performance tracker is a new module of the system supervisor that continuously observes performance measures of the system such as CPU load, memory consumption, IP data traffic, etc. These samples help our support team to understand what load a device was confronted with when facing a problem. To investigate some dynamic issues you will be asked to provide these performance measures to our support team. This is done by uploading the measures as tape archive (tar-file) to your TFTP server with the copy command and send us the tar file by email.

Note The collected data **DO NOT** contain any confidential information such as IP addresses, call records, passwords, etc. If you are interested in the information exchanged with our support team you may open the tar file: You will find three CSV files (comma-separated values), in which each row contains the data of one measurement. The first rows provides information about the columns. The three files correspond to measures collected over the last hour, day, and month, respectively. Next to these CSV files there is a text file with meta-information about the device such as model and host name, serial number, etc.

As mentioned before performance measures are stored in RAM and are therefore lost when a power cycle occurs. However, the storage area survives a soft reboot, for example when manually reloading the device or if a crash occurs. If you suspect a crash to be related to a dynamic problem such as heavy CPU load, voice-call or data traffic, please copy the performance data to your TFTP server immediately after the device came up again. This is because the collected information is stored at the maximum resolution only for the last hour. Next to this explicit way of exporting performance measures the supervisor log also stores the measures of the last 15 minutes before a crash.

The following command uploads the performance measures of the past month since the last power-cycle to a TFTP server.

Mode: Administrator execution

Step	Command	Purpose
1	**device#copy log:performance.tar tftp://<ip-addr>/<path>/performance.tar**	Uploads recent performance measures to a TFTP server.

Note Send the performance.tar file to our support team for further analysis.

Chapter 11 Programmable System-Event Configuration

Chapter contents

Introduction

This chapter describes how to use programmable system events in Trinity. System events are triggered when a subsystem changes its state, for example if the link of an interface comes up. Trinity's event system is programmable: the administrator is able to combine existing events into new types of events. For example, it is possible to trigger the provisioning of a new software image when a custom event pattern occurs. Let us assume we want to provision a software image when the system is up and as soon as the management IP interface is up for some minutes and NTP is synchronized. This chapter explains how to construct and program such composite events.

The event programming system of Trinity is organized in three layers (System Variables, Expressions, and Actions) as depicted in figure 12.

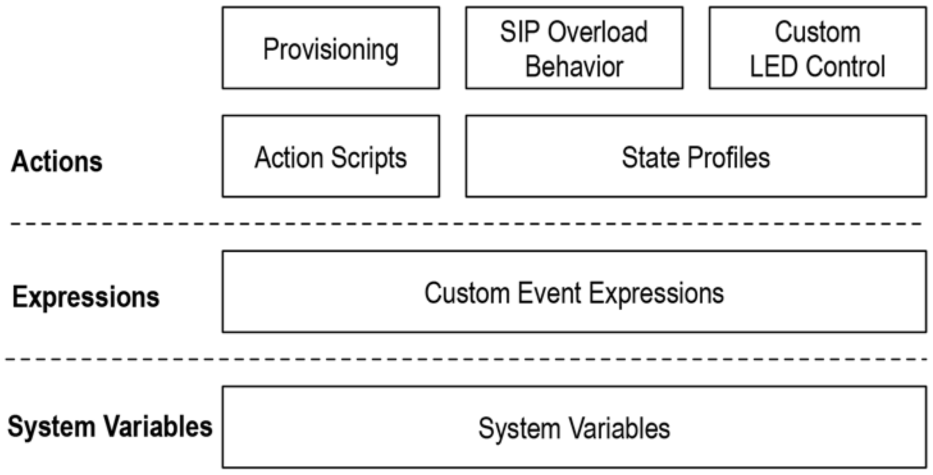


Figure 12. The Event-Programming System is organized in three layers

The remainder of this introduction gives a brief overview of these layers starting at the bottom. The sections following the introduction provide more detailed information and contain examples of how to configure the programmable event system.

System variables

Trinity exposes many subsystem states as system variables upon which action components can react. A system variable essentially is a name-value pair. The name uniquely identifies the variable and the value stores its current state. For example the system variable *sys.up* is initialized to *false*, but is changed to *true* as soon as the system is fully functional after boot-up.

Table 2 lists the most important system variables that Trinity exposes to the user.

Table 2. Essential System Variables

Variable Name	Description
ip.ctx:*ctx-name*. **if**:*if-name*.**up**	True if the IP interface *if-name* in IP context *ctx-name* is up; false if the IP interface is down.

Table 2. Essential System Variables (Continued)

Variable Name	Description
ip.ctx:*ctx-name*. **if**:*if-name*. **addr**:*addr-name*.**up**	True it the IP address *addr-name* in IP interface *if-name* in IP context *ctx-name* is up; false if the IP address is down.
ntp.init	True if the time has been set by NTP.
ntp.sync	True if the time has been fully synchronized over NTP.
sys.cpu.load1m	CPU load, averaged over one minute, in percent.
sys.cpu.util1m	CPU utilization, averaged over one minute, in percent.
sys.ram.avail	Random-Access-Memory currently available in MB.
sys.up	True if the system is up and the startup-config has been applied.

As the name suggests system variables are generated by the system; they cannot be created or modified by the user.

Whenever the value of a variable changes a system event is triggered to all upper layers. Components that refer to the variable in order to execute an action (e.g. start provisioning, switch LEDs, etc.) are triggered automatically if the underlying variable changes.

User-Defined expressions

User-Defined expressions are composite variables created by the device administrator. An expression combines system variables and other expression using Trinity's powerful temporal expression algebra.

For example the administrator might want to know whether all of the following conditions are true (according to the provisioning example introduced above) when

- the system is up,
- the management IP interface is up for at least two minutes, and
- the device has synchronized its clock over NTP.

This overall condition is captured by the following expression entered as CLI configuration command:

```
expression READY "sys.up
              && DEBOUNCEINC(ip.ctx:ROUTER.if:WAN.up, 2m)
              && ntp.sync"
```

As depicted in figure 13 on page 115 the expression creates a new variable called *READY*. Such composite variables are useful to build more complex expressions out of system variables. For example the administrator could create yet another expression called *NOT-READY*, which is true if the above conditions are not met:

```
expression NOT-READY "!READY"
```

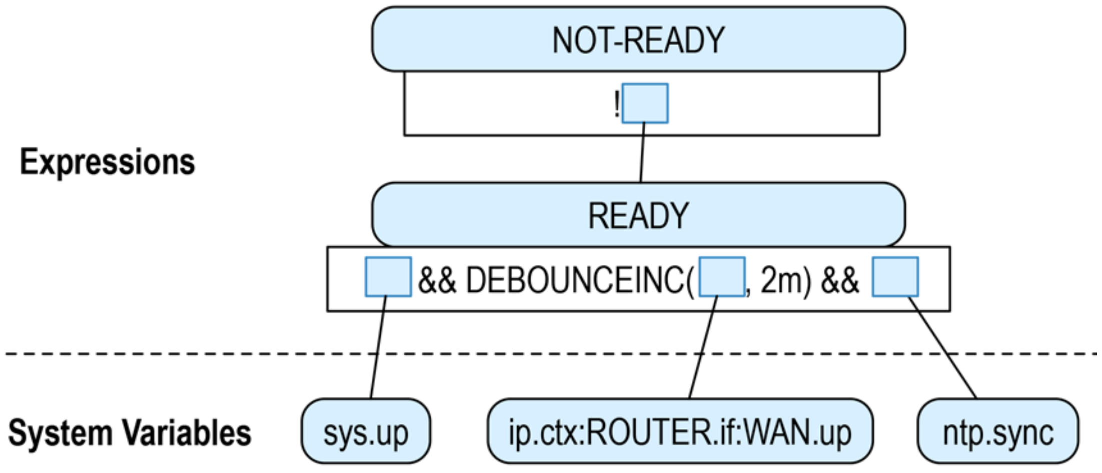


Figure 13. Expression Tree

Actions

Several Trinity components use system variables and expressions to control their behavior. For example the administrator may write an action script to execute a set of CLI commands whenever a variable changes its value from false to true. This powerful feature allows re-configuring virtually every aspect of the Patton device based on complex event patterns.

The following list briefly describes the components that can be linked to system variables and expressions. This list will be expanded in future releases of Trinity:

- **Action Scripts**: **NOT SUPPORTED YET**. Currently, action scripts are triggered by events that are not based on the programmable system events. Support will be added in one of the next releases.
- **SNMP Traps**: **NO SUPPORTED YET**. In the future, it will be possible to generate SNMP traps when a variable changes its value.
- **State Profiles**: Build simple state machines where variable changes trigger transitions between states. (See also section"State Profiles" on page 139.) State profiles map different events into well-defined and customizable states. For example, the *OVERLOAD* state-profile defines three states, *NORMAL*, *WARNING*, and *CRITICAL*, which reflect how the system resources are used based on the current CPU and memory utilization. The state profile is used by other components, for example the SIP call-limiter that drops incoming SIP messages if the system is overloaded.
- **SIP Overload Behavior**: You are able to configure in which circumstances the SIP user agent shall drop incoming and outgoing calls due to an overload situation. This feature uses a state profiles (see above) in combination with a description about which SIP messages to drop in which state. (For more information, refer to Chapter 47, "SIP Overload Configuration" on page 529)

Expression configuration task list

To configure user-defined expressions, perform the tasks in the following sections:

- "Collect information about the variables that build the expression" on page 116
- "Validate an expression (on-the-fly computation)" on page 117)

- "Create/Modify an expression" on page 118
- "Extend a system variable by an expression" on page 119
- "Create/Modify an expression family" on page 120
- "Delete an expression" on page 123

Collect information about the variables that build the expression

User-defined expressions are entered as mathematical equations that combine existing system variables and expressions. In order to build a new expression you need to know which variables to use.

Display existing system variables and expressions

The following CLI command lists all variables alphabetically by name. The second column of the table shows the current value of the variables.

Mode: Operator exec

Step	Command	Purpose
1	***node*>show variable**	Displays a list of all system variables and user-defined expressions (names and values).
2	***node*>show variable *name***	Displays detailed information about the specified variable.

Example: Show all system variables and expressions and their current values:

```
Node>show variable

Variables
================

Variable                                                    Value
------------------------------------------------------------------------
ip.ctx:ROUTER.if:LAN.addr:LAN.up                             true
ip.ctx:ROUTER.if:LAN.up                                      true
ntp.init                                                     true
ntp.sync                                                    false
sip.pktq.delay                                                  0
sys.bootNr                                                    440
sys.cpu.load1m                                                  0
sys.cpu.util                                                    0
sys.cpu.util1m                                                  0
sys.initLevel                                                 100
sys.ram.avail                                                 158
sys.up                                                       true
```

Example: Display all information available about a specific variable (sys.up):

```
node>show variable sys.up

Variable: sys.up
================

Value:       true
```

```
Type:        Boolean
Data-Type:   Boolean
Description: True if the system is up and the config
             has been applied
```

Each variable has a data type and description assigned to it. The data type defines the range of values a variable can take. Next to the Boolean data type (true/false) there are data types for integer and floating-point numbers, text strings, and types for storing absolute time stamps and relative time differences. (For more information, refer to section "Data Types" on page 124.) Variables generated by the system also have a description assigned that describes the purpose and behavior of the variable.

Tracking real-time changes of system variables and expressions

The show commands presented above prints the current state of a variable. Trinity also offers a debug command with which you can observe value changes in real time:

Mode: Operator exec

Step	Command	Purpose
1	***node*>debug variable [detail *level* | full-detail]**	Enables the real-time debugger that prints information about changing variable values to the current terminal. Use detail-level 1 for a brief one-line statement for each variable change.

Example: Show all system variables and expressions and their current values:

```
node>debug variable detail 1
13:59:40.000  VAR      # [sys.cpu.util] 0 -> 2
13:59:50.000  VAR      # [sys.cpu.util] 2 -> 0
14:00:00.000  VAR      # [sys.cpu.util] 0 -> 25
14:00:00.000  VAR      # [sys.cpu.util1m] 1 -> 5
14:00:00.000  VAR      # [sys.ram.avail] 154 -> 150
14:00:10.000  VAR      # [sys.cpu.util] 25 -> 7
14:00:10.000  VAR      # [sys.cpu.util1m] 5 -> 6
```

The trace output above shows how the CPU load and available memory changes over a short period of time. Other than those performance values no variables were changed in the trace period.

Both the show and the debug commands are helpful instruments to debug your user-defined expressions.

Validate an expression (on-the-fly computation)

Before you add an expression to the system configuration permanently it is often useful to check whether the expression is syntactically valid. The CLI compute command can be used to compute an expression on the fly and print the result to the current terminal.

Mode: Operator exec

Step	Command	Purpose
1	***node*>compute *expression***	Computes the mathematical expression on-the-fly and displays the result.

Example: You may use compute instead of the show variable to quickly display the current value of a variable:

```
node>show variable sys.up
Variable: sys.up
================

Value:       true
Type:        Boolean
Data-Type:   Boolean
Description: True if the system is up and the config
             has been applied

node>compute sys.up
true
```

Example: Furthermore, compute allows you to combine variables with mathematical expression. The following example computes some expressions, some of which are syntactically invalid or use variables that do not exist:

```
node>compute 1+2
3
node>compute 1/0
#DIV/0!
node>compute 1:2
% EXPRESSION ERROR:
   1:2
    ^ Unexpected character ':'
node>compute sys.up
true
node>compute !sys.up
false
node>compute sys.doesnotexist
#N/A!
```

Create/Modify an expression

Before you enter the expression command make sure you understand the meaning of the system variables you want to use. For example, if you want to use the sys.ram.avail variable recognize that the value represents the available memory in megabytes. Use the "**show variable** *name*" command to get information about a variable.

The procedure described below creates a new expression:

Mode: Configure

Step	Command	Purpose
1	***node*(cfg)#expression**	Creates a new variable called *name* the value of which is defined by the mathematical *expression*. If an expression variable with the same name already exists it is updated with the new mathematical *expression* specified.

An expression constitutes of two parts: a variable name and a mathematical expression that combines existing variables. Trinity exposes the expression as a variable that can be used in other expression. Therefore each expression must be tagged with a unique name. You cannot re-use names of existing system variables or expressions.

The mathematical equation roughly follows Microsoft Excel formulas. (Refer to section "Operators" on page 126 for a list of all available operations and functions.) If you are familiar with Microsoft Excel you know how to compute the value of a cell based on the value of other cells by specifying the referred-to cells (e.g. A1+A2). Unlike Excel Trinity variables are not organized in a two-dimensional grid. In Trinity variables are identified by their name (e.g. MYVAR1+MYVAR2).

User-defined expression can also be used to define re-usable constants. The following example shows how to compute the area of a circle based on the constant PI and the radius R, both specified as expression variables:

Example: Define a constant PI, a variable *R*, and an expression *AREA* to compute the area of a circle. Change the variable *R* and show that the expression computing the area is updated accordingly.

```
node>enable
node#configure
node(cfg)#expression PI 3.14
node(cfg)#expression R 2
node(cfg)#expression AREA "PI*R*R"
node(cfg)#show variable
System Variables
================
System Variable                                  Value
-----------------------------------------------------
AREA                                             12.56
PI                                                3.14
R                                                    2

node(cfg)#expression R 3
node(cfg)#show variable
System Variables
================
System Variable                                  Value
-----------------------------------------------------
AREA                                             28.26
PI                                                3.14
R                                                    2
```

Extend a system variable by an expression

The name of Trinity system variables is structured in an object-oriented manner (e.g. *sys.up*, *ip.ctx:ROUTER.if:LAN.up*). A variable name consists of several nested objects separated by a dot (.). For example the *sys.up* variable can be regarded as variable up of object *sys*. If there are different instances of an object (e.g. different IP interfaces) the object is identified by its name after a colon (:). Thus the variable *ip.ctx:ROUTER.if:LAN.up* can be regarded as variable up of the object *if* with name *LAN* within the object *ctx* with name *ROUTER* within then object *ip*. This object structure often reflects the structure of the device configuration: "interface LAN" is configured within "context ip ROUTER".

You are allowed to extend such an object by your own custom expressions. For example you may want to define an expression to delay the up-signal of the system.

Example: Add an expression to delay the up-signal of the system by one minute:

```
node(cfg)#expression sys.delayed-up DELAY(sys.up,1m)
```

This creates a new variable *delayed-up* in the object *sys*. If both the expression you create and the variables it refers to operate on the same object (e.g. *sys*) you may want to use the following shortcut notation:

```
node(cfg)#for sys expression delayed-up DELAY(up,1m)
node(cfg)#show variable
System Variables
================
System Variable                                Value
-----------------------------------------------------
sys.delayed-up                                 false
sys.up                                          true

a;sdlfk;lasdkf;laskdf
```

This adds the "delayed-up" variable to the "sys" object and looks for referred-to variable ("up") in the same "sys" object as well. This is summarized by the following procedure.

Mode: Configure

Step	Command	Purpose
1	***node*(cfg)#for *context* expression *name* *expression***	Creates a new variable called *context.name*. The value of the expression is defined by the mathematical *expression*. All variables mentioned in the expression are looked up in the context of the context object. If an expression variable with the same name already exists it is updated with the new mathematical *expression* specified.

Create/Modify an expression family

An expression family is a processing rule that is applied to a set of similar variables.

Example: Let us imaging you want to create an expression that delays the link state of every IP interface by 10 seconds. Instead of writing an expression for each individual IP interface

```
node(cfg)#expression ip.ctx:ROUTER.if:LAN.delayed-up \
                     DELAY(ip.ctx:ROUTER.if:LAN.up)

node(cfg)#expression ip.ctx:ROUTER.if:WAN.delayed-up \
                     DELAY(ip.ctx:ROUTER.if:WAN.up)
node(cfg)#expression ip.ctx:ROUTER.if:DMZ.delayed-up \
                     DELAY(ip.ctx:ROUTER.if:DMZ.up)
```

Trinity allows you to define an expression family for all IP interfaces using the for expression syntax with wildcards:

```
node(cfg)#for ip.ctx:%.if:% expression delayed-up
                                       DELAY(up)
```

The wildcard (%) matches all object names. Thus the string "if:%" matches to all our interfaces "if:LAN", "if:WAN", and "if:DMZ". The command above creates a new variable "delayed-up" in each IP interface object and sets it to the delayed "up" variable of the IP interface as a value.

Let's repeat the syntax for the **for expression** command again with the emphasis that the *context* argument may be an object name containing wildcards or regular expressions.

Mode: Configure

Step	Command	Purpose
1	***node*(cfg)#for *context-family* expression *name expression***	Creates a family of new variable called *name* in any object that matches the *context-family* specification. The value of the expression is defined by the mathematical *expression*. All variables mentioned in the expression are looked up in the context of the corresponding context object. If an expression variable with the same name already exists it is updated with the new mathematical *expression* specified.

Note If the system creates a system variable that matches the *context-family* argument of an existing expression family, that expression family is automatically extended with an expression variable for the new system variable. For example let us assume you configured the expression family *delayed-up* above. If you create a new IP interface called *LAN2*, a new variable called *ip.ctx:ROUTER.if:LAN2.delayed-up* will be created automatically for the new interface.

Wildcards and regular expressions in context-family names

An expression is added for a whole family of variables if the *context-family* argument of the **for expression** command contains one or more wildcards or regular expressions:

- **Wildcard symbol (%)**: The wildcard symbol matches any character up to the next object separator (.) or object-name separator (:). For example the *context-family* string *a.%.c* matches the object names
  - *a.b.c* (% ~= b),
  - *a.x.c* (% ~= x),
  - *a.test.c* (% ~= test),
  - but not match *a.b.b.c* (% !~= b.b)
- **Regular expression ({})**: If you need more complex matching rules you can write a regular expression in curly braces. For example the *context-family* string *a.{test\d+}.c* matches the object names
  - *a.test1.c* (test\d+ ~= test1),
  - *a.test2.c* (test\d+ ~= test),
  - *a.test.c* (test\d+ ~!= test),

The regular expression itself may use the following tokens:

Table 3. Regular Expression Tokens

Symbol	Description	Example
.	Matches any single character.	
[*charset*]	Matches a single character that is contained within the brackets. [abc] matches “a”, “b”, or “c”. [a-z] specifies a range which matches any lowercase letter from “a” to “z”. These forms can be mixed: [abc1-9] matches “a”, “b”, “c”, or any digit. The “-” character is treated as a literal character if it is the last or the first. For example, [abc-] matches “a”, “b”, “c”, or “-”.	LAN[0-9] matches any string starting with the prefix “LAN” followed by a digit.
[^*charset*]	Matches a single character that is not contained within the brackets. [^abc] matches any character other than “a”, “b”, or “c”. [^a-z] matches any single character that is not a lowercase letter from “a” to “z”.	[^-]+ matches any non-empty string without the dash symbol
( )	Groups a subexpression.	(test)+ matches any repetition of the string “test”, e.g. “testtesttest”.
\d	Matches any digit, same as [0-9]	LAN\d+ matches {“LAN1”, “LAN2”, etc.}
\D	Matches any character other than a digit, same as [^0-9].	
\u	Matches any uppercase letter, same as [A-Z].	
\U	Matches any character other than an uppercase letter, same as [^A-Z].	
\l	Matches any lowercase letter, same as [A-Z].	
\l	Matches any character other than a lowercase letter, same as [^A-Z].	
\w	Matches any letter, same as [A-Za-z].	
\W	Matches any character other than a letter, same as [^A-Za-z].	
[:*class:*]	Matches any character from the specified character class. The following character classes exist and are representing the following character group: • alnum[0-9A-Za-z] • alpha[A-Za-z] • blank[ ] • digit[0-9] • lower[a-z] • puncte.g. “.”, “,”, “@”, etc. • upper[A-z] • word[0-9A-Za-z] • xdigit[0-9A-Fa-f]	
?	Matches zero or one occurrences of the preceding element.	test? ~= {“tes”, “test”}

Table 3. Regular Expression Tokens (Continued)

Symbol	Description	Example
%	Matches zero or more occurrences of the preceding element.	test% ~= {“tes”, “test”, “testtttttt”}
+	Matches one or more occurrences of the preceding element.	test+ ~= {“test”, “testt”, “testtttt”}

Example: Create an expression that returns true only if all LAN IP interfaces are up in the default context ROUTER. We assume that all LAN IP interface follow the naming format LAN1, LAN2, etc.

```
node(cfg)#for ip.ctx:ROUTER expression all-if-up \
                                       AND(if:{LAN\d+}.up)
```

Note The curly brackets {} are used to tell the CLI that whatever is between the brackets is a regular expression. The curly brackets themselves are not part of the regular expression. In the example above, the regular expression is *LAN\d+*.

Delete an expression

Use one of the following commands to:

- Delete an expression by its name
- Delete an expression within an object context
- Delete an entire expression family

Mode: Configure

Step	Command	Purpose
1	***node*(cfg)#no expression *name***	Deletes the expression called *name*.
2	***node*(cfg)#no for *context* expression *name***	Deletes the expression *context.name*.
3	***node*(cfg)#no for *context-family* expression *name***	Deletes the expression *name* for the set of all contexts specified by the *context-family*.

Note Trinity does not prevent you from deleting an expression variable that is referred to by another expression. If an expression uses a non-existing variable, its value will be re-computed to an #N/A! error.

Example: Delete the constant PI that is used by the expression AREA.

```
node(cfg)#no expression PI
node(cfg)#show variable
System Variables
================
System Variable                                 Value
-----------------------------------------------------
AREA                                            #N/A!
R
```

Expression Syntax

Trinity expressions are used to combine system variables with a mathematical expression. An expression may contain the following parts:

- Constants (The Boolean values true or false, numbers, time stamps, text strings)
- References (Names of other variables as placeholders for their values)
- Operators (The '+' operator sums two values)
- Functions (The NOW() function returns the current date and time)

Data Types

Constants are parts of the expression that are not calculated. Consider the expression "3.14*R*R" where "3.14" is a floating-point constant. Trinity expressions are type-aware. This means that each constant is assigned to a certain data type. We will discuss those data types and some of its typical constant values below.

Booleans

The Boolean data type only has two values (**true** and **false**). Boolean values represent the truth-values of logic operations (e.g. "true && false" => false). You enter Boolean constants by using one of the following words:

- false
- true

Numbers

Number data types come in two flavors, integers (whole numbers) and floating-point numbers (e.g. 3.14). Trinity automatically takes care of using the adequate flavor for each operation, so you don't have to convert manually between integers and floating-point numbers. You can enter numeric constants with different notations as shown below:

- Integer constants:
  - 123 (decimal notation)
  - 0xff(=255; hexadecimal notation; "0x" prefix)
  - 010 (=8; octal notation; "0" prefix)
- Floating-point constants:
  - 3.14 (floating-point constant)
  - 2.3e-3 (=0.0023; exponent notation)
- Integer and floating-point constants can be used in any order when building expressions:
  - 3.14+2 => 5.14
  - 1.1+2.2 => 3.3
  - 1.1-0.1 => 1

Time Stamps

Trinity distinguishes between absolute time stamps (date/time) and relative time stamps (time difference). Below you find examples how to enter time stamp constants:

- Clock (absolute time stamps): Entered in ISO time notation (yyyy-mm-ddThh:mm:ss)
  - 2016-01-11T12:40:00 (date and time in time zone of the local clock)
  - 2016-01-11 (date only)
  - 12:40:00 (time of day only)
- Time (relative time difference): Entered as a combination of days (d), hours (h), minutes (m), seconds (s), milliseconds (ms), microseconds (us), and nanoseconds (ns).
  - 1d2h (one day plus two hours)
  - 2m30s (two and a half minutes)
  - 1s500ms (one and a half second)
  - In expressions relative time constants can be added to or subtracted from absolute time stamps. The result is a new absolute time stamp:
  - 2016-01-11+1d => 2016-01-12
  - 12:40:00+2m => 12:40:02

Text Strings

Variables can also be used to store text strings. Thus it is important to be able to enter text constants in expression, for example to compare the strings. Text strings must be put in single quotes (') to distinguish them from variable names.

- ‘This is a test’

Strings can be concatenated by the (+) operator and repeated by the (*) operator

- ‘abc’+’def’ => ‘abcdef’
- ‘x’*4 => ‘xxxx’

Errors

Errors cannot be entered as constants but may results as computation value. Table 4 shows all possible computation errors and their meanings:

Table 4. Error Values

Error Value	Description
#DIV/0!	**Division by zero**: Result of an expression that contains a division by zero operation, e.g. “2/(1-1)”.
#N/A!	**Invalid reference**: Result of an expression that refers to a variable that does not exist. Also if your expression uses the result of another expression and you delete that second expression, the first expression will result in this error.
#NUM!	**Invalid number**: An operator or function is expecting a number but finds another data type, e.g. “2+’test’”.

Table 4. Error Values (Continued)

Error Value	Description
#VAL!	**Invalid value**: A function for example expects a positive number but finds a negative number as argument.

Operators

Operators are symbols in your equation that specify the calculation you want to perform on the left-hand side and the right-hand side of the operator. The following tables list all available operators available to Trinity expressions grouped by type.

Logical Operators

Logical operators are used to reason about logical statements.

Table 5. Logical Operators

Operator	Description	Example
!	**Unary negation operator**. Returns true if the value that follows the operator evaluates to false, returns false if the value evaluates to true.	!true => false
|| **or**	**Logical OR operator**. Returns true if either the left-hand side value or the right-hand side value evaluates to true. You can either use the symbolic notation or the keyword "or".	true || false => true true or false => true
^^ **xor**	**Logical XOR operator**. Returns true if either the left-hand side value or the right-hand side value but not both evaluate to true. You can either use the symbolic notation or the keyword "xor".	true ^^ true => false true xor true => false
&& **and**	**Logical AND operator**. Returns true if both the left-hand side value and the right-hand side value evaluate to true. You can either use the symbolic notation or the keyword "and".	true && false => false true and false => false

If the right-hand or the left-hand side value of a logical operator is not of a Boolean data type the value is automatically converted to true or false according to the following rules:

- **Numbers**: Non-zero numbers are evaluated to true, zero is evaluated to false.
- **Absolute Timestamps**: Only the epoch (1970-01-01:00:00:00) is evaluated to false; all other timestamps are valuated to true.
- **Relative Timestamps**: Zero time delta values (e.g. 0s) is evaluated to false, non-zero time delta values (e.g. 500ms) are evaluated to true.
- **Text strings**: An empty string ('') is evaluated to false whereas all non-empty strings (e.g. 'test') are evaluated to true.

Examples:

- 1 && 2 => true
- 0 || ‘test’ => true
- 1m && 0s => false

Bitwise Operators

Bitwise operators operate on number data types only and apply a Boolean operation bit by bit.

Table 6. Bitwise Operators

Operator	Description	Example
~	**Unary bitwise complement operator**. Returns the bit-wise complement of a number, i.e. turns zero-valued bits into one-valued bits and vice-versa	~0xffffffffffffffff => 0
|	**Bitwise OR operator**. Sets each result bit to one if one of the corresponding bits in the left- and right-hand side values is one; sets the result bit to zero otherwise.	0x11 | 0x01 => 0x11
^	**Bitwise XOR operator**. Sets each result bit to one if one but not both of the corresponding bits in the left- and right-hand side values are one; sets the result bit to zero otherwise.	0x11 ^ 0x01 => 0x10
&	**Bitwise AND operator**. Sets each result bit to one if both corresponding bits in the left- and right-hand side values are one; sets the result bit to zero otherwise.	0x11 & 0x01 => 0x01
<<	**Bitwise left shift**: Shifts the number on the right-hand side of the operator left by the number of bits specified with the right-hand side value.	0x01 << 4 => 0x10
>>	**Bitwise left shift**: Shifts the number on the right-hand side of the operator right by the number of bits specified with the right-hand side value.	0x10 >> 4 => 0x01

Arithmetic Operators

Table 7. Arithmetic Operators

Operator	Description	Example
+	**Sum**. Computes the sum of the left-hand side and the right-hand side value. If both values are strings, the sum operator concatenates the strings. If one of the values is an absolute time-stamp and the other is a relative time delta, the result is an absolute timestamp.	1+2 => 3 ‘abc’+’def’ => ‘abcdef’ 2015-01-10+2d => 2015-01-12
-	**Difference**. Computes the difference between the left-hand side and the right-hand side value. If the left-hand side value is an absolute timestamp and the right-hand side value is a relative time delta, the result is an absolute timestamp.	1-2 => -1 2015-01-10-2d => 2015-01-08
*	**Product**. Computes the product of the left-hand side and the right-hand side value. If the left-hand side value is a text string and the right-hand side value is an integer number, the result is the string concatenated n times to itself. If one argument is a relative time delta and the other argument is an integer number, the result is the time delta multiplied n times.	2*3 => 6 ‘x’*3 => ‘xxx’ 30m*2 => 1h
/	**Division**. Computes the division between the left-hand side and the right-hand side value. If the left-hand side value is a relative time delta and the right-hand side is an integer, the result is a relative time delta.	5/2 => 2.5 1d/2 => 12h
%	**Modulo**. Computes the remainder from the integer division of the left-hand side value and the right-hand side value.	13%5 => 3

Comparison Operators

Table 8. Comparison Operators

Operator	Description	Example
==	**Equal**. Returns true if the left-hand side value is equal to the right-hand side value. **Note**: Data types are not converted automatically.	1 == 2 =>false (1+1) == 2 => true 1 == '1' => false

Table 8. Comparison Operators (Continued)

Operator	Description	Example
!=	**Not equal**. Returns true if the left-hand side value is not equal to the right-hand side value. **Note**: Data types are not converted automatically.	1 != 2 => true (1+1) != 2 => false 1 != '1' => false
<	**Less than**. Returns true if the left-hand side value is less than the right-hand side value. If both arguments are strings, the operator performs a lexical comparison. If the arguments are not of the same data type an #VAL! error is returned.	1 < 2 => true 1 < 1 => false 'abc' < 'def' => true 1 < '1' => #VAL!
>	**Greater than**. Returns true if the left-hand side value is greater than the right-hand side value. If both arguments are strings, the operator performs a lexical comparison. If the arguments are not of the same data type an #VAL! error is returned.	1 > 2 => false 1 > 1 => false 'abc' > 'def' => false 1 > '1' => #VAL!
<=	**Less than or equal**. Returns true if the left-hand side value is less than or equal the right-hand side value. If both arguments are strings, the operator performs a lexical comparison. If the arguments are not of the same data type an #VAL! error is returned.	1 <= 2 => true 1 <= 1 => true 'abc' <= 'def' => true 1 <= '1' => #VAL!
>=	**Greater than or equal**. Returns true if the left-hand side value is greater or equal than the right-hand side value. If both arguments are strings, the operator performs a lexical comparison. If the arguments are not of the same data type an #VAL! error is returned.	1 >= 2 => false 1 >= 1 => true 'abc' >= 'def' => false 1 > '1' => #VAL!

Operator Precedence

The order in which Trinity evaluates operators is well defined according to the table below. For example, the products operator (*) is evaluated before the sum operator (+):

2+2*2 => 6

You can change this order by using parentheses:

(2+2)*2 => 8

If the equation contains operators with the same precedence they are evaluated from left to right:

2+2-2 => (2+2)-2 => 3

The following table shows the order in which operators are evaluated. The operators in the top row are evaluated first, those in the bottom row at the end.

Table 9. Operator Precedence

Operator	Description
! **not** **~** **-**	Unary operators (Boolean negation, bitwise complement, arithmetic negation)
***** **/** **%**	Multiplication, division, and modulo operators
<< **>>**	Bit shift operators
< **>** **<=** **>=**	Comparison operators except equal and not equal
== **!=**	Equal and not-equal comparison operators
&	Bitwise AND operator
^	Bitwise XOR operator
|	Bitwise OR operator
&& **and**	Logical AND operator
^^ **xor**	Logical XOR operator
|| **or**	Logical OR operator

Functions

Functions are predefined equations that perform calculations on their arguments. Functions start with the upper-case function name, followed by an open parenthesis, an list of comma-separated arguments, and a closing parenthesis, e.g.

- DAY(2015-01-12) => 12
- BOOL(123) => true
- POWER(2,10) => 1024
- SQRT(9) => 3
- COUNT(1,2,3,4,5) => 5

Functions can be nested. That is instead of specifying a argument with a constant or variable name you can use another function:

- DAY(NOW()) => 11
- BOOL(SQRT(9)) => true
- COUNT(NOW(),SQRT(9)) => 2

The following tables lists all available functions available to Trinity expressions grouped by type.

Logical Functions

All logical operators (not, or, xor, and) are also available as functions. So instead of writing the expression "a && b" you may write the expression "AND(a,b)". The advantage of the function is that it may take more than two arguments, e.g. "AND(a,b,c,d"), which evaluates to true only if all arguments are true.

All logical functions take one or more arguments (except NOT(), which takes exactly one), convert all arguments to a Boolean value and returns the logical operation on all the arguments. If no argument is provided an invalid-value error (#VAL!) is returned.

- AND(1,2,3) => true
- AND(1,2,3,0) => false
- AND() => #VAL!

If an argument specifies a set of variables by using a regular expression, the regular expression is expanded and all matching variables are passed to the function. For example if there are three variables "test:1", "test:2", "test:3" the expression "AND(test:%)" is expanded to "AND(test:1,test:2,test:3).

Table 10. Logical Functions

Function	Description	Example
NOT(*boolean*)	**Negation**. Returns true if the boolean argument evaluates to false, returns false if the argument evaluates to true.	NOT(false) => true NOT(true) => false
OR(*boolean1*[,*boolean2*...])	**Logical OR operator**. Returns true if at least one of the arguments evaluate to true.	OR(1,2,3) => true OR(1,2,0) => false OR() => #VAL
XOR(*boolean1*[,*boolean2*...])	**Logical XOR operator**. Returns true if an odd number of arguments evaluate to true.	XOR(1,2,3) => true XOR(1,2,0) => false XOR() => #VAL
AND(*boolean1*[,*boolean2*...])	**Logical AND operator**. Returns true if both all arguments evaluate to true.	AND(1,2,3) => true AND(1,2,0) => false AND() => #VAL
BOOL(*arg*)	Converts the argument to a Boolean value. Converts error to the value false. Use this function if you don't want errors in used variables to be passed as errors to the result of the expression.	BOOL(1/0) => false BOOL('test') => true BOOL('') => false

Bitwise Functions

The bitwise operators (~, |, ^, &) are also available as functions. So instead of writing the expression "a & b" you may write the expression "BITAND(a,b)". The advantage of the function is that it may take more than two arguments, e.g. "BITAND(a,b,c,d)", sets a bit in the result to true only if the same bit is true in all arguments.

If an argument specifies a set of variables by using a regular expression, the regular expression is expanded and all matching variables are passed to the function. For example if there are three variables "test:1", "test:2", "test:3" the expression "BITAND(test:%)" is expanded to "BITAND(test:1,test:2,test:3).

Table 11. Bitwise Functions

Function	Description	Example
BITNOT(*number*)	**Bitwise complement**. Returns the bit-wise complement of a number, i.e. turns zero-valued bits into one-valued bits and vice-versa	BITNOT(0xffffffffffffffff) => 0
BITOR(*number* [,*number*…])	**Bitwise OR**. Sets each result bit to one if one of the corresponding bits in any of the arguments is one; sets the result bit to zero otherwise.	BITOR(0x1,0x2) => 0x3 BITOR() => #VAL
BITXOR(*number* [,*number*…])	**Bitwise XOR**. Sets each result bit to one if the corresponding bits in an odd number of arguments are one; sets the result bit to zero otherwise.	BITXOR(0x1,0x2) => 0x3 BITXOR() => #VAL
BITAND(*number* [,*number*…])	**Bitwise AND**. Sets each result bit to one if the corresponding bits in all arguments is one; sets the result bit to zero otherwise.	BITAND(0x1,0x2) => 0x0 BITAND() => #VAL
BITLSHIFT(*number*, *shift*)	**Bitwise left shift**: Shifts the number argument by shift bits to the left.	BITLSHIFT(0x1,4) => 0x10
BITRSHIFT(*number*, *shift*)	**Bitwise right shift**: Shifts the number argument by shift bits to the right.	BITLSHIFT(0x10,4) => 0x1

Arithmetic Functions

Table 12. Arithmetic Functions

Function	Description	Example
ABS(*number*)	Returns the absolute value of a *number*	ABS(123) => 123 ABS(-3.14) => 3.14
CEILING(*number* [,*significance*])	Rounds a *number* up to the nearest integer or to the nearest multiple of *significance*. The default *significance* is 1.	CEILING(3.14,0.1) => 3.2
DIFF(*minuend*, *subtrahend*)	Subtracts the *subtrahend* from the *minuend* (same as the '-' operator). If the minuend is an absolute timestamp and the subtrahend is a relative time delta, the result is an absolute timestamp.	DIFF(1,2) => -1 DIFF(2015-01-10,2d) => 2015-01-08

Table 12. Arithmetic Functions (Continued)

Function	Description	Example
DIV(*dividend, divisor*)	Divides the *dividend* by the *divisor* (same as the '/' operator). If the *divisor* evaluates to zero the function returns a #DIV/0! error. If the *dividend* is a relative time delta and the divisor is an integer, the result is a relative time delta.	DIV(5,2) => 2.5 DIV(1/0) => #DIV/0! DIV(1d,2) => 12h
FLOOR(*number [,significance]*)	Rounds a *number* down to the nearest integer or to the multiple of *significance*. The default *significance* is 1.	FLOOR(3.14,0.1) => 3.1
INT(*number*)	Rounds a *number* down to the nearest integer.	INT(3.14) => 3
MOD(*dividend, divisor*)	Computes the remainder from the integer division of the *dividend* and the *divisor.* If the *divisor* evaluates to zero the function returns a #DIV/0! error.	MOD(13,5) => 3
MROUND(*number [,multiple]*)	Returns the *number* rounded to the desired *multiple*. The default *multiple* is 1.	MROUND(3.25,0.1) => 3.3
NEG(*number*)	Returns the negative *number.*	NEG(2+2) => -4
POWER(*number, power*)	Raises the *number* to the specified *power.*	POW(3,2) => 9
PRODUCT(*factor [,factor…]*)	Computes the product of all factors. If the first argument is a text string and the second a positive integer this function returns the text concatenated n times to itself.	PRODUCT(1,2,3,4,5) => 120 PRODUCT('x',3) => 'xxx' PRODUCT() => #VAL!
QUOTIENT(*dividend, divisor*)	Returns the integral part of the division of the *dividend* and the *divisor.* If the *divisor* evaluates to zero the function returns a #DIV/0! error.	QUOTIENT(9,2) => 4

MROUND(*number [,digits]*)	Returns a *number* argument rounded to the desired fractional *digits*; the default number of *digits* is 0	MROUND(3.25,1)
		=> 3.3
		MROUND(123,-2)
		=> 100

SQRT(*number*)	Computes the square root of the *number* argument.	SQRT(16) => 4

Table 12. Arithmetic Functions (Continued)

Function	Description	Example
SUM(*summand* [,*summand*...])	Computes the sum over all *number* arguments. If all arguments are text strings the function concatenates the strings.	SUM(1,2,3,4,5) =>15 SUM('a','b','c') => 'abc' SUM() => #VAL!

Comparison Functions

Table 13. Comparison Functions

Function	Description	Example
EQ(*arg1, arg2*)	Returns true if the two arguments are equal (equal type and equal value) (same as the '==' operator).	EQ(1,2) => false EQ(1d,60m) => true
NEQ(*arg1, arg2*)	Returns true if the two arguments are not equal (unequal type or unequal value) (same as the '!=' operator).	NEQ(1,2,) => true NEQ(1d,60m) => false
IF(*condition, true-expr, false-expr*)	Returns the *true-expr* argument if the condition evaluates to true, else returns the *false-expr* parameter.	DIFF(1,2) => -1 DIFF(2015-01-10,2d) => 2015-01-08
ISERR(*arg*)	Returns true if the argument *arg* is an error.	ISERR(1/1) => false ISERR(1/0) => true
LE(*arg1, arg2*)	Less than or equal. Returns true if *arg1* is less than or equal *arg2* (same as the '<=' operator). If both arguments are strings, the function performs a lexical comparison. If the arguments are not of the same data type an #VAL! error is returned.	LE(1,2) => true LE(1,1) => true LE('abc','def') => true LE(1,'1') => #VAL!
LT(*arg1, arg2*)	Less than. Returns true if *arg1* is less than *arg2* (same as the '<' operator). If both arguments are strings, the function performs a lexical comparison. If the arguments are not of the same data type an #VAL! error is returned.	LT(1,2) => true LT(1,1) => false LT('abc','def') => true LT(1,'1') => #VAL!
GE(*arg1, arg2*)	Greater than or equal. Returns true if *arg1* is greater or equal than *arg2* (same as the '>=' operator). If both arguments are strings, the function performs a lexical comparison. If the arguments are not of the same data type an #VAL! error is returned.	GE(1,2) => false GE(1,1) => true GE('abc','def') => false GE(1,'1') => #VAL!

Table 13. Comparison Functions (Continued)

Function	Description	Example
GT(*arg1, arg2*)	Greater than. Returns true if *arg1* is greater than *arg2* (same as the '>' operator). If both arguments are strings, the function performs a lexical comparison. If the arguments are not of the same data type an #VAL! error is returned.	GT(1,2) => false GT(1,1) => false GT('abc','def') => false GT(1,'1') => #VAL!

Set Functions

These functions return information about the number of arguments but don't look at the arguments themselves.

Table 14. Set Functions

Function	Description	Example
COUNT([*arg])**	Counts how many values are in the list of arguments	COUNT(1,false,4,1,0) => 5 COUNT(ip.ctx:ROUTER.if:%.up) => counts the number of IP interfaces in context ROUTER
COUNTIF([*arg])**	Counts how many values are in the list of arguments that evaluate to true	COUNTIF(1,false,4,1,0) => 3 COUNTIF(ip.ctx:ROUTER.if:%.up) => counts the number of IP interfaces in context ROUTER that are up

Time/Date Functions

Table 15. Time/Date Functions

Function	Description	Example
DATE(*year,* *month,* *day*)	Builds a date value from the specified *year, month*, and *day* number arguments. This function is most useful in situations where the year, month, and day are supplied by formulas or references.	DATE(2015,12,11) => 2015-12-11
DATETIME(*date,* *time*)	Builds a date/time value from the specified *date* and *time* arguments. This function is most useful in situations where the date and time values are supplied by formulas or references.	DATETIME(TODAY(), 12:00:00) => 2016-01-11T12:00:00
DAY(*date-time*)	Returns the days part of a date/time value	DAY(2015-12-13T10:20:30) => 13
EPOCH()	Returns the earliest date/time value possible.	EPOCH() => 1970-01-01T00:00:00

Table 15. Time/Date Functions (Continued)

Function	Description	Example
HOUR(*date-time*)	Returns the hours part of a date/time value	DAY(2015-12-13T10:20:30) => 10
MINUTE(*date-time*)	Returns the minutes part of a date/time value	DAY(2015-12-13T10:20:30) => 20
MONTH(*date-time*)	Returns the months part of a date/time value	DAY(2015-12-13T10:20:30) => 12
NOW()	Returns the current date/time in the local timezone; note that if NOW() is used in an expression, that expression is only updated every 10 seconds.	NOW() => 2016-01-11T19:05:00
SECOND(*date-time*)	Returns the seconds part of a date/time value	DAY(2015-12-13T10:20:30) => 30
TIME(*hour,* *minute,* *second*)	Builds a time value from the specified *hour, minute*, and *second* number arguments. This function is most useful in situations where the *hour, minute*, and *second* are supplied by formulas or references.	TIME(10,11,12) => 10:11:12
TODAY()	Returns the current date in the local time-zone; note that if TODAY() is used in an expression, that expression is only updated every 10 seconds.	TODAY() => 2016-01-11
YEAR(*date-time*)	Returns the year part of a date/time value	DAY(2015-12-13T10:20:30) => 2015

Temporal Functions

The following functions allows to reason about the temporal relation of events. Temporal functions also provide means of temporally manipulate event occurrences, for example delaying events.

Table 16. Temporal Functions

Function	Description
DEBOUNCE(*arg, period*)	De-bounces the argument *arg* by the specified time period, i.e. filters out frequent changes of *arg*. If *arg* changes, the change is not passed to the output immediately but is rather delayed by the specified period. If *arg* changes back to the original value within the period (unlike the DELAY function) the spurious value is never passed to the output. Figure 14 on page 138 shows a time diagram of an input argument x, de-bounced by 500ms.

Table 16. Temporal Functions

Function	Description
DEBOUNCEDEC(*arg, period*)	Only de-bounces the argument *arg* by the specified time period if *arg* is **decreased**, i.e., if the new value is lower (e.g. true -> false). If *arg* is **increased** (e.g. false -> true) the value of *arg* is always passed to the output immediately. Figure 14 on page 138 shows a time diagram of an input argument x and its decrease de-bounced by 500ms. This function is useful to trigger a link-down event only after a certain time and only if the link does not come up again in this period: DEBOUNCEDEC(ip.ctx:ROUTER.if:LAN, 1m)
DEBOUNCEINC(*arg, period*)	Only de-bounces the argument *arg* by the specified time period if *arg* is **increased**, i.e., if the new value is higher (e.g. false -> true). If *arg* is **decreased** (e.g. true -> false) the value of arg is always passed to the output immediately. Figure 14 on page 138 shows a time diagram of an input argument x and its increase de-bounced by 500ms. This function is useful to trigger a link-up event only after a certain time and only if the link does not go down again in this period: DEBOUNCEINC(ip.ctx:ROUTER.if:LAN, 1m)
DEBOUNCEINC(*arg, period, condition*)	Only de-bounces the argument *arg* by the specified time period if the condition evaluates to true. While the condition evaluates to false the value of *arg* is always passed to the output immediately. Figure 14 on page 138 shows a time diagram of an input argument x, a condition c, and its output de-bounced by 500ms.
DELAY(*arg, period*)	Delays the argument *arg* by the specified time period. Figure 15 on page 138 shows a time diagram of an input argument x delayed by 500ms.
ONCE(*arg*)	Changes to true once when the argument *arg* evaluates to true, but never changes back to false until the device is rebooted. Figure 15 on page 138 shows a time diagram of an input argument x passed though the ONCE function. This function is useful to make sure a certain action is only executed once after boot-up, for example after the device synchronized its clock over NTP: ONCE(ntp.sync)
STABLE(*arg, period*)	Returns false as soon as *arg* changes. Switches to true only when the argument *arg* is stable for the specified *period* of time. Figure 15 on page 138 shows a time diagram of the STABLE function applied to an input argument x by 500ms.

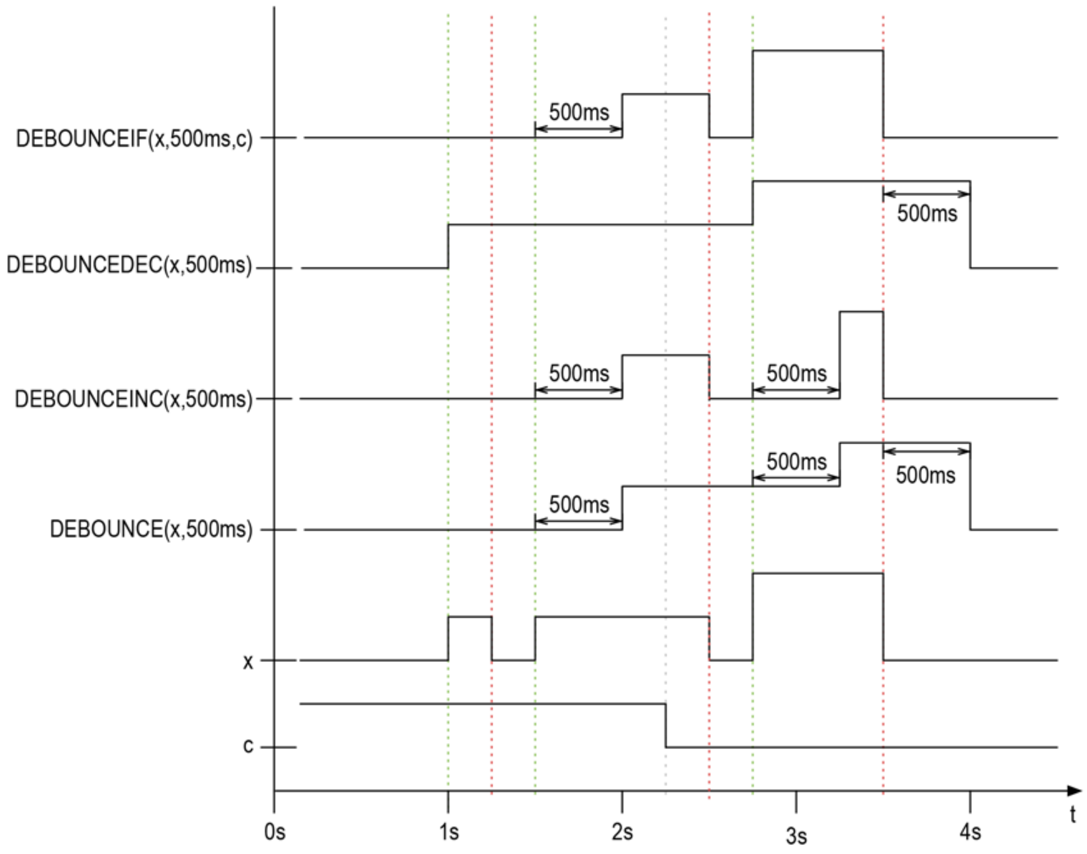


Figure 14. Time Diagram of the DEBOUNCE Functions

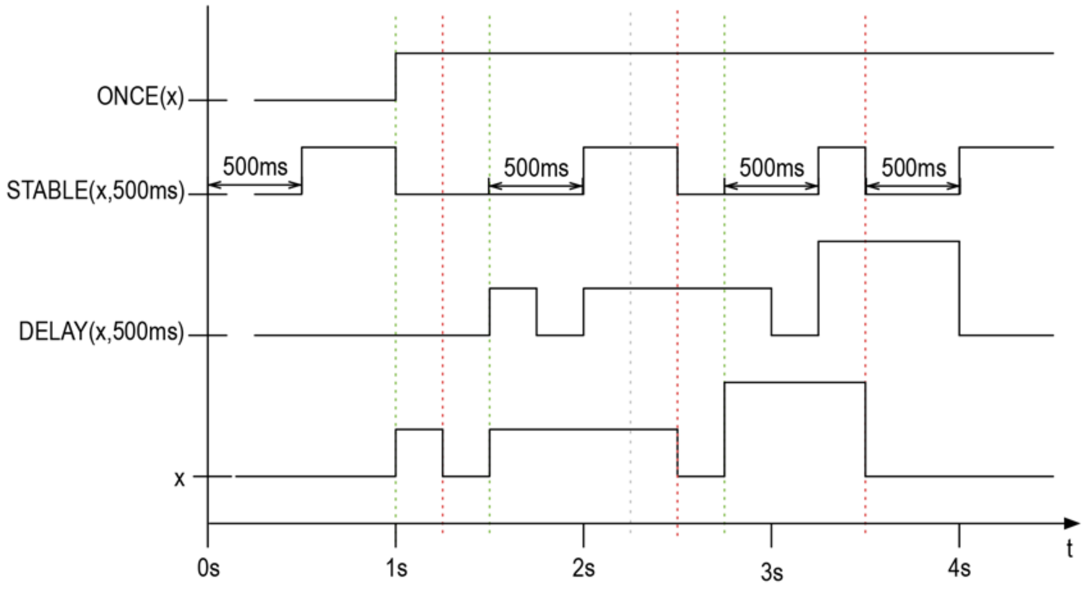


Figure 15. Time Diagram of the Functions ONCE, STABLE, and DELAY

Example Expressions

The following section lists some expression example that may serve as hints of how to use the expression algebra to build powerful temporal expressions.

Example: **De-Bouncing link down-up transitions**: If the DHCP address of the LAN interface comes up this expression only changes to true after 10 seconds. If the DHCP address goes down, this change is propagated immediately to the created variable *ip.ctx:ROUTER.if:LAN.addr:DHCP.UP-STABLE*.

```
node(cfg)#for ip.ctx:ROUTER.if:LAN.addr:DHCP \
          expression UP-STABLE \
                     DEBOUNCEINC(up, 10s)
```

Example: Combining link states of all addresses of an interface: The following expression creates a new variable *ip.ctx:ROUTER.if:LAN.ALL-ADDR-UP*, which only evaluates to true if the link state of all addresses of the LAN interface is up.

```
node(cfg)#for ip.ctx:ROUTER.if:LAN \
          expression ALL-ADDR-UP \
                     AND(addr:%.up)
```

Example: Wait until the system is up and ready for communication: The following expression creates a new variable *READY*, which only evaluates to true communication with the device can start. This is the case if the system reports up and if the WAN link is up for a minute and if NTP time synchronization has been taken place.

```
node(cfg)#expression READY \
                     "sys.up \
                      && DEBOUNCEINC(ip.ctx:ROUTER.if:WAN.up, 1m) \
                      && ntp.sync"
```

Example: Recognize longer down-cycles: The following expression creates a new variable *ip.ctx:ROUTER.if:LAN.LONG-DOWN* that changes to true if the link state of the LAN interface is down for more than ten minutes.

```
node(cfg)#for ip.ctx:ROUTER.if:LAN \
          expression LONG-DOWN \
                     "!up && STABLE(up, 10m)"
```

The STABLE(up, 10m) term switches to true if the *ip.ctx:ROUTER.if:LAN.up* variable stays at the same value (true or false) for ten minutes. This has to be combined with the term !up such that the resulting variable only evaluates to true if the link is down for that period of time.

State Profiles

State profiles build simple state machines from system variables and expressions. These state machines are used by other applications such as the SIP user agent. SIP drops incoming packets when the system is overloaded (see Chapter 47, "SIP Overload Configuration" on page 529). The overload situation is detected by the OVERLOAD state profile, which is fully programmable by the user.

Consider the example depicted in figure 16 on page 140:

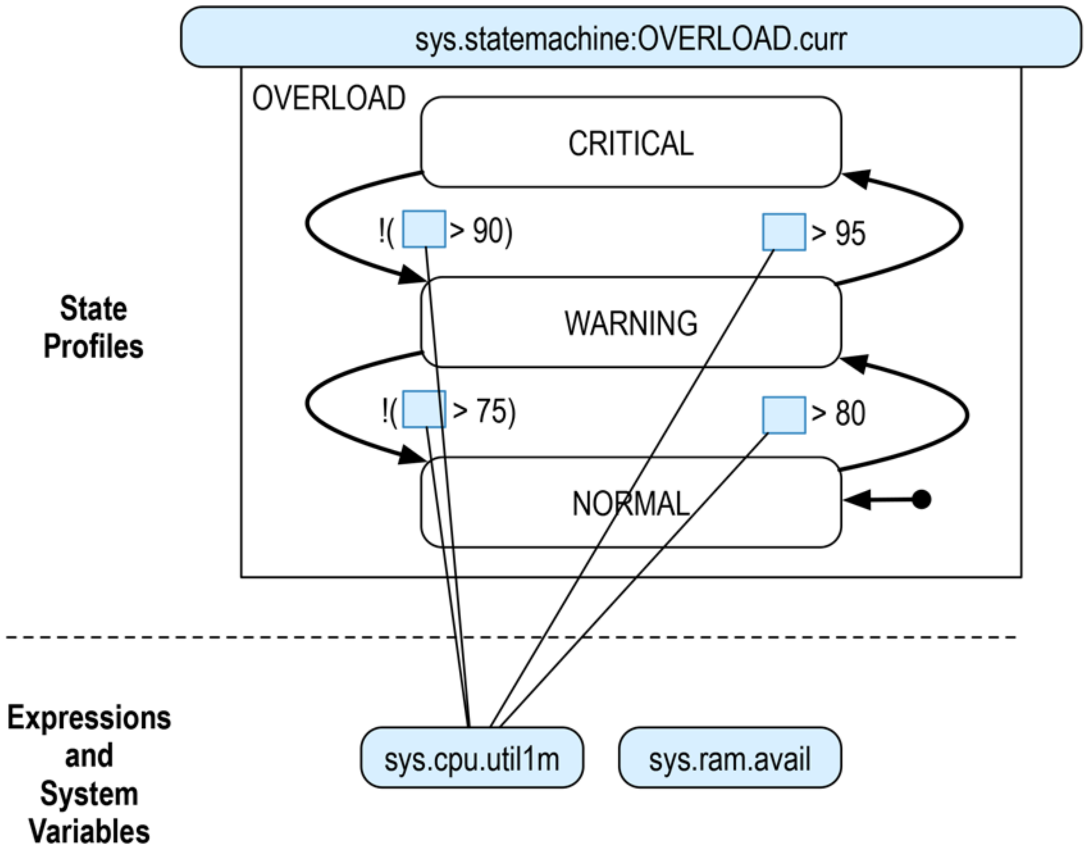


Figure 16. State Profile

Next to the initialization state *NORMAL* the state profile *OVERLOAD* defines two additional states, *WARNING* and *CRITICAL*. The transitions between those states are specified by expressions using the system variable sys.cpu.util1m, which exposes the CPU utilization, averaged over one minute. If the CPU utilization raises above 80% the *OVERLOAD* state machine changes to the *WARNING* state, if it crosses the 95% margin the *CRITICAL* state is entered. You can also define hold conditions to build a hysteresis as depicted in figure 17 (green=NORMAL, yellow=WARNING, red=CRITICAL).

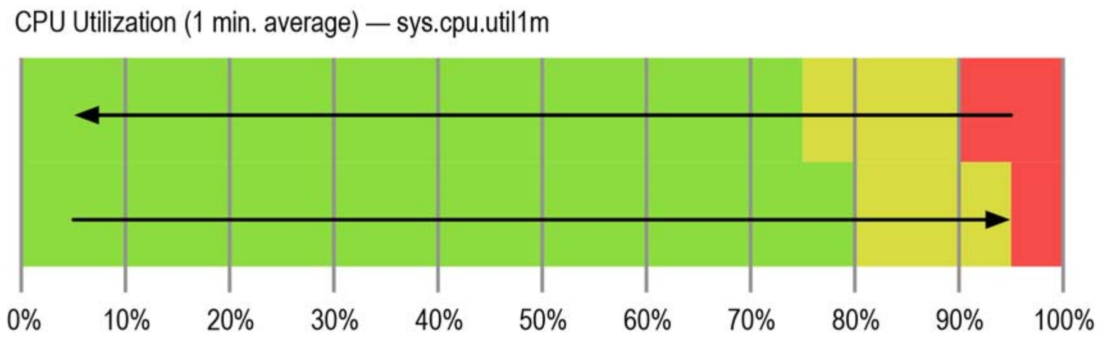


Figure 17. State Transition Hysteresis

In the example above the state profile only changes back from the CRITICAL to the WARNING state if the CPU utilization drops to 90% or below. This example is configured by the CLI commands below:

```
profile state OVERLOAD
  init-state NORMAL

  state 1 WARNING
    enter-if sys.cpu.util1m>80 hold-when sys.cpc.util1m>75

  state 2 CRITICAL
    enter-if sys.cpu.util1m>95 hold-when sys.cpu.util1m>90
```

A state profile exposes its state as another system variable. For example, the *OVERLOAD* profile defines the variable *sys.statemachine:OVERLOAD.curr*, a string variable that is either set to 'NORMAL', 'WARNING', or 'CRITICAL', dependent on the current state of the state machine. Other components may use this variable to react on state changes.'

Default OVERLOAD state profile

The device automatically creates a default OVERLOAD state profile, populated with reasonable expressions to switch the device into WARNING or CRITICAL state. Other applications such as the SIP user agent uses this OVERLOAD profile to limit VoIP calls if the system resources are in a critical state. The default CLI configuration for the OVERLOAD state profile is printed in the code snippet above

The default OVERLOAD state profile transitions to state

- **WARNING**: if the CPU utilization (1 min. av.) rises above 80%.
- **CRITICAL**: if the CPU utilization (1min. av.) rises above 95%.

The device should not reach any of those thresholds if it is operated under nominal load as described in the product manual. However, if the device is under a denial-of-service attack, using the OVERLOAD state profile to define the SIP overload behavior makes sure the ongoing calls are not affected by an excessive load (see Chapter 11, "Programmable System-Event Configuration" on page 111).

Create and configure a new state profile

The following procedure shows how to create a state profile similar to the default OVERLOAD profile shown in the example above. The procedure illustrates how to create a new profile, configure the name of the initialization state, create a new state, and add enter and hold conditions for that state. You may repeat steps 3 and/or 4 to create multiple states or to add multiple conditions to a state.

Mode: Configure

Step	Command	Purpose
1	***node*(cfg)#profile state *name***	Creates a new state profile called name. Use the same name when linking another component to the profile with the **use profile state** command.
2	***node*(pf-state)[*name*]#init-state *init-state***	Sets the profile's initialization state name. The state machine is in this state if none the entering-conditions of any other state evaluates to true. If you don't enter this command the default init state will be called **INIT**.

Step	Command	Purpose
3	***node*(pf-state)[*name*]#state *state***	Appends a new state called *state* in the state profile *name*. As long as no entering conditions are configured for the state the state machine will never enter the state.
4	***node*(state)[*name*]#enter-if *enter-expression* [hold-when *hold-expression*]**	Configures an *enter-expression* that if it evaluates to true will cause the state machine to enter the current state. To build a hysteresis, use the optional *hold-expression*, which defines how long the state machine should stay in this state. If absent the *hold-expression* will be set to the *enter-expression*.

Note States within a state profile are ordered by an ordinal number, e.g. **state 1 WARNING**, **state 2 CRITICAL**. This order is important. The state machine tries to enter the state with the highest ordinal number the enter-expression of which evaluates to true. Consider again Figure 5: If the CPU utilization changes to 98% the enter expression of both states, WARNING and CRITICAL are true. Since CRITICAL is a higher-order state than WARNING the state machine immediately enters the CRITICAL state.

Check the configuration of state profiles

Use the following show command to either get a list of all configured state profiles or to print detailed information about one particular profile.

Mode: Operator exec

Step	Command	Purpose
1	***node*>show profile state [*name*]**	If a profile *name* is specified this command prints detailed information about the state profiles. If the profile *name* is skipped, a list of all profiles is printed.

Example: Print a list of all state profiles:

```
node>show profile state
State Profiles
==============

State Profile  Current State
------------------------------
LED-STATE      UP
NTP-STATE      INIT
OVERLOAD       WARNING
```

Example: Print detailed information about the OVERLOAD state profile:

```
State Profile: OVERLOAD
=======================

  Init State:           NORMAL
  Other States:         CRITICAL, WARNING
  Current State:        NORMAL (init state)
  System-Variable Name: sys.statemachine:OVERLOAD.curr
```

Debug state transitions

Use the following debug command to observe state transitions:

Mode: Operator exec

Step	Command	Purpose
1	***node*>debug state full-detail**	Enables the real-time debugger that prints information about state transitions of state profiles to the current terminal.
2	***node*>debug variable detail 1**	Additionally observes changes of system variables, on which the state-transition expressions are built.

Example: Observe all state-machine transitions and changes of all system variables:

```
node>debug state full-detail
node>debug variable detail 1
15:01:23.000  VAR      # [sys.cpu.util1m] 94 -> 98
15:01:23.000  STATE    # [OVERLOAD] Re-evaluating state-machine...
15:01:23.000  STATE    # [OVERLOAD]   Check state 1 WARNING:
15:01:23.000  STATE    # [OVERLOAD]     Enter condition: true
15:01:23.000  STATE    # [OVERLOAD]     Hold condition:  true
15:01:23.000  STATE    # [OVERLOAD]   Check state 2 CRITICAL:
15:01:23.000  STATE    # [OVERLOAD]     Enter condition: false
15:01:23.000  STATE    # [OVERLOAD]     Hold condition:  false
15:01:23.000  STATE    # [OVERLOAD] Transition to state WARNING
```

Example: To test whether the system behaves as expected in a state that is rarely entered you may temporarily add a new expression to that state that always evaluates to true.

```
node>debug state full-detail
node>debug variable detail 1
node>enable
node#configure
node(cfg)#profile state OVERLOAD
node(pf-state)[OVERLOAD]#state WARNING
node(state)[WARNING]#enter-if true
15:01:23.000  STATE    # [OVERLOAD.WARNING.true] onNew
15:01:23.000  STATE    # [OVERLOAD.WARNING] Update expression...
15:01:23.000  STATE    # [OVERLOAD.WARNING]   Enter expression:
  BOOL(sys.cpu.util1m >= 80) || BOOL(true)
15:01:23.000  STATE    # [OVERLOAD.WARNING]   Hold expression:
  BOOL(sys.cpu.util1m >= 75) || BOOL(true)
15:01:23.000  STATE    # [OVERLOAD] Re-evaluating state-machine...
15:01:23.000  STATE    # [OVERLOAD]   Check state 1 WARNING:
15:01:23.000  STATE    # [OVERLOAD]     Enter condition: true
15:01:23.000  STATE    # [OVERLOAD]     Hold condition:  true
15:01:23.000  STATE    # [OVERLOAD]   Check state 2 CRITICAL:
15:01:23.000  STATE    # [OVERLOAD]     Enter condition: false
15:01:23.000  STATE    # [OVERLOAD]     Hold condition:  false
15:01:23.000  STATE    # [OVERLOAD] Transition to state WARNING
```

Chapter 12 Alarm Management

Chapter contents

Introduction

This management component is responsible for managing the alarm sub-system. It allows you to add/remove and see alarms. Trinity is equipped with alarm sub-system. Any component may register alarms and once triggered they will show up in the alarms table. Each alarms can have one of the following severities:

- **Informative:** General informative alarm
- **Minor:** Minor warning like alarm. The system is about to overheat
- **Major:** System is above the allowed heat threshold
- **Critical:** System had a catastrophic failure

Alarms are predefined and are configured by the trinity. You cannot add alarms dynamically. However you can change the severities and you can enable and disable them.

Alarm Configuration Task List

The following sections describe how to configure/use the Alarm component:

- Changing alarm options

Viewing alarms

Mode: administrative access

Step	Command	Purpose
1	[*device*]**#show alarms**	Lists alarms and their configurations

Example output:

```
    00A0BA0~(cfg)#show alarms

#      Description                      Severity        Count   Alarm Status    Fault Status    Auto Clear

0      CPU Over Threshold               Major           0                                       no
1      DSL 0/0                          Major           0                                       no
2      DSL 0/1                          Major           0                                       no
3      DSL 0/2                          Major           0                                       no
4      DSL 0/3                          Major           0                                       no
5      Hardware Sensor Failure          Major           0                                       no
6      Temperature Over Threshold       Major           0                                       no
```

Fields:

- **Count** – Alarm counter. Shows how many time this specific alarm has been triggered.
- **Fault status** – If true, signifies that there is a fault present. System is malfunctioning
- **Alarm status** – If true, signifies that the alarm is active regardless of the fault status. You can clear the alarm if the fault status is no longer present.
- **Auto Clear** – a flag that tell the system to clear the alarm once the fault condition has gone.

Changing alarm options

Mode: administrative access

Step	Command	Purpose
1	[*device*]**#alarm** <*name*> **clear**	This command enables and disables the clear-all flag of the specified alarm.
2	[*device*]**#alarm clear-all**	This command enables and disables the clear-all flag of all alarms present.

Chapter 13 Auto Provisioning of Firmware and Configuration

Chapter contents

Introduction

This chapter provides an overview of Trinity's Auto Provisioning capabilities and tasks involved to configure it. The auto provisioning capability enables you to automatically distribute up-to-date configurations and firmware to a large number of units using TFTP, HTTP, or HTTPS. It works as follows: The unit downloads a specific file from a TFTP server. If this file has changed since the last download, it is stored and executed. If the file on the server did not change since the last download, no action is taken. If the units are configured to do auto provisioning, a network operator can only update the firmware files on the server, which automatically distributes it to all units. The "profile provisioning" configures this.

Provisioning Profile

Creation

A provisioning profile is a set of commands dedicated for keeping updated a specific file or software. For example one profile can keep the firmware updated and another profile can keep the configuration up to date.
Mode: Configure

Step	Command	Purpose
1	***node*(cfg)# [no] profile provisioning** <name>	Creates or enters a new provisioning profile with the according name and enters it for editing.

Destination

The destination specifies the type of file which is keeping up to date. There are three possible destinations available:

- **script:** Used for firmware update and firmware update on daughter boards.
- **configuration:** Used for updating the startup configuration.
- **upload:** Uploads the current configuration back to the server.
- **firmware:** Used for updating the firmware
- **wizard:** Used for provisioning wizard xml files

Mode: Profile provisioning

Step	Command	Purpose
2	**node(pf-prov)**[name]**# destination (script | configuration | upload | firmware | wizard)**	Specifies the type of provisioning

Destination Script

The destination script is used to provision firmware, firmware for daughter boards or updating other specific files on the device, for example WEB pages. Multiple locations can be used and locations for scripts can use the following protocols: TFTP, HTTP and HTTPS. Together with HTTP or HTTPS it is possible to make user authentication. Using HTTPS it is possible to make server authentication with a certificate. The target file of a location can be either a **.tar** file containing a **manifest** file. Or it can be a manifest file directly.

In the case of a **.tar** file the provisioning needs to download the complete tar file each time the provisioning triggers. For firmware updates this can generate a lot of traffic. Due to when the manifest in the tar is detected being the same file as the last updated, the provisioning does not perform an update. The advantage of using a **.tar** directly is that the delivered files can be used as received without additional manipulations.

In the case of a **manifest** the provisioning downloads only the manifest file each time the provisioning is triggered. If then a change compared to the last downloaded manifest is detected all other files belonging to the delivery are downloaded. Therefore these files must be stored exactly at the same place as the manifest file on the server.
In either case the activation reload immediate should be configured to bring new software into actions as soon as possible.

An example configuration for provisioning software from a .tar file:

```
profile provisioning FIRMWARE
  destination script
  use profile tls DEFAULT
  location 1 tftp://firmware.patton.com/devices/SN5300.tar
  location 2 tftp://192.168.2.2/devices/SN5300.tar
  activation reload immediate
```

An example configuration for provisioning software from a manifest file:

```
profile provisioning FIRMWARE
   destination script
   use profile tls DEFAULT
   location 1 tftp://firmware.patton.com/devices/SN5300/manifest
   location 2 tftp://192.168.2.2/devices/ SN5300/manifest
   activation reload immediate
```

Destination Configuration

The destination configuration is used to provision the startup configuration of the device. Multiple locations can be used and locations for configuration can use the following protocols: TFTP, HTTP and HTTPS. Together with HTTP or HTTPS it is possible to make user authentication. Together with HTTPS it is possible to make server authentication with a certificate.

The activation reload immediate should be configured to bring new configuration into action as soon as possible.

Destination Upload

The destination upload is used to upload the startup configuration from the device up to a server. Only one location can be used and only with the protocol TFTP. There should be no activation configured because there is no update on the device itself.

- **Destination Firmware:** The destination firmware is used to provision newer firmware to the device
- **Destination Wizard:** The destination wizard allows for provisioning of wizard xml files

Locations

Under locations the protocol, server and the file on that server is specified. Some of the features of provisioning are only available for certain protocols. The available protocols are:

- **TFTP:** Standard download protocol without security features, can be used for uploads too.
- **HTTP:** Web download with the possibility of requesting digest authentication from the device. Not available for uploads.
- **HTTPS:** Secure web download with the possibility of requesting digest authentication from the device and the possibility of authenticate the certificate of the webserver. Not available for uploads.

At least one location has to be configured for a profile. But it is possible to configure multiple locations as fallback if the first locations cannot be reached. Important is that on all the locations in the same profile the exact same file has to be reached. Uupdating or replacing of these files, should be done simultaneously.

Mode: Profile provisioning

Step	Command	Purpose
3	***node*(pf-prov)**[name]**# location [<index>] tftp://**server/path/file	Adds a location with tftp protocol at the index or at the end
3	***node*(pf-prov)**[name]**# location [<index>] http://**server/path/file	Adds a location with http protocol at the index or at the end
3	***node*(pf-prov)**[name]**# location [<index>] https://**server/path/file	Adds a location with https protocol at the index or at the end
3	***node*(pf-prov)**[name]**# location [<index>] $(dhcp.66)**	Adds a location with tftp server from the DHCP option 66
3	***node*(pf-prov)[name]# location [<index>] $(dhcp.66)**<suffix>	Adds a location with tftp server from the DHCP option 66
3	***node*(pf-prov)[name]# no location <index>**	Removes the location at the specified index

Placeholders in Locations

In the server, path and filename of the location placeholders can be used. These placeholders are replaced by their corresponding value and can be used to build the provisioning server location request. The following placeholders are available:

- **$(cli.major)**— CLI major version as numerical value
- **$(cli.minor)**— CLI minor version as numerical value
- **$(system.hw.major)**— Hardware major version as numerical value
- **$(system.hw.minor)**— Hardware minor version as numerical value
- **$(system.sw.major)**— Software major version as numerical value
- **$(system.sw.minor)**— Software minor version as numerical value
- **$(system.sw.build)**— Software build version as numerical value
- **$(system.sw.date)**— Software release date as string in the format ‘YYYY-MM-DD’

- **$(system.product.name)**— Product name as string
- **$(system.mac)**—MAC address of ETH 0/0 as string in the format 'AABBCCDDEEFF' (without ":" between the hexadecimal characters)
- **$(system.serial)**—serial number of the unit as string in the format 'AABBCCDDEEFF'
- **$(dhcp.66)**—TFTP server provided from DHCP. The following fields in the DHCP offer are considered in descending priority:
  - **option 66:** TFTP server IP, as string delivered by the DHCP server.
  - **siaddr:** BOOTP next server, converted to string delivered by the DHCP server.
  - **sname:** BOOTP server name, as string delivered by the DHCP server.
  - When no DHCP is enabled or none of the above fields is present, the string is empty.
- **$(dhcp.67)**—TFTP file name provided form DHCP. The following fields in the DHCP offer are considered in descending priority:
  - **option 67:** TFTP file name, as string delivered by the DHCP server.
  - **file:** BOOTP file name, as string delivered by the DHCP server.
  - When no DHCP is enabled or none of the above fields is present, the string is empty.

Conditional Placeholders in Locations

A conditional placeholder may also be used to choose between two values and or placeholders in the function of a condition. It has the following syntax:

- **$(number_1=number_2|true_placeholder|false_placeholder)**
- **$(number_1>number_2|true_placeholder|false_placeholder)**
- **$(number_1>=number_2|true_placeholder|false_placeholder)**
- **$(number_1<number_2|true_placeholder|false_placeholder)**
- **$(number_1<=number_2|true_placeholder|false_placeholder)**

Explanation of elements:

- **number_1 and number_2:** Values used to determine if the condition is true of false. Can be a placeholder corresponding to a numerical value or any numerical value.
- **true_placeholder:** String to replace the condition if the condition is true. This can be also a nested placeholder or nested conditional placeholder.
- **false_placeholder:** String to replace the condition if the condition is false. This can be also a nested placeholder or nested conditional placeholder. This can be left out and if the condition is false then the condition is replaced by an empty string

Here are some examples for locations with conditional placeholders:

- tftp://192.168.1.1/software/$($(system.hw.major)>=2|new|old)/image.tar
- tftp://192.168.1.1/software/$($(system.sw.major)=3|$(system.product.name)|image).tar
- tftp://192.168.1.1/software/$($(cli.major)>=4|/trinity)/image.tar

Activation

For the destination script and configuration it is preferred to let the device reboot after the update in order for the update to become active. The following modes are available:

- **activation reload immediate:** After an update of the firmware or the configuration a reload happens immediately to bring the updated files into action. In most cases this should be used.
- **activation reload deferred:** The reload is not triggered automatically and happens later only when executing the command "reload if-needed".
- **no activation:** No reload is executed after the provisioning. This should be used only for the upload destination.
- **graceful**: Activates the provisioned files or firmware with a reload as soon as there are no ongoing calls on the device any more.

Mode: Profile provisioning

Step	Command	Purpose
4	***node***(**pf-prov)**[name]**# [no] activation reload (immediate | deferred | graceful)**	Specifies if and when an activation with a reload has to take place, to bring the updated files into action.

User authentication

For configuration and firmware provisioning through HTTP or HTTPS optional authentication is available. It is required if the final server where the configured items have to be downloaded is configured for basic or digest authentication. This has no impact when the TFTP protocol is used.

Mode: Profile provisioning

Step	Command	Purpose
5	***node***(**pf-prov)**[name]**# [no] authentication [realm** <realm>**] username** <user> **(no-password | password** <password> **)**	Adds/Removes authentication credentials for the configured realm. If no realm is given the configured credentials act as wild-card. The wild-card is considered if the realm provided in 401 Unauthorized doesn't match any explicit configured realm.

Server authentication

For HTTPS locations, server authentication can be enabled. To do so, a TLS profile has to be bound to the provisioning profile. Be aware that in the case of HTTPS only a subset of the TLS configuration options are used. Here a list of the actual TLS parameters used in the case of HTTPS:

- authentication outgoing
- trusted-certificate

The used TLS profile can be configured as indicated below.

Mode: Profile provisioning

Step	Command	Purpose
6	**node(pf-tls)**[name]**# [no] use profile tls** <tls-profile-name>	Chooses the TLS settings to use for HTTPS locations

TLS Profile commands used from provisioning

To enable the server authentication the following command in the TLS profile can be used. Enabling server authentication makes use of the configured trusted certificates from the TLS profile to verify the trustworthy of the server certificate. If the server authentication is disabled the server certificate is accepted always.
Mode: Profile TLS

Step	Command	Purpose
7	**node(pf-tls)[name]# [no] authentication outgoing**	Enables or disables server authentications with HTTPS provisioning

If server authentication is enabled the trusted-certificate command in the TLS profiles defines which are the root certificates we trust, and from one of these the server certificate must be signed off.
There are following variants to configure:

- **built-in-defaults:** Use all certificates which are built in provided from the software as trusted. This group of certificates is dependent of the software which is running on the device. With each software update this whole list is updated too. New certificates may be added. Certificates are removed only if they expire or needs to be considered as not trustworthy anymore. The user cannot make any changes to this group.
- **user-stored:** Use all certificates which are installed from the user. It is possible to install certificates from tftp or from the built-in-defaults certificate group.
- **all-stored:** Use all built-in-defaults certificates and all user-stored certificates as trusted certificates.
- **specific list:** The user can create the list of trusted certificates on his own. Therefore any certificate of the group built-in-defaults or user-stored can be used. All certificates which are not listed are not considered as trusted certificates.

Mode: Profile tls

Step	Command	Purpose
8	**node(pf-tls)**[name]**# trusted-certificate (all-stored | built-indefaults | user-stored)**	Configures to use all imported trusted certificates for the current TLS profile. This is the default setting.

Mode: Profile tls

Step	Command	Purpose
8	**node(pf-tls)**[name]**# [no] trustedcertificate pki:trustedcertificate/** certificate	Adds or removes a certificate to or from the set of trusted certificates of the current PKI profile. Removing the last link sets the TLS-profile configuration back to the default setting, which is to use all trusted certificates in PKI.

Mode: Profile tls

Step	Command	Purpose
8	**node(pf-tls)**[name]**# [no] trustedcertificate pki:trusted-certificatedefaults/** certificate	Adds or removes a certificate to or from the set of trusted certificates of the current PKI profile. Removing the last link sets the TLS-profile configuration back to the default setting, which is to use all trusted certificates in PKI.

PKI commands used from provisioning

Under PKI the group of built-in-defaults trusted certificates is used. This group is managed from Patton and can be changed only through a software update. This group of certificates is dependent of the software which is running on the device. With each software update this whole list is updated too. New certificates may be added. Certificates are removed only if they expire or need to be considered as not trustworthy anymore. The user cannot make any changes to this group. But it is possible to copy certificates from this group to another group or to user-stored certificates or to a tftp server for the usage on other devices.
Mode: configure

Step	Command	Purpose
1	**node# copy pki:/trustedcertificates- defaults/**<certificate> **pki:/trusted-sertificates/**<certificate>	Copy a certificate from the built-in-defaults group to the user-stored group to be used there

Mode: Configure

Step	Command	Purpose
1	**node# copy pki:/trustedcertificates-defaults/**<certificate> **tftp://**<server/path/certificate>	Copy a certificate from the built-in-defaults group to a tftp server to be installed on other devices

Using Provisioning

To trigger a provisioning profile the execute command has to be used.
Mode: configure

Step	Command	Purpose
1	**node# provisioning execute** <name>	Executes the provisioning profile immediately one time

To continuously poll for firmware or configuration changes, use the provisioning execute command together with the timer command. Here's how to do both firmware and configuration provisioning, with a polling interval of one day:

```
    timer FIRMWARE_UPDATE now + 2 minutes every day “provisioning execute FIRMWARE”
    timer CONFIG_UPDATE now + 2 minutes every day “provisioning execute CONFIG”
```

Provisioning Reset

When a device executes a provisioning of a script successfully, it stores the information about executing that script permanently on the device. This is to avoid repeated execution of the same script. Now, if the user manually performs software updates or switching of images and expects the provisioning to execute the software update script after that, this is never going to happen because of the stored state of the script. Therefore a command is introduced to manually reset the stored provisioning state and let the provisioning execute the next time it is triggered.

Mode: Enable

Step	Command	Purpose
1	**node#provisioning reset [<profile-Name>]**	Resets the stored provisioning state for all the provision-ing profiles or only the one specified with *profileName*.

Provisioning Status

To show the status of provisioning events, the following can be done to view this information.

Mode: Operator

Step	Command	Purpose
1	**node>show provisioning status [continuously]**	Shows current and pending execution of provisioning pro-file

Chapter 14 Ethernet Port Configuration

Chapter contents

Introduction

This chapter provides an overview of Ethernet ports and describes the tasks involved in configuring Ethernet ports through the Trinity operating system.

Ethernet Port Configuration Task List

To configure Ethernet ports, perform the tasks described in the following sections. Most of the tasks are required to have an operable Ethernet port, some of the tasks are optional, but might be required for your application.

- Entering the Ethernet port configuration mode (see page 157)
- Configuring medium for an Ethernet port (see page 157)
- Configuring Ethernet encapsulation type for an Ethernet port (see page 158)
- Binding an Ethernet port to an IP interface (see page 158)
- Configuring multiple IP addresses on the Ethernet ports (see page 159)
- Configuring a VLAN (see page 160)
- Configuring layer 2 CoS to service-class mapping for an Ethernet port (advanced) (see page 161)
- Closing an Ethernet port (see page 162)

Entering the Ethernet Port Configuration Mode

To enter port configuration mode and begin configuring an Ethernet port, enter the command **port ethernet** `slotport`in administrator execution mode. The keywords *slot* and *port* represent the number of the respective physical entity.

Configuring Medium for an Ethernet Port

All Ethernet ports are configured by default to auto-sense both the port speed and the duplex mode. This is the recommended configuration. Command options are (if supported by the platform):

- **auto**—auto-negotiate the port speed and duplex, advertising all supported speeds and duplexes.
- **negotiated**—auto-negotiate the port speed and duplex, only advertising the specified speed and duplex
- **manual**—disable auto-negotiation and force the port speed and duplex. The remote port must also be configured to force speed and duplex. Otherwise, the remote port will operate in half-duplex, resulting in collisions.
- **10**—for 10 Mbps
- **100**—for 100 Mbps
- **1000**—for Gigabit Ethernet
- **half**—for half-duplex
- **full**—for full-duplex
- **sfp**—for ports that support both copper and SFP modules, force the port to use the SFP module

This procedure describes how to configure the medium for the Ethernet port on *slot* and *port*

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#port ethernet** *slot port*	Enters the Ethernet port configuration mode for the physical Ethernet connector on *slot* and *port*.
2	***device*(prt-eth)[*slot/port*]#medium { auto | negotiated** *speed duplex* **| manual** *speed duplex* **[sfp] }**	Configures the Ethernet port's medium.

Example: Configuring medium for an Ethernet port

The following example shows how to configure medium auto-sense for the Ethernet port on slot 0 and port 0.

```
device(cfg)#port ethernet 0 0
device(prt-eth)[0/0]#medium auto
```

Binding an Ethernet Port

You must bind the Ethernet port to an existing bridge-group, switch-group, and/or interface. When executing the **bind** command, the requested bridge-group, switch-group, and/or interface must exist (see Chapter 22, “IP Interface Configuration” on page 238 to learn how to create a new IP). If no IP context is given, the system looks for the interface in the default IP context known as *ROUTER*.

Figure 18 shows the logical binding of the Ethernet port at slot *0* on port *0* to the IP interface *LAN,* which is defined in the default IP context *ROUTER*.

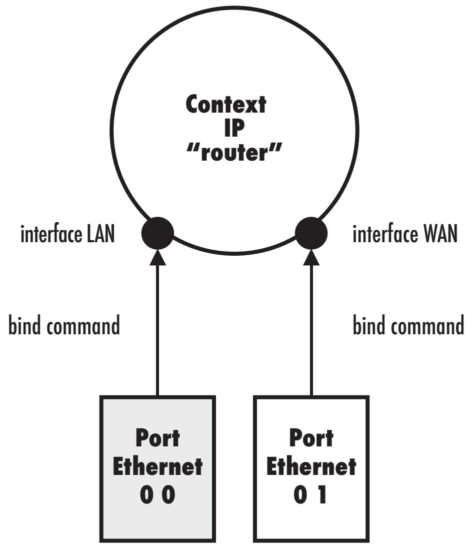


Figure 18. Binding of an Ethernet port to an IP interface

This following procedure describes how to bind the Ethernet port to an already existing IP interface

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#port ethernet** *slot port*	Enters Ethernet port configuration mode for the physical Ethernet connector on *slot* and *port*
2	***node*(prt-eth)[*0/0*]#bind interface [<ctxName>]** <if*Name*>	Binds the Ethernet port to the logical IP interface ifname in IP context ctxName

The <ifName> parameter specifies the name of the IP interface within the IP context given with the <ctxName> parameter.

Note Note: The <ctxName> parameter is optional. If you don't provide it, Trinity binds to an IP interface in the default ROUTER context.

Example: Create an IP interface with one static address and bind it to port ethernet.

```
node>enable
node#configure
node(cfg)#context ip ROUTER
node(ctx-ip)[ROUTER]#ip interface LAN
node(ip-if)[ROUTER.LAN]#ipaddress STATIC 10.1.1.1/24
node(ip-if)[ROUTER.LAN)#port ethernet 0 0
node(prt-eth)[0/0]#bind interface ROUTER LAN
node(prt-eth)[0/0]#no shutdown
```

Multiple IP Addresses on Ethernet Ports

It is possible to use multiple IP addresses on an Ethernet port by creating multiple IP addresses on the logical (bound) IP interface.

Note In the previous software releases, multiple IP addresses on an Ethernet port could be achieved by binding the same Ethernet port to multiple IP interfaces. This is no longer supported. Instead, you should now create multiple IP addresses on the same IP interface and bind the Ethernet port to that interface.

The procedures below demonstrate how two different IP addresses (potentially in the same network) can be used on an Ethernet port. However, if necessary any number of static IP addresses and an optional DHCP address can be created on an IP interface.

Mode: Configure

Step	Command	Purpose
1	***node*(cfg)#context ip** [ROUTER]	Enter the IP context mode for the default virtual router.
2	***device*(ctx-ip)[ROUTER]#interface** *<name>*	Creates a new IP interface and/or enters the configuration mode on an existing IP interface.
3	***device*(ip-if)[ROUTER.***<name>***]#ipaddress** *[<label>] <address>*	Creates a new IP address and network mask on the IP interface <name> or changes the IP address and network mask of an existing IP address, identified by its <label>.

The <label> parameter specifies the name of the address. An IP interface may host more than one IP address. The label must be unique within the IP interface; you may re-use the same label in a different IP interface.

Note Note: The label parameter is optional. If you don't provide it, Trinity creates a new IP address for the interface and automatically assigns it a unique label.

Note Note: If you specify DHCP as label, you create a DHCP address. Only one DHCP address can be created per interface.

The <address> parameter specifies the network address and mask size in dotted-decimal format a.b.c.d/m for IPv4 or in the colon format a:b:c::x/m for IPv6. Alternatively, you may enter the network address and full network mask with two consecutive parameters, a.b.c.d e.f.g.h or a:b:c::x e:f:g::y, respectively.

Example: Create an IP interface with one static an a DHCP address and bind it to port ethernet.

```
node>enable
node#configure
node(cfg)#context ip ROUTER
node(ctx-ip)[ROUTER]#ip interface LAN
node(ip-if)[ROUTER.LAN]#ipaddress STATIC 10.1.1.1/24
node(ip-if)[ROUTER.LAN]#ipaddress DHCP
node(ip-if)[ROUTER.LAN)#port ethernet 0 0
node(prt-eth)[0/0]#bind interface LAN
```

Configuring a VLAN

By default, no VLAN ports are configured on an Ethernet port. One or more VLAN ports can be created on each Ethernet port.

You must bind each VLAN port to an existing IP interface which must be distinct from the IP interface bound by the Ethernet port and by other VLANs. When executing the **bind** command, the requested interface must exist (see Chapter 22, "IP Interface Configuration" on page 238 to learn how to create a new IP interface).

For incoming VLAN packets, each of the 8 possible layer-2 class of services (CoS) can be mapped to an internal traffic-class. Unless otherwise configured, all CoS values map to the default traffic-class.

By default all VLAN ports are initially disabled. They can be enabled with the **no shutdown** command. The corresponding Ethernet port must also be enabled for the VLAN port to work. If the Ethernet port is disabled, all associated VLAN ports are also disabled.

When a VLAN port is closed, the IP interface that is bound to this port is also closed. All static routing entries that are using this interface change their state to *invalid* and all dynamic routing entries will be removed from the route table manager.

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#port ethernet** *slot port*	Enter Ethernet port configuration mode.
2	***device*(prt-eth)[*slot*/*port*]#vlan** *id*	Create new VLAN port.
3	***device*(vlan)[eth-*slot*/*port*.*id*]#bind interface [ROUTER]** *name*	Bind the VLAN port to the existing interface *name*. If no IP context is given, the system looks for the interface in the default IP context known as *ROUTER*.
4	***device*(vlan)[eth-*slot*/*port*.*id*]#no shutdown**	Activate the VLAN port.
5	***device*(vlan)[eth-*slot*/*port*.*id*]#exit**	Returns to the Ethernet port configuration mode
6	***device*(prt-eth)[*slot*/*port*]#no shutdown**	Make sure the hosting Ethernet port is also activated.

Configuring Layer-2-CoS to Service-class Mapping for a VLAN

To enable to transport real-time and delay-sensitive services such as VoIP traffic across the network, Trinity supports the delivery of Quality of Service (QoS) information in the CoS (Class of Service) field of the 802.1pQ header (VLAN header). This is a three-bit field (values 0 to 7). Internally we use traffic-class tags to mark packets belonging to a certain class of service (see Chapter 22, "IP Interface Configuration" on page 238 for more detail on the concept of traffic-classes).

To define the Class of Service (CoS) to traffic-class mapping and vice-versa, the **map** command in the VLAN configuration mode is used. The following procedure describes how to change layer-2-CoS to traffic-class and the traffic-class to layer-2-CoS mapping.

Mode: vlan

Step	Command	Purpose
1	***device*(vlan)[eth-*slot*/*port*.*id*]#map cos *cos to* traffic-class** *traffic-class*	Applies for all inbound packets with the layer-2-CoS field set to cos, which sets the traffic-class tag for those packets to traffic-class. New traffic-classes are created on the fly.
2	***device*(vlan)[eth-*slot*/*port*.*id*]#map traffic-class** *traffic-class* **to cos** *cos*	Applies for all outbound packets with the internal traffic-class tag set to traffic-class, which sets the layer-2-CoS field of those packets to cos.

Example: Adding a mapping-table entry

The following example shows how to add a mapping table entry, which converts a layer 2 class of service value of 2 into a traffic-class tag *VOICE* and vice-versa on VLAN 100 on the Ethernet port on slot *0* and port *1*.

```
device>enable
device#configure
device(cfg)#port ethernet 0 1
device(prt-eth)[0/1]#vlan 100
device(vlan)[eth-0/1.100]#map cos 2 to traffic-class VOICE
device(vlan)[eth-0/1.100]#map traffic-class VOICE to cos 2
```

Closing an Ethernet Port

An Ethernet port can be closed with the **shutdown** command. This command also disables and closes the IP interface that is bound to that port. All static routing entries that are using this interface change their state to 'invalid' and all dynamic routing entries will be removed from the route table manager.

This procedure describes how to disable the Ethernet port on *slot* and *port*.

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#port ethernet** *slot port*	Enters the Ethernet port configuration mode for the physical Ethernet connector on *slot* and *port.*
2	***device*(prt-eth)[*slot*/*port*]#shutdown**	Disables Ethernet port on *slot* and *port.*

The **no** prefix before the **shutdown** command causes the port to open with the interface to which it is bound.

Example: Disabling an Ethernet port

The following example shows how to disable the Ethernet port on slot *0* and port *1*.

```
device(cfg)#port ethernet 0 1
device(prt-eth)[0/1]#shutdown
```

Checking the state of the Ethernet port on slot 0 and port 1 shows that the interface was closed.

```
device(prt-eth)[0/1]#show port ethernet 0 1

Ethernet 0 1
-------------------------------------

Medium : Auto
Administrative State : Down
Bound IP Interface : ROUTER.WAN
IP Addresses : WAN (static): down
 Configured: 20.1.1.1/24
Operational: 20.1.1.1/24
DHCP (DHCP): down
Requested: (none)
Operational: (none)
Medium : Not Connected
Operational State : Down
Hardware Address : 00:a0:ba:06:d6:0a
MTU : 1500
```

```
ARP : enabled
Multicast: enabled
Rx Statistics : 0 bytes in 0 packets
0 errors 0 drops, 0 overruns
0 multicast packets
Tx Statistics : 2038 bytes in 13 packets
0 errors 0 drops, 0 collisions 0 carrier errors
```

Moreover the IP interface, which is bound to the Ethernet port on slot 0 and port 0 gets also closed. Checking the state of the IP interface *WAN* indicates this with the Status parameter set to shutdown on the interface and down on all its addresses, respectively.

```
device(prt-eth)[0/1]#show ip interface

IP Interface Status Address Type Status Configured Operational
-------------------------------------------------------------
WAN shutdown WAN static down 20.1.1.1/24 20.1.1.1/24
DHCP DHCP down (none) (none)
```

Chapter 15 Cellular Modem

Chapter contents

Introduction

This chapter describes how to configure a cellular modem.

System variables

Support for 4G is provided for modems using the Qualcomm chipset. Multiple vendors support this technology such as ZTE, Novatel Wireless, Sierra Wireless and others. Future support for 3G technologies is planned.

About Cellular Modem

The cellular modem consists of two distinct parts, a virtual and physical entity.

The first allows a creation of a device using the CLI. This is used to describe the type of physical device that is needed in order to complete the connection and provide a path to the provider's network.

The latter is simply a physical device that allows a connection to a provider. It consists of a vendor and product identification and a SIM card that is used to authenticate and provide access. The SIM card contains information about the provider and country of origin that is part of the IMEI.

Note You need to provision the modem correctly on a computer before using it on Trinity.

Configuring a Virtual Port

This procedure describes how to create a virtual port.

Step	Command	Purpose
1	***node*(cfg)#[*no*] port virtual cell-modem *port***	Creates/removes the virtual port profile *port* and enters/exits its configuration mode.
2	***node*(vprt-cell)[*port*]#link usb vendor 0x**value **product 0x**value **priority value**	Links a virtual device to a specified physical device. Valid 16-bit hexadecimal values are accepted for the vendor and product. Inclusive values from 1 to 2^{32}-1 are valid for priority.
3	***node*(vprt-cell)[*port*]#*apn*** *apn*	Provide the access point name to use.
4	***node*(vprt-cell)[**port**]#bind interface** *interface interface*	Binds a specified interface to a named IP interface.
5	***node*(vprt-cell)[*port*]#[no] shutdown**	Enables/disables the device to connect to the provider's network.

Example: Creating a virtual port and assigning it to a physical device.

```
node(cfg)#port virtual cell?]modem 1
node(vprt?]cell)[1]#link usb vendor 0x19d2 product 0x0157 priority 300
node(vprt?]cell)[1]#apn fast.t?]mobile.com
node(vprt?]cell)[1]#bind interface ROUTER WAN
node(vprt?]cell)[1]#no shutdown
```

In order to complete the procedure, an interface is required. This tells Trinity that the provider will issue the details for the IP address, gateway and mask.

Step	Command	Purpose
1	***node*(cfg)#context ip** *name*	Creates a new interface name that represents an IP interface.
2	**node(ctx-ip)[name]#interface** *interface*	Assigns a name to the context IP interface.
3	***node*(if-ip)[***name.interface***]#ipaddress** *interface* ***cellmodem* [route-metric** *<value>***]**	Provide the access point name to use.

Example 1: Create an IP interface for a virtual port with a metric value of 20.

```
node(cfg)#context ip ROUTER
node(ctx?]ip)[ROUTER]#interface WAN
node(if?]ip)[ROUTER.WAN]#ipaddress WAN dhcp route?]metric 20
```

Example 2: Create an IP interface for a LAN and virtual port.

```
node(cfg)#context ip ROUTER
node(ctx?]ip)[ROUTER]#interface LAN
node(if?]ip)[ROUTER.WAN]#ipaddress LAN dhcp
node(cfg)#context ip ROUTER
node(ctx?]ip)[ROUTER]#interface WAN
node(if?]ip)[ROUTER.WAN]#ipaddress WAN cellmodem route?]metric 10
```

The **route-metric** adds the ability to assign a metric or hop count cost. The lower metric takes precedence since it is a lower *cost* to route.

In Example 2, the LAN will be the default route with a metric of 0. When the LAN goes down then the WAN will become the default route with a metric of 10. The LAN will once again become the default route once it returns.

Configuring a Physical Device

There is no configuring a physical device. All of its information will be provided by querying the physical device. If however a virtual device does not exist or no match found when a physical device is plugged in, Trinity will create a virtual device consisting of the port and link commands. Trinity will examine the running configuration for any virtual device. If any virtual device is found, but not matching the physical device then the largest virtual port value found plus one will be used for the new virtual device. The USB hotplug will provide the vendor and product identification and the lowest priority will be assigned.

Step	Command	Purpose
1	***node*(cfg)#port virtual cellmodem** *port*	Creates the virtual port profile port+1 where port is the largest value found from the running configuration.
2	***node*(vprt-cell)[***port***]#link usb vendor 0x***value* **product 0x***value* **priority 1**	Links a virtual device to a specified physical device and assigns a priority of 1, the lowest possible priority.

In order to complete the connection to the provider, the virtual port generated will need to bind and have the 'no shutdown' issued as stated in the virtual port section of this document. If no access point name is provided then Trinity will try to find one using the data provided by querying the IMEI. The IMEI consists of a country of origin and provider number that is used to search through the known provider list found in Trinity. It is recommended that an APN always be entered as per the information you got from the provider.

Chapter 16 Hardware Switching

Chapter contents

Introduction

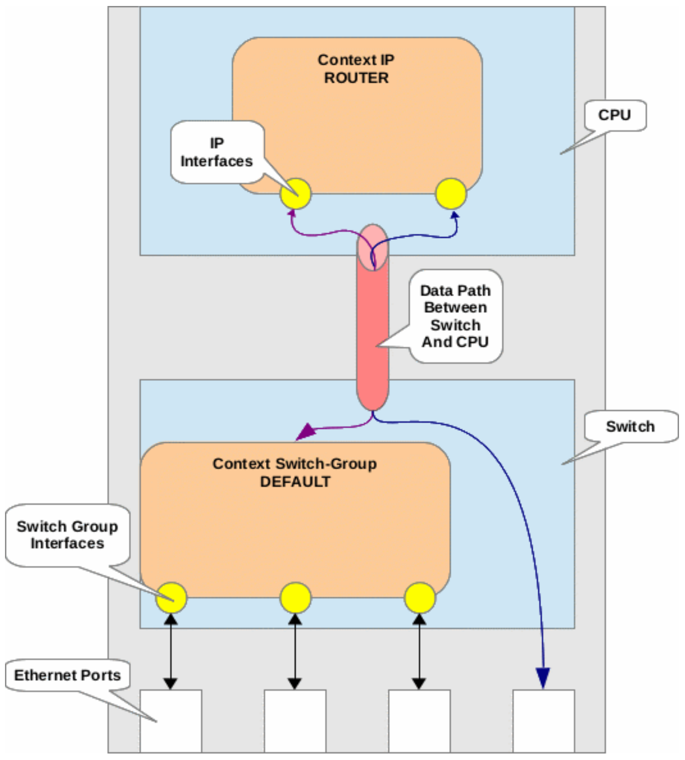
This chapter discusses hardware switching. The following are covered:

- Switch Groups
- VLAN (802.1p/Q)
- Access Control List Configuration
- QoS Traffic Scheduler
- ToS Stripping and Prioritization
- MAC Filter Configuration
- Trunk Configuration
- Ethernet Service Policy Configuration

Switch Groups

This device has an Ethernet switch inside to which external ports are connected. Thus, Ethernet switching between these ports is possible without using the main CPU. This frees the CPU to be used for other tasks, such as IP routing or voice applications.

However, the ports must be configured to use the switch by binding them to switch group interfaces. Otherwise, the switch will forward all packets they receive to the CPU. Consider the diagram below:



- The IP context resides on the CPU indicating that IP routing and termination is performed by the device's main CPU.
- The switch group context resides on the switch, indicating that Ethernet switching is performed by the switch without using CPU resources.
- The first three Ethernet ports are bound to interfaces in the switch group context, indicating that Ethernet traffic is switched between the first three ports, but not the fourth.
- The switch group context is bound to an IP interface, indicating that traffic received by the first three Ethernet ports can be routed out the fourth Ethernet interface. Also, a PC attached to one of the first three Ethernet ports can be used to manage the device.
- Even though the fourth Ethernet port is not bound to a switch group interface, its traffic still goes through the switch before reaching the CPU.
- All ports attached to the switch share a single data path to the CPU. This means that even though traffic can be switched between ports at line rate, routing between ports connected to the switch is limited by the speed of the data path between the switch and CPU.

This chapter describes the tasks involved in configuring a switch group to perform Ethernet switching between the device's ports.

Switch Group Configuration Task List

To configure a switch group, perform the tasks described in the following sections.

Create a switch group

By default, this device has one switch group, `DEFAULT`. Unless you need more than one switch group, use `DEFAULT` rather than create another.

Mode: Configure

Step	Command	Purpose
1	**device(cfg)#context switch-group** *ctx-name*	Create switch group *ctx-name* if it doesn't yet exist and enter “context switch group” mode.

Bind switch group to an IP interface

This step is optional. It is only needed if you want this device to serve as the gateway for this network or if you want this device to be managed from this network.

Mode: Context switch group

Step	Command	Purpose
1	**device(ctx-swgrp)[***ctx-name***]#bind interface [***ip-ctx-name***]** *ip-if-name*	Bind this switch group to IP interface *ip-if-name* in IP context *ip-ctx-name*.

Create switch group interfaces

Perform these steps for each of the ports you want in this switch group.

Mode: Context switch group

Step	Command	Purpose
1	**device(ctx-swgrp)[***ctx-name***]#interface** *if-name*	Create interface *if-name* in switch group *ctx-name* if it doesn't yet exist and enter "interface switch group configuration" mode.

Bind ports to switch group interface

Perform these steps for each of the ports you want in this switch group.

Mode: Configure

Step	Command	Purpose
1	**device(cfg)#port** *type slot port*	Enter port configuration mode
2	**device(prt-***type***)[***slot***/***port***]#bind switch-group** *ctx-name if-name*	Bind this port to interface *if-name* in switch group *ctx-name*. It will switch traffic with all other ports bound to interfaces in switch group *ctx-name*.

Examples

LAN/WAN Configuration

The following example shows how to configure `port ethernet 0 0` as a WAN port and the remaining Ethernet ports as LAN ports. The switch will process all traffic from one device on the LAN destined to another device on the LAN. The CPU will only process traffic that must be routed between the LAN and the WAN.

Notice that `port ethernet 0 0` binds directly to an IP interface, whereas the other Ethernet ports bind to switch group interfaces. The switch group then binds to an IP interface.

```
device(cfg)#context ip ROUTER
device(ctx-ip)#interface LAN
device(if-ip)[LAN]#ipaddress 192.168.1.1/24
device(if-ip)[LAN]#exit
device(ctx-ip)#interface WAN
device(if-ip)[WAN]#ipaddress 10.0.0.1/24
device(if-ip)[WAN]#exit

device(cfg)#context switch-group DEFAULT
device(ctx-swgrp)[DEFAULT]#bind interface ROUTER LAN
device(ctx-swgrp)[DEFAULT]#interface 0_1
device(if-swgrp)[DEFAULT.0_1]#exit
device(ctx-swgrp)[DEFAULT]#interface 0_2
device(if-swgrp)[DEFAULT.0_2]#exit
device(ctx-swgrp)[DEFAULT]#interface 0_3
device(if-swgrp)[DEFAULT.0_3]#exit
device(ctx-swgrp)[DEFAULT]#exit

device(cfg)#port ethernet 0 0
device(prt-eth)[0/0]#bind interface ROUTER WAN
device(prt-eth)[0/0]#exit

device(cfg)#port ethernet 0 1
device(prt-eth)[0/1]#bind switch-group DEFAULT 0_1
device(prt-eth)[0/1]#exit

device(cfg)#port ethernet 0 2
device(prt-eth)[0/2]#bind switch-group DEFAULT 0_2
device(prt-eth)[0/2]#exit

device(cfg)#port ethernet 0 3
device(prt-eth)[0/3]#bind switch-group DEFAULT 0_3
device(prt-eth)[0/3]#exit
```

Configuring Two LANs

The following example shows how to configure `port ethernet 0 0` and `port ethernet 0 1` as one LAN, and `port ethernet 0 2` and `port ethernet 0 3` as a second LAN. Devices attached to the one LAN will have no access to devices attached to the other LAN.

```
device(cfg)#context switch-group LAN1
device(ctx-swgrp)[LAN1]#interface 0_0
device(if-swgrp)[LAN1.0_0]#exit
device(ctx-swgrp)[LAN1]#interface 0_1
device(if-swgrp)[LAN1.0_1]#exit
device(ctx-swgrp)[LAN1]#exit

device(cfg)#context switch-group LAN2
device(ctx-swgrp)[LAN2]#interface 0_2
device(if-swgrp)[LAN2.0_2]#exit
device(ctx-swgrp)[LAN2]#interface 0_3
device(if-swgrp)[LAN2.0_3]#exit
device(ctx-swgrp)[LAN2]#exit

device(cfg)#port ethernet 0 0
device(prt-eth)[0/0]#bind switch-group LAN1 0_0
device(prt-eth)[0/0]#exit

device(cfg)#port ethernet 0 1
device(prt-eth)[0/1]#bind switch-group LAN1 0_1
device(prt-eth)[0/1]#exit

device(cfg)#port ethernet 0 2
device(prt-eth)[0/2]#bind switch-group LAN1 0_2
device(prt-eth)[0/2]#exit

device(cfg)#port ethernet 0 3
device(prt-eth)[0/3]#bind switch-group LAN1 0_3
device(prt-eth)[0/3]#exit
```

VLAN (802.1p/Q)

This section describes the tasks involved in configuring VLANs on switch group interfaces through the CLI.

VLANs may be used to:

- Isolate interfaces from one another. For example, if devices connected to interfaces ETHERNET_0_0 and ETHERNET_0_1 are in a separate network than devices connected to interfaces ETHERNET_0_2 and

ETHERNET_0_3, VLANs can be used to prevent traffic from being forwarded from one network to the other.

- Indicate to an external device which network a given packet came from by using VLAN tags.

VLAN configuration task list

To configure VLANs on the switch, perform the tasks described in the following sections.

Configure switch mode

In order to configure VLANs on the switching hardware, it must be configured for VLAN mode, as opposed to switch-group mode.

When the switch is put into VLAN mode, it automatically:

- Binds all ports to switch-group DEFAULT, if they are not already bound to it.
- Configures all ports to permit untagged traffic only.

Step	Command	Purpose
1	**node(cfg)#switch mode vlans**	Put the switch into a mode such that VLANs can be configured on it.

Note Some older hardware did not have the switch that supports VLANs installed. If that is the case, the switch mode command will not be available. You will need to be in VLAN mode instead of Switch Group mode. Issue the command “switch mode vlans” from the configure mode.

Enter switch group interface configuration mode

VLAN configuration takes place in the switch group interface configuration mode. Perform the following steps to enter the switch group interface configuration mode.

Mode: Configure

Step	Command	Purpose
1	**node(cfg)#context switch-group** *if-name*	Enter configuration mode for switch group *ctx-name*.
2	**node(ctx-swgrp)***[ctx-name]***#interface** *if-name*	Enter configuration mode for interface *if-name* in switch group *ctx-name*.

Permit untagged packets

To permit untagged packets and to forward them transparently, i.e. without encapsulating them in a VLAN tag, use the **permit untagged** command. This is the default, so you will only need to use this command to return the interface to its default configuration. That is, this command overrides the **permit untagged encapsulate vlan** *vlan* and **permit all encapsulate vlan** *vlan* commands that are listed below.

Mode: Switch group interface configuration

Step	Command	Purpose
1	**node(if-swgrp)***[ctx-name.if-name]***#permit untagged**	Permit untagged packets and forward them transparently.

To deny untagged traffic, use the **deny untagged** command. Note that to use this command; you must have previously used the **permit vlan** *vlan* command to permit tagged traffic. Otherwise, the interface would deny all traffic.

Mode: Switch group interface configuration

Step	Command	Purpose
1	**node(if-swgrp)***[ctx-name.if-name]***#deny untagged**	Deny untagged traffic.

Permit tagged packets

To permit tagged packets and to forward them transparently, i.e. without encapsulating them in a second VLAN tag, use the **permit vlan** *vlan* command. You may enter this command multiple times to add to the set of permitted VLAN IDs. This command overrides the **permit all encapsulate vlan** *vlan* command that is listed below.

Mode: Switch group interface configuration

Step	Command	Purpose
1	**node(if-swgrp)[***ctx-name.if-name***]#permit vlan** *vlan*	Permit packets tagged with any VLAN ID specified in *vlan*. *vlan* may be a single VLAN ID, a list, or a range. Valid VLAN IDs are 0 to 4095, inclusive. Examples are: 100, 100,200, and 100,200..299.

To remove from the set of permitted VLAN IDs, use the **deny vlan** *vlan* command.

Mode: Switch group interface configuration

Step	Command	Purpose
1	**node(if-swgrp)[***ctx-name.if-name***]#deny vlan** *vlan*	Deny traffic tagged with any VLAN ID specified in *vlan*. *vlan* may be a single VLAN ID, a list, or a range. Valid VLAN IDs are 0 to 4095, inclusive. Examples are: 100, 100,200, and 100,200..299.

Encapsulate untagged traffic

To permit untagged packets and to encapsulate them in a VLAN tag, use the **permit untagged encapsulate vlan** *vlan* command. This command overrides the **permit untagged** command that is listed above and the **permit all encapsulate vlan** *vlan* command that is listed below.

Mode: Switch group interface configuration

Step	Command	Purpose
1	**node(if-swgrp)[***ctx-name.if-name***]#permit untagged encapsulate vlan** *vlan*	Permit untagged packets and encapsulate them in a VLAN tag with ID *vlan*.

Encapsulate all traffic

To permit all packets, tagged and untagged, and to encapsulate them in a (second) VLAN tag, use the **permit untagged encapsulate vlan** *vlan* command. This command overrides the **permit untagged, permit vlan** *vlan*, and **permit untagged encapsulate vlan** *vlan* commands that are listed above.

Note Not all products support this feature.

Mode: Switch group interface configuration

Step	Command	Purpose
1	**node(if-swgrp)[***ctx-name.if-name***]#permit all encapsulate vlan** *vlan*	Permit all packets and encapsulate them in a (second) VLAN tag with ID vlan.

Examples

Example 1: Interface Isolation

In the following example, interfaces ETHERNET_0_0 and ETHERNET_0_1 are isolated from ETHERNET_0_2 and ETHERNET_0_3. That is, ETHERNET_0_0 and ETHERNET_0_1 form one virtual network and ETHERNET_0_2 and ETHERNET_0_3 for another virtual network. Neither network has access to the other.

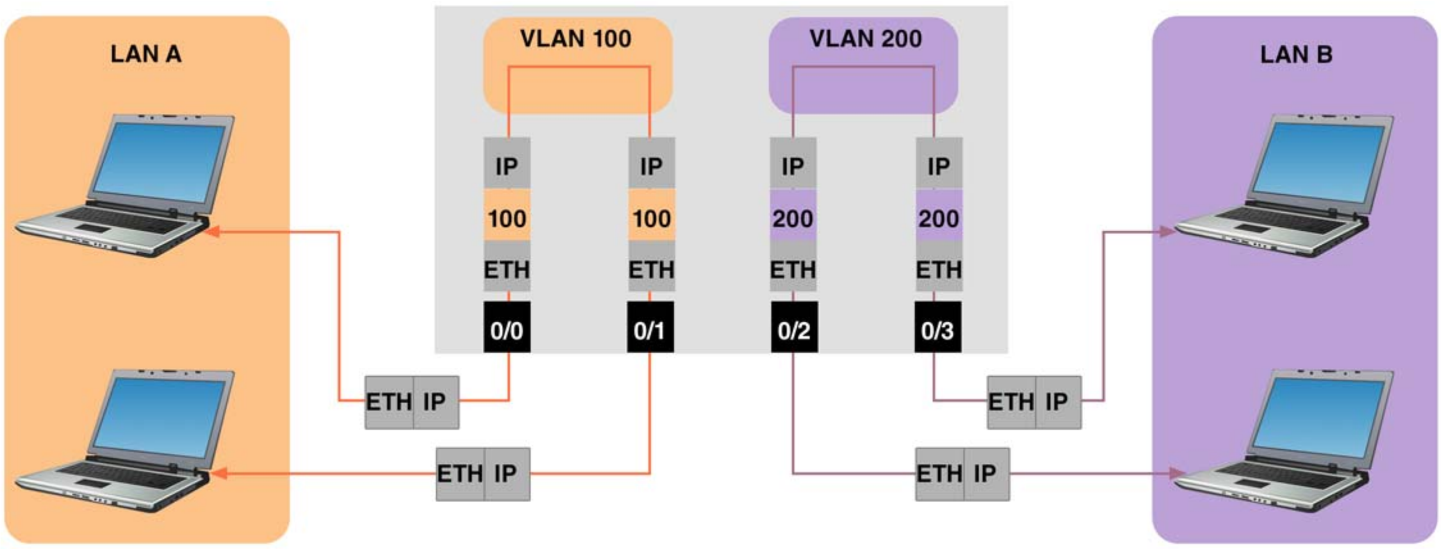


Figure 19. VLAN example 1

```
node(cfg)#context switch-group DEFAULT
node(ctx-swgrp)[DEFAULT]#interface ETHERNET_0_0
node(if-swgrp)[DEFAULT.ETHERNET_0_0]#permit untagged encapsulate vlan 100
node(if-swgrp)[DEFAULT.ETHERNET_0_0]#exit
node(ctx-swgrp)[DEFAULT]#interface ETHERNET_0_1
node(if-swgrp)[DEFAULT.ETHERNET_0_1]#permit untagged encapsulate vlan 100
node(if-swgrp)[DEFAULT.ETHERNET_0_2]#exit
node(ctx-swgrp)[DEFAULT]#interface ETHERNET_0_2
node(if-swgrp)[DEFAULT.ETHERNET_0_2]#permit untagged encapsulate vlan 200
node(if-swgrp)[DEFAULT.ETHERNET_0_2]#exit
node(ctx-swgrp)[DEFAULT]#interface ETHERNET_0_3
node(if-swgrp)[DEFAULT.ETHERNET_0_3]#permit untagged encapsulate vlan 200
node(if-swgrp)[DEFAULT.ETHERNET_0_3]#exit
```

Example 2: VLAN Tagging

In the following example, DSL_0_0, DSL_0_1, DSL_0_2, and DSL_0_3 all provide network access to different customers. ETHERNET_0_0 VLAN tags its traffic to indicate to a router which customer the traffic came from.

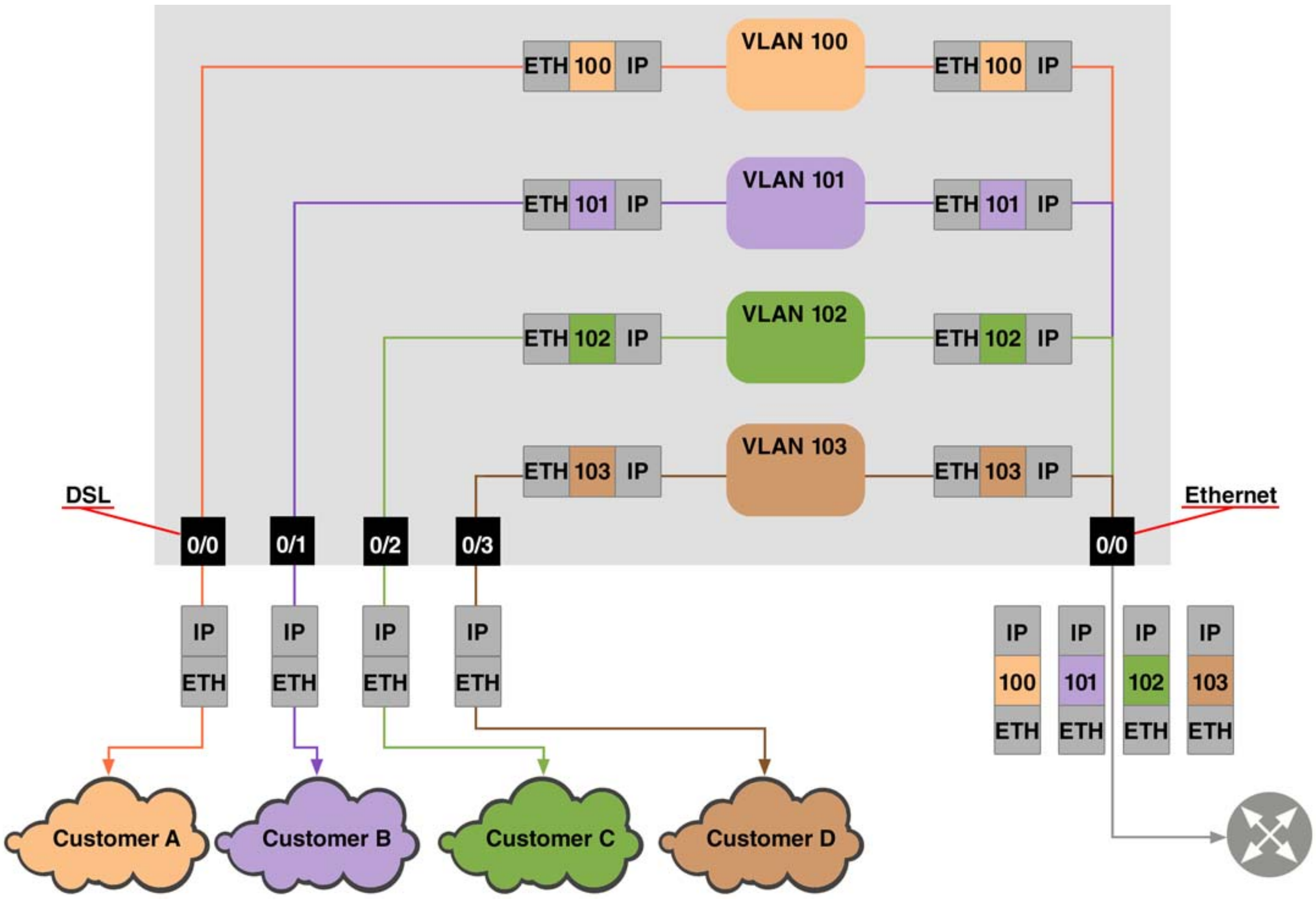


Figure 20. VLAN example 2

```
node(cfg)#context switch-group DEFAULT
node(ctx-swgrp)[DEFAULT]#interface DSL_0_0
node(if-swgrp)[DEFAULT.DSL_0_0]#permit untagged encapsulate vlan 100
node(if-swgrp)[DEFAULT.DSL_0_0]#exit
node(ctx-swgrp)[DEFAULT]#interface DSL_0_1
node(if-swgrp)[DEFAULT.DSL_0_1]#permit untagged encapsulate vlan 101
node(if-swgrp)[DEFAULT.DSL_0_1]#exit
node(ctx-swgrp)[DEFAULT]#interface DSL_0_2
node(if-swgrp)[DEFAULT.DSL_0_2]#permit untagged encapsulate vlan 102
node(if-swgrp)[DEFAULT.DSL_0_2]#exit
node(ctx-swgrp)[DEFAULT]#interface DSL_0_3
node(if-swgrp)[DEFAULT.DSL_0_3]#permit untagged encapsulate vlan 103
node(if-swgrp)[DEFAULT.DSL_0_3]#exit
node(ctx-swgrp)[DEFAULT]#interface ETHERNET_0_0
node(if-swgrp)[DEFAULT.ETHERNET_0_0]#permit vlan 100..103
node(if-swgrp)[DEFAULT.ETHERNET_0_0]#exit
```

Example 3: Q-in-Q

In the following example, DSL_0_0 and DSL_0_1 provide network access to two different customers. Both customers' networks have their own VLAN schemes which must be preserved, so ETHERNET_0_0 encapsulates traffic in a second VLAN tag so it can keep the customer's VLAN tag, yet still distinguish which customer the traffic came from.

```
node(cfg)#context switch-group DEFAULT
node(ctx-swgrp)[DEFAULT]#interface DSL_0_0
node(if-swgrp)[DEFAULT.DSL_0_0]#permit all encapsulate vlan 100
node(if-swgrp)[DEFAULT.DSL_0_0]#exit
node(ctx-swgrp)[DEFAULT]#interface DSL_0_1
node(if-swgrp)[DEFAULT.DSL_0_1]#permit all encapsulate vlan 200
node(if-swgrp)[DEFAULT.DSL_0_1]#exit
node(ctx-swgrp)[DEFAULT]#interface ETHERNET_0_0
node(if-swgrp)[DEFAULT.ETHERNET_0_0]#permit vlan 100,200
node(if-swgrp)[DEFAULT.ETHERNET_0_0]#exit
```

Access Control List Configuration

This chapter provides an overview of hardware switch Access Control Lists (ACLs) and describes the tasks involved in configuring them.

The hardware switch ACLs covered in this chapter are in contrast to the IPv4/v6 Services ACLs available on many of our router products. Hardware switch ACLs perform the packet filtering on the device's hardware switch, whereas IPv4/v6 Services ACLs perform the packet filtering on the device's main CPU. This results in the following differences:

- Hardware switch ACLs have no CPU overhead, whereas IPv4/v6 Services ACLs do.
- Hardware switch ACLs are stateless, whereas IPv4/v6 Services ACLs are stateful. That is, hardware switch ACLs do not perform connection tracking, so they cannot dynamically permit a packet based on the state of the connection it is part of.

When the term ACL is used in this chapter, it refers to a hardware switch ACL, as opposed to an IPv4/v6 Services ACL.

This chapter includes the following sections:

- About ACLs
- ACL configuration task list (see page 181)

About Access Control Lists (ACLs)

This section briefly describes what ACLs do, why and when you should configure them, and their features.

What ACLs Do

ACLs implement a firewall by filtering network traffic, forwarding or dropping each packet based on the ACL rules and bindings. ACL rules specify the criteria used to match packets, e.g. source/destination MAC address, IP address, and/or TCP ports. ACL bindings determine which ingress interface will be matched.

Note Sophisticated users can sometimes successfully evade or fool basic access lists because no authentication is required.

Why You Should Configure ACLs

You should use ACLs to provide a basic level of security for accessing your network. If you do not configure ACLs on your device, all packets passing through the device could be allowed onto all parts of your network. For example, ACLs can allow one host to access a part of your network, and prevent another host from accessing the same area. In figure 21, host A is allowed to access the Human Resources network and host B is prevented from accessing the Human Resources network.

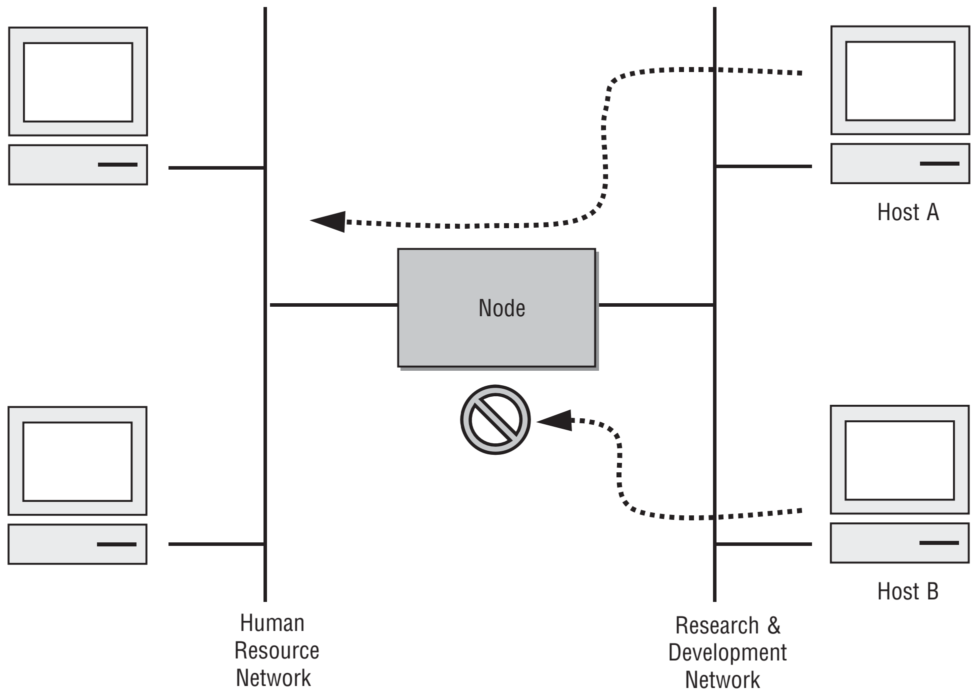


Figure 21. Using traffic filters to prevent traffic from being routed to a network

You can also use ACLs to decide which types of traffic are forwarded or blocked at the device interfaces. For example, you can permit e-mail traffic to be forwarded but at the same time block all Telnet traffic.

Features of Access Control Lists

The following features apply to all ACLs:

- The system may contain a maximum of 415 ACL rules.
- An ACL may contain multiple rules. The order of rules is significant. Each rule is processed in the order it appears in the configuration file. As soon as a rule matches, the corresponding action is taken and no further processing takes place.
- All ACLs have an implicit **deny** all rule at the end. A packet that does not match the criteria of the first rule is subjected to the criteria of the second rule and so on until the end of the ACL is reached, at which point the packet is dropped.
- An empty ACL is treated as an implicit **deny** all list.

Note Two or more administrators should not simultaneously edit the configuration file. This is especially the case with ACLs. Doing this can have unpredictable results.

Once in ACL configuration mode, each command creates a rule in the ACL. When the ACL is applied, the action performed by each rule is one of the following:

- **permit** statement causes any packet matching the criteria to be accepted.
- **deny** statement causes any packet matching the criteria to be dropped.

Access Control List (ACL) Configuration Task List

To configure an ACL, perform the tasks in the following sections.

- Mapping out the goals of the ACL
- Creating an ACL profile and entering configuration mode (see page 181)
- Adding a filter rule to the current ACL profile (see page 182)
- Binding and unbinding an ACL profile to a switch port (see page 185)
- Displaying an ACL profile (see page 185)

Mapping the Goals of the ACL

To create an ACL, you must:

- Assign a unique name to the ACL
- Define packet-filtering criteria

A single ACL can have multiple filtering criteria statements.

Before you begin to enter the commands that create and configure the ACL, be sure that you are clear about what you want to achieve with the firewall. Consider whether it is better to deny specific accesses and permit all others or to permit specific accesses and deny all others.

Note A single ACL can have multiple rules, but editing those rules online can be tedious. Therefore, we recommend editing complex ACLs offline within a configuration file and downloading the configuration file later via TFTP to your device.

Creating an ACL Profile and Entering Configuration Mode

This procedure describes how to create an ACL profile and enter ACL configuration mode.

Mode: Administrator execution

Step	Command	Purpose
1	***node*(cfg)#profile switch acl** *name*	Creates the ACL profile *name* and enters the configuration mode for this ACL.

The ACL profile name will be known by *name*. Entering this command puts you into *ACL configuration mode* where you can enter the individual statements that will make up the ACL.

Use the **no** form of this command to delete an ACL profile. You cannot delete an ACL profile if it is currently bound to a switch port.

Example: Create an ACL profile

In the following example the ACL profile named FROM_MODEM is created and the shell of the ACL configuration mode is activated.

```
node>enable
node#configure
node(cfg)#profile switch acl FROM_MODEM
node(pf-switch-acl)[FROM_MODEM]#
```

Adding and Deleting a Filter Rule to the Current ACL Profile

The commands **permit** or **deny** are used to define an ACL rule. This procedure describes how to create an ACL rule.

Mode: ACL Configuration

Step	Command	Purpose
1	***node*(pf-switch-acl)[*name*]#{permit | deny} [src-mac** *src-mac]* **[dest- mac** *dest-mac*] **[vlan-id** *vlan-id*] **[vlan-prio** *vlan-prio*] **[dscp** *dscp*] **[protocol {arp | ip | icmp | tcp | udp | sctp}] [src-ip {***src-address* | *src- network/src-mask-size* | *src-network src-mask*}] **[dest-ip {***dest-address* | *dest-network/dest-mask-size* | *dest-network dest-mask*}] **[icmp-type** *icmp-type* [*icmp-code*]] **[src-port** *src-port*] **[dest-port** *dest-port*]]	Creates an ACL rule that either permits or denies access according to the match criteria.

The table below explains the syntax:

Keyword	Meaning
permit	Forward any packet matching the match criteria.
deny	Drop any packet matching the match criteria.
src-mac	Matches packets from this MAC address, e.g. 00:a0:ba:01:23:45
dest-mac	Matches packets to this MAC address, e.g. 00:a0:ba:0a:bc:de
vlan-id	Matches packets on this VLAN. Note that this matches the VLAN that the switch internally assigns to the packet. For ports that are untagged members or access port members of a VLAN, the internal VLAN does not match the packet's VLAN ID field. See Chapter 14, "Ethernet Port Configuration" on page 156 for details. Valid values are 0 through 4094.
vlan-prio	Matches packets with this IEEE 802.1p VLAN priority field. Note that this will only match packets that were received with a VLAN tag. Valid values are 0 through 7.
dscp	Matches packets with this !DiffServ Code Point. Valid values are 0 through 63, or any of the following: af11 af12 af13 af21 af22 af23 af31 af32 af33 af41 af42 af43 cs1 cs2 cs3 cs4 cs5 cs6 cs7 ef. Requires **protocol ip, icmp, tcp, udp, or sctp** to be specified.
arp	Matches ARP packets

Keyword	Meaning
ip	Matches IPv4 packets
icmp	Matches ICMP packets
tcp	Matches TCP packets
udp	Matches UDP packets
sctp	Matches SCTP packets
src-address	Matches packets from this IPv4 host, e.g. 192.168.1.1. Requires **protocol ip, icmp, tcp, udp, or sctp** to be specified.
src-network/src-mask-size	Matches packets from this IPv4 subnet, e.g. 192.168.1.0/24. Requires **protocol ip, icmp, tcp, udp, or sctp** to be specified.
src-network src-mask	Alternate syntax to match packets from this IPv4 subnet, e.g. 192.168.1.0 255.255.255.0. Requires **protocol ip, icmp, tcp, udp, or sctp** to be specified.
dest-address	Matches packets to this IPv4 host, e.g. 192.168.2.1. Requires ***protocol ip, icmp, tcp, udp, or sctp** to be specified.
dest-network/dest-mask-size	Matches packets to this IPv4 subnet, e.g. 192.168.2.0/24. Requires **protocol ip, icmp, tcp, udp, or sctp** to be specified.
dest-network dest-mask	Alternate syntax to match packets from this IPv4 subnet, e.g. 192.168.2.0 255.255.255.0. Requires **protocol ip, icmp, tcp, udp, or sctp** to be specified.
icmp-type	Matches packets with this ICMP type. Valid values are 0 through 255, or any of the following: echo-reply destination-unreachable redirect alternate-host-address echo-request router-advertisement router-selection time-exceeded parameter-problem timestamp-request timestamp-reply information-request information-reply address-mask-request address-mask-reply traceroute datagram-conversion-error mobile-host-redirect ipv6-where-are-you ipv6-i-am-here mobile-registration-request mobile-registration-reply skip photuris. Requires **protocol ip,** to be specified.
icmp-code	Matches packets with this ICMP code. Valid values are 0 through 255. Requires **protocol ip** to be specified.
src-port	Matches packets from this TCP, UDP, or SCTP port. Requires **protocol ip, udp**, or **sctp** to be specified.
dest-port	Matches packets to this TCP, UDP, or SCTP port. Requires **protocol tcp**, **udp** or **sctp** to be specified.

If no match criteria is specified, the rule will match all packets, so if you place the rule **deny** at the top of an ACL profile, no packets will pass regardless of the other rules you defined.

Example: Create ACL rules

Select the ACL profile named FROM_MODEM and create some rules to:

- drop all ICMP echo requests (as used by the ping command),
- but forward all other ICMP traffic,
- forward any TCP traffic to host 193.14.2.10 via port 80,

- forward UDP traffic from host 62.1.2.3 to host 193.14.2.11 via any port in the range from 1024 to 2048, and
- forward all traffic from the 97.123.111.0/24 subnet to host 193.14.2.11.

```
node(cfg)#profile switch acl FROM_MODEM
node(pf-switch-acl)[FROM_MODEM]#deny protocol icmp icmp-type echo-request
node(pf-switch-acl)[FROM_MODEM]#permit protocol icmp
node(pf-switch-acl)[FROM_MODEM]#permit protocol tcp dest-ip 193.14.2.10 dest-port
80
node(pf-switch-acl)[FROM_MODEM]#permit protocol udp src-ip 62.1.2.3 dest-ip
193.14.2.11 dest-port 1024..2048
node(pf-switch-acl)[FROM_MODEM]#permit protocol ip src-ip 97.123.111.0/24 dest-ip
193.14.2.11
node(pf-switch-acl)[FROM_MODEM]#exit
node(cfg)#
```

The **no** form of the **permit** or **deny** command is used to delete an ACL rule. This procedure describes how to delete an ACL rule.

Mode: ACL Configuration

Step	Command	Purpose
1	***node*(pf-switch-acl)[*name*]#show profile switch acl *name***	Show the ACL rules in the ACL *name*. Use this command to get the index of the rule you want to delete. You will use it in step 2.
2	***node*(pf-switch-acl)[*name*]#no permit *index***	Removes the ACL rule at the specified *index*.

The table below explains the syntax:

Keyword	Meaning
permit	Delete a permit rule.
deny	Delete a drop rule.
index	The order in the ACL of the rule to delete.

Example: Delete an ACL rule

Select the ACL profile named FROM_MODEM and delete the rule to permit any TCP traffic to host 193.14.2.10 via port 80.

```
node(cfg)#profile switch acl FROM_MODEM
node(pf-switch-acl)[FROM_MODEM]#show profile switch acl FROM_MODEM
deny 1 protocol icmp icmp-type echo-request
permit 2 protocol icmp
permit 3 protocol tcp dest-ip 193.14.2.10 dest-port 80
permit 4 protocol udp src-ip 62.1.2.3 dest-ip 193.14.2.11 dest-port 1024..2048
permit 5 src-ip 97.123.111.0/24 dest-ip 193.14.2.11
node(pf-switch-acl)[FROM_MODEM]#no permit 3
node(pf-switch-acl)[FROM_MODEM]#exit
node(cfg)#
```

Binding and Unbinding an ACL Profile to a Switch Port

The command **use** is used to bind an ACL profile to a switch port. This procedure describes how to bind an ACL profile to incoming packets on a switch port.

Mode: Switch Port

Step	Command	Purpose
1	***node*(switch-port-*type*)[*slot*/*port*]#use acl** *name* **in**	Binds ACL profile *name* to incoming packets on port *type slot port.*

The **no** form of the **use** command is used to unbind an ACL profile from a port. This procedure describes how to unbind an ACL profile from a port.

Mode: Switch Port

Step	Command	Purpose
1	***node*(switch-port-*type*)[*slot*/*port*]#no use acl in**	Unbinds the ACL profile from incoming packets on port *type slot port.*

The table below explains the syntax:

Keyword	Meaning
type	The type of the port to which the ACL profile is bound, e.g. ethernet or dsl.
slot	The port number of the port to which the ACL profile is bound.
port	The port number of the port to which the ACL profile is bound.
name	The name of an ACL profile that has already been created using the **profile switch acl** command. This is not used for the **no** form of the command.

Example: Bind and unbind an ACL profile to/from a port

Bind ACL profile FROM_MODEM to incoming packets on the port ethernet 0 0.

```
node(cfg)#port ethernet 0 0
node(prt-eth)[0/0]#switch
node(switch-port-ethernet)[0/0]#use acl FROM_MODEM in
```

Unbind the ACL profile from incoming packets on the port ethernet 0 0.

```
node(cfg)#port ethernet 0 0
node(prt-eth)[0/0]#switch
node(switch-port-ethernet)[0/0]#no use acl in
```

Displaying an ACL Profile

The **show profile switch acl** command displays the indicated ACL profile. If no specific profile is selected, then all created ACL profiles are shown.

This procedure describes how to display a certain ACL profile.

Mode: Administrator execution or any other mode, except the operator execution mode

Step	Command	Purpose
1	***node*#show profile switch acl** *name*	Displays the ACL profile *name.*

Example: Displaying an ACL entry

The following example shows how to display the ACL profile named FROM_MODEM.

```
node#show profile switch acl FROM_MODEM
deny 1 protocol icmp icmp-type echo-request
permit 2 protocol icmp
permit 3 protocol tcp dest-ip 193.14.2.10 dest-port 80
permit 4 protocol udp src-ip 62.1.2.3 dest-ip 193.14.2.11 dest-port 1024..2048
permit 5 src-ip 97.123.111.0/24 dest-ip 193.14.2.11
```

QoS Traffic Scheduler

Some products include an internal Ethernet switch to which the external ports are connected. This switch provides efficient bridging between the Ethernet and Ethernet-like ports. This chapter explains how to configure the switch to provide a differing quality of service (QoS) to packets of different types.

About QoS

There are two main aspects to implementing QoS: packet classification and packet scheduling.

Packet classification is performed by an access control list, In addition to filtering packets, as described in the Chapter 14, "Hardware Switching: Access Control List Configuration" on page 102, an access control list can also assign packets to a class of service profile.

The class of service profile then assigns the packet to a traffic class.

Packet scheduling is performed by a service policy. The service policy schedules packets for transmission based the packet's traffic class.

In addition, you may optionally configure rate shaping on the transmit port. Rate shaping can be used in conjunction with packet scheduling, but it can also be used independently.

Packet walkthrough

A packet received by an Ethernet switch port is processed as follows:

1. If the port is bound to an access control list, that access control list assigns the packet to a class of service profile.
2. The class of service profile assigns the packet to a traffic class.
3. The switch selects a port to transmit the packet out of based on the packets destination MAC address.
4. The port enqueues the packet for transmission based on the packet's traffic class. The port has 8 transmit queues, one for each transmit class. If the port's transmit queue that corresponds to the packet's traffic class is full, then the packet is dropped. This is called tail-dropping.

5. When the port is finished transmitting a packet, if its transmit rate shaper is enabled, it waits to dequeue the next packet such that the bandwidth will remain under the specified rate. Otherwise it proceeds immediately to the next step.
6. The port dequeues a packet from one of its transmit queues and begins transmitting it. Which transmit queue it dequeues from is based on its service policy's configuration. The service policy configures each traffic class for one of two scheduling algorithms:
   - **Priority:** Packets will be dequeued and transmitted from the traffic class's queue until there is a packet in a higher priority traffic classes queue. (Traffic class 7 is highest priority and 0 is lowest.) This means that packets assigned to lower priority traffic classes will never be transmitted as long as there are packets assigned to this traffic class. Lower priority traffic classes queues will fill up and packets assigned to those traffic classes will be dropped if there is enough traffic assigned to the higher priority traffic class.
   - **Shared (shaped deficit weighted round robin, or SDWRR):** Traffic classes that are configured for shared will share the bandwidth available while the higher priority traffic classes' queues are empty. Each shared traffic class is guaranteed at least its configured percentage of the available bandwidth, but may use more if the other traffic classes require less bandwidth than their configured percentage.

Note Traffic classes are configured for shared scheduling should be contiguous or else the behavior will be unexpected. For example, it is acceptable for traffic classes 0 and 6-7 to be configured for priority and traffic classes 1-5 to be configured for shared, but it is not acceptable for classes 1 and 6-7 to configured for priority and traffic classes 0 and 2-5 to be configured for shared because 0 and 2 are not contiguous.

There are 4 different service policies that may each be configured independently. Each Ethernet switch port is assigned to one of these 4 service policies.

QoS Traffic Scheduler Configuration Task List

To configure a service policy, perform the tasks in the following sections:

- Create a class of service profile and assign its traffic class (see page 187)
- Create an access control list profile and create classifier rules (see page 188)
- Binding an access control list to the receiving switch port (see page 188)
- Configure traffic classes' scheduling modes (see page 188)
- Binding a service policy to the transmitting switch port (see page 189)
- Configuring transmit rate shaping for a switch port (see page 190)

Create a class of service profile and assign its traffic class

This procedure describes how to create a class of service profile and assign its traffic class. The traffic class, together with the service policy used by the egress port, determines what priority will be given to the packet internally within the switch. Higher priority traffic should be assigned to higher traffic classes.

Mode: Configure

Step	Command	Purpose
1	**node(cfg)#profile switch cos** *cos-name*	Creates the class of service profile *cos-name* and enters its configuration mode.
2	**node(pf-switch-cos)[***cos-name***]#traffic-class** *traffic-class*	Configures the traffic class to which packets will be assigned. Valid values are 0 to 7, inclusive, which higher numbers being treated with higher priority.

Example: Creating a class of service profile and assigning its traffic class

Create class of service WEB with traffic class 1, giving it the second to lowest priority.

```
node(cfg)#profile switch cos WEB
node(pf-switch-cos)[WEB]#traffic-class 1
```

Create an access control list profile and create classifier rules

This procedure describes how to create an access control list profile and create rules to classify traffic.

Mode: Configure

Step	Command	Purpose
1	**node(cfg)#profile switch acl** *acl-name*	Creates the access control list *acl-name* and enters its configuration mode
2	**node(pf-switch-acl)[***acl-name***]#permit** *match* **set cose** *cos-name*	Assign packets matching *match* criteria to existing class of service profile *cos-name*. See "Adding and Deleting a Filter Rule to the Current ACL Profile" on page 106 for details on //match// syntax.

Binding an access control list to the receiving switch port

This procedure describes how to bind an access control list to the switch port that will receive the packets.

Mode: Configure

Step	Command	Purpose
1	**node(cfg)#port** *type slot port*	Enters port configuration mode.
2	**node(prt-***type***)[***slot /port***]#switch**	Enters Ethernet switch port configuration mode.
3	**node(port-switch-***type***)[***slot /port***]#use acl** *acl-name* **in**	Configure the switch port to classify incoming traffic according to access control list *acl-name*

Configure traffic classes' scheduling modes

This procedure describes how to configure a traffic classes scheduling modes.

Mode: Configure

Step	Command	Purpose
1	**node(cfg)#profile switch service-policy** *srvpl-name*	Enters the configuration mode for service policy *srvpl-name*
2	**node(pf-srvpl)[***srvpl-name***]#source traffic-class** *traffic-class*	Enters the configuration mode for the traffic class *traffic-class* within service policy *srvpl-name*
3	**node(src)#[***srvpl-name.traffic-class***]#{priority | share** *percent***}**	Configures all ports that use service policy *srvpl-name* to dequeue from their transmit queue for traffic class *traffic-class* using either the **priority** algorithm, or the **share** algorithm with weight *percent*.

Example: Configuring traffic classes' scheduling modes

Use priority scheduling for highest priority traffic class (7) and share traffic equally among traffic classes 0-3.

```
node(cfg)#profile switch service-policy 0
node(pf-srvpl)[0]#source traffic-class 7
node(src)[0.7]#priority
node(src)[0.7]#source traffic-class 3
node(src)[0.3]#share 25
node(src)[0.3]#source traffic-class 2
node(src)[0.2]#share 25
node(src)[0.2]#source traffic-class 1
node(src)[0.1]#share 25
node(src)[0.1]#source traffic-class 0
node(src)[0.0]#share 25
```

Binding a service policy to the transmitting switch port

This procedure describes how to bind a service policy to a switch port that will transmit the packets.

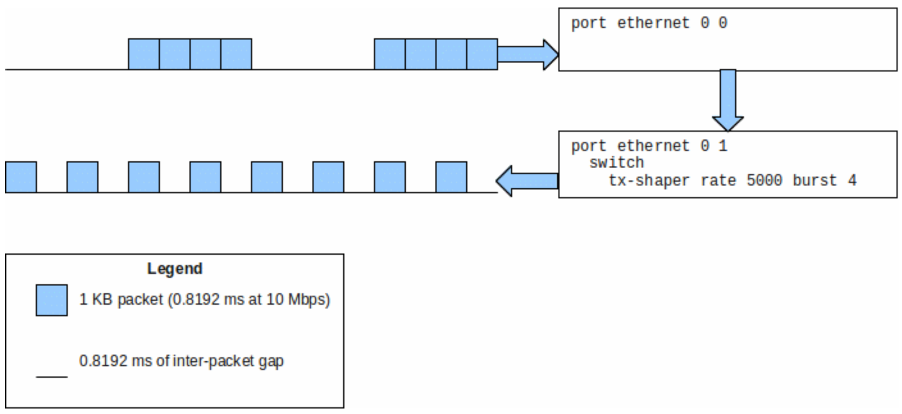
By default all ports use service policy 0.

Mode: Configure

Step	Command	Purpose
1	**node(cfg)#port** *type slot port*	Enters port configuration mode.
2	**node(prt-***type***)[***slot /port***]#switch**	Enters Ethernet switch port configuration mode.
3	**node(port-switch-***type***)[***slot /port***]#use service-policy** *srvpl-name*	Configure the switch port to schedule outgoing traffic according to service policy *srvpl-name*

Configuring transmit rate shaping for a switch port

This is an optional task where, when configured, the port will limit its transmit bandwidth to the specified rate. It will also absorb the specified burst before it drops packets. The following diagram illustrates how the transmit shaper works:


port ethernet 0 0
port ethernet 0 1
switch
tx-shaper rate 5000 burst 4
Legend
1 KB packet (0.8192 ms at 10 Mbps)
0.8192 ms of inter-packet gap

In the diagram, suppose both Ethernet 0/0 and Ethernet 0/1 are running at 10 Mbps. Ethernet 0/1 is configured to shape its traffic at 5000 kilobits per second (i.e. 5 megabits per second) and to absorb up to 4 kilobytes of burst.A station connected to Ethernet 0/0 is transmitting 1 kilobyte packets in bursts of 4 and then pausing for 4 packet times so that its average rate over the course of a second is 5 megabits per second. Ethernet 0/1 shapes or "smooths" the traffic so that it transmits at a constant 5 megabits per second.

This might be useful if Ethernet 0/1 were connected to a DSL modem that had a 5 megabit per second connection to the far end. If the DSL modem had only a two packet buffer, and if Ethernet 0/1 were to transmit the 4 packet bursts, the DSL modem would receive the first packet in the burst and still be transmitting it when the second packet arrived, so the second packet would be dropped.

While most modern DSL modems would have more than 4 kilobits of packet buffers, it is still possible that the internal Ethernet switch has more buffer capacity than an external device. The internal Ethernet switch allows up to a 16 megabyte burst to be configured, which many external devices would not have.

Mode: Configure

Step	Command	Purpose
1	**node(cfg)#port** *type slot port*	Enters port configuration mode.
2	**node(prt-***type***)[***slot* /*port***]#switch**	Enters Ethernet switch port configuration mode.
3	**node(port-switch-***type***)[***slot* /*port***]#tx-shaper rate** *rate* **burst** *burst*	Configures transmit shaping on the given port. The rate is specified in kilobits per second, and the burst is specified in kilobytes. The supported rates are based on the burst size, so if an unsupported rate is given the system will automatically select the closest supported rate.

Example: Configuring transmit rate shaping for a switch port

The following example shows how to configure the Ethernet port on slot 0 and port 0 to shape its transmit traffic to 1 megabit per second and to absorb up to 128 kilobytes of burst.

```
node(cfg)#port ethernet 0 0
node(prt-eth)[0/0]#switch
node(switch-port-ethernet)[0/0]#tx-shaper rate 1000 burst 128
```

Example

The following example provides the following quality of service to packets received by port ethernet 0 0 and transmitted by port ethernet 0 1 depending on their DiffServ code point:

- Any packets arriving with a DSCP of EF (expedited forwarding) will be forwarded immediately.
- Any packets arriving with a DSCP of AFX1 (assured forwarding) will share the bandwidth left over from the EF traffic. AF41 is assured of at least 40% of the remaining bandwidth, AF31 is assured of at least 30%, AF21 is assured of at least 20%, and AF11 is assured of at least 10%.
- Any other traffic is given “best effort,” i.e. any bandwidth remaining after all EF and AF traffic have been transmitted will be given to it.

Note that the access control list profile is bound to the receiving port (in this example, port ethernet 0 0), and the service policy profile is bound to the transmitting port (in this example, port ethernet 0 1). In addition, port ethernet 0 1 is connected to a DSL modem running at 5.7 Mbps, so we enable its rate shaper to limit its transmit rate to 5.7 Mbps.

```
node>enable
node#configure
node(cfg)#profile switch cos EF
node(pf-switch-cos)[EF]#traffic class 7
node(pf-switch-cos)[EF]#exit
node(cfg)#profile switch cos AF41
node(pf-switch-cos)[AF41]#traffic class 4
node(pf-switch-cos)[AF41]#exit
node(cfg)#profile switch cos AF31
node(pf-switch-cos)[AF31]#traffic class 3
node(pf-switch-cos)[AF31]#exit
node(cfg)#profile switch cos AF21
node(pf-switch-cos)[AF21]#traffic class 2
node(pf-switch-cos)[AF21]#exit
node(cfg)#profile switch cos AF11
node(pf-switch-cos)[AF11]#traffic class 1
node(pf-switch-cos)[AF11]#exit
node(cfg)#profile switch cos BE
node(pf-switch-cos)[BE]#traffic class 0
node(pf-switch-cos)[BE]#exit
node(cfg)#profile switch acl CLASSIFIER
node(pf-switch-acl)[CLASSIFIER]#permit dscp ef protocol ip set cos EF
node(pf-switch-acl)[CLASSIFIER]#permit dscp af41 protocol ip set cos AF41
node(pf-switch-acl)[CLASSIFIER]#permit dscp af31 protocol ip set cos AF31
node(pf-switch-acl)[CLASSIFIER]#permit dscp af21 protocol ip set cos AF21
node(pf-switch-acl)[CLASSIFIER]#permit dscp af11 protocol ip set cos AF11
node(pf-switch-acl)[CLASSIFIER]#permit set cos BE
node(pf-switch-acl)[CLASSIFIER]#exit
node(cfg)#profile switch service-policy 1
node(pf-srvpl)[1]#source traffic-class 7
node(src)[1.7]#priority
node(src)[1.7]#exit
node(pf-srvpl)[1]#source traffic-class 4
node(src)[1.4]#share 40
node(src)[1.4]#exit
node(pf-srvpl)[1]#source traffic-class 3
node(src)[1.3]#share 30
node(src)[1.3]#exit
node(pf-srvpl)[1]#source traffic-class 2
node(src)[1.2]#share 20
node(src)[1.2]#exit
node(pf-srvpl)[1]#source traffic-class 1
node(src)[1.1]#share 10
node(src)[1.1]#exit
node(pf-srvpl)[1]#source traffic-class 0
node(src)[1.0]#priority
node(src)[1.0]#exit
node(pf-srvpl)[1]#exit
node(cfg)#port ethernet 0 0
node(prt-eth)[0/0]#switch
node(switch-port-ethernet[0/0]#use acl CLASSIFIER in
node(cfg)#port ethernet 0 1
node(prt-eth)[0/1]#switch
node(switch-port-ethernet[0/1]#use service-policy 1
node(switch-port-ethernet[0/1]#tx-shaper rate 5700 burst 128
```

ToS Stripping and Prioritization

Some products include an internal Ethernet switch to which the external ports are connected. This switch provides efficient bridging between the Ethernet and Ethernet-like ports. This chapter describes how to configure the switch to set the VLAN priority and/or DSCP fields of transmitted packets to indicate to the receiving device what quality of service (QoS) to provide to each packet. (Note that these fields are only indications, so it depends on the receiving device supporting and being configured for QoS.)

Note Hardware Switching applies to the ForeFront product series only.

About ToS stripping and prioritization

There are two common methods used to indicate to other devices what quality of service should be given to packets:

- VLAN priority field
- IP DSCP field (replaces the older IP ToS field)

Setting either of these fields is referred to as ToS stripping and prioritization. This is implemented as follows:

- Packet classification is performed by an access control list, In addition to filtering packets, as described in the chapter 14, "Hardware Switching: Access Control List Configuration" on page 102, an access control list can also assign packets to a class of service profile.
- The class of service profile then assigns the packet's VLAN priority and/or DSCP field value(s).

ToS Stripping and Prioritization Configuration Task List

To configure VLAN priority or DSCP packet modification, perform the tasks in the following sections:

- Create a class of service profile and configure its VLAN priority and/or DSCP (see page 193)

- Create an access control list profile and create classifier rules (see page 193)
- Binding an access control list to the receiving switch port (see page 193)

Create a class of service profile and configure its VLAN priority and/or DSCP

This procedure describes how to create a class of service profile and configure its VLAN priority and/or DSCP.

If the VLAN priority is configured, then any packet assigned to this class of service will be transmitted with its VLAN priority field set to the configured value. Otherwise, it will be transmitted with the same VLAN priority field it arrived with. Note that the VLAN priority field is part of the VLAN header, so if the packet is transmitted untagged, this configuration has no effect.

If the DSCP is configured, then any packet assigned to this class of service will be transmitted with its DSCP field set to the configured value. Otherwise, it will be transmitted with the same DSCP field it arrived with. Note that the DSCP field is part of the IP header, so if the packet is not an IP packet, this configuration has no effect.

Mode: Configure

Step	Command	Purpose
1	**device(cfg)#profile switch cos** *cos-name*	Creates the class of service profile *cos-name* and enters its configuration mode.
2	**device(pf-switch-cos)[***cos-name***]#set layer2 cos** *priority*	Optional. Configures the value placed in the VLAN tag's priority field. Valid values are 0 to 7, inclusive.
3	**device(pf-switch-cos)[***cos-name***]#set ip dscp** *dscp*	Optional. Configures the value placed in the IP header's DSCP field. Valid values are 0 to 63, inclusive, or any of the following: af11 af12 af13 af21 af22 af23 af31 af32 af33 af41 af42 af43 cs1 cs2 cs3 cs4 cs5 cs6 cs7 ef

Create an access control list profile and create classifier rules

This procedure describes how to create an access control list profile and create rules to classify traffic.

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#profile switch acl** *acl-name*	Creates the ACL profile *acl-name* and enters the configuration mode for this ACL.
2	**device(pf-switch-acl)[***acl-name***]#permit** *match* **set cose** *cos-name*	Assign packets matching *match* criteria to existing class of service profile *cos-name*. See "Adding and Deleting a Filter Rule to the Current ACL Profile" on page 106 for details on *match* syntax.

Binding an access control list to the receiving switch port

This procedure describes how to bind an access control list to the switch port that will receive the packets.

Mode: Configure

Step	Command	Purpose
1	**device(cfg)#port** *type slot port*	Enters port configuration mode.
2	**device(prt-***type***)[***slot* /*port***]#switch**	Enters Ethernet switch port configuration mode.
3	**device(port-switch-***type***)[***slot* /*port***]#use acl** *acl-name* **in**	Configure the switch port to classify incoming traffic according to access control list *acl-name*

Example

The following example causes packets received by port ethernet 0 0 to be transmitted with VLAN priority and DSCP as follows:

- SSH and Telnet (TCP ports 22 and 23, respectively) are used for device management on this network, so they are transmitted with VLAN priority 3 (critical applications) and DSCP EF (expedited forwarding) to tell other devices to give them higher priority.
- Everything else is considered data, so it is transmitted with VLAN priority 0 (best effort) and DSCP 0 (best effort) to tell other devices to give them lower priority.

```
device>enable
device#configure
device(cfg)#profile switch cos MGMT
device(pf-switch-cos)[VOICE]#set layer2 cos 3
device(pf-switch-cos)[VOICE]#set ip dscp ef
device(pf-switch-cos)[VOICE]#exit
device(cfg)#profile switch cos DATA
device(pf-switch-cos)[VOICE]#set layer2 cos 0
device(pf-switch-cos)[VOICE]#set ip dscp 0
device(pf-switch-cos)[VOICE]#exit
device(cfg)#profile switch acl CLASSIFIER
device(pf-switch-acl)[CLASSIFIER]#permit protocol tcp dest-port 22 set cos MGMT
device(pf-switch-acl)[CLASSIFIER]#permit protocol tcp dest-port 23 set cos MGMT
device(pf-switch-acl)[CLASSIFIER]#permit set cos DATA
device(pf-switch-acl)[CLASSIFIER]#exit
device(cfg)#port ethernet 0 0
device(prt-eth)[0/0]#switch
device(switch-port-ethernet[0/0]#use acl CLASSIFIER in
```

MAC Filter Configuration

Some products include an internal Ethernet switch to which the external ports are connected. This switch provides efficient bridging between the Ethernet and Ethernet-like ports. These ports support blocking received packets based on the source MAC address, destination MAC address, VLAN or any combination thereof. This chapter describes the tasks involved in configuring the switch's ports to filter traffic based on the Ethernet header information.

Note Hardware Switching applies to the ForeFront product series only.

Ethernet Switch MAC Filter Configuration Task List

To configure the Ethernet switch port, perform the tasks described in the following sections:

- Creating a MAC filter profile and enter configuration mode (see page 195)
- Adding a filter rule to the current MAC filter profile (see page 195)
- Binding and unbinding a MAC filter profile to an Ethernet switch port (see page 195)
- Displaying a MAC filter profile (see page 196)

Creating a MAC Filter Profile and Enter Configuration Mode

Mode: Configure

Step	Command	Purpose
1	**device(cfg)#profile switch mac-filter** *name*	Creates the MAC filter profile *name* and enters the configuration mode for this profile. By default, the MAC filter has no rules and will block all received packets.

Adding a Filter Rule to the Current MAC Filter Profile

Mode: Profile MAC filter

Step	Command	Purpose
1	**device(pf-switch-mac-filter)#permit {***src* **| any} {***dest* **| any} [***vid***]**	Creates a rule that permits packets matching all of the supplied command options.

Where the syntax is:

- **src**—The source MAC address to be matched, e.g. 00:a0:ba:01:23:45
- **any**—Indicates that any MAC address is matched.
- **dest**—The destination MAC address to be matched, e.g. 00:a0:ba:01:23:45
- **vid**—The VLAN ID to be matched. Valid values are 2 through 4094. Note that this matches against the VLAN; the switch assigns to the packet for internal processing. For ports that are untagged members or access port members of a VLAN, the internal VLAN does not match the packet's VLAN. See Chapter 14, "Ethernet Port Configuration" for details.

Binding and Unbinding a MAC Filter to an Ethernet Switch Port

The command **use** is used to bind a MAC filter profile to an Ethernet switch port. This procedure describes how to bind a MAC filter profile to incoming packets on an Ethernet switch port.

Mode: Configure

Step	Command	Purpose
1	**device(cfg)#port** *type slot port*	Enter port configuration mode.
2	**device(prt-*type*)#**[*slot* /*port*]**#switch**	Enter Ethernet switch port configuration mode.
3	**device(switch-port-*type*)#**[*slot* / *port*]**#use mac-filter** *name*	Binds the MAC filter profile *name* to incoming packets on Ethernet switch port *type slot/port*.

The **no** form of the **use** command is used to unbind a MAC filter profile from an Ethernet switch port. When using this form, the name of a MAC filter profile represented by the *name* argument above, is not required. This procedure describes how to unbind a MAC filter profile from incoming packets on an Ethernet switch port.

Mode: Ethernet switch port

Step	Command	Purpose
1	**device(switch-port-*type*)#**[*slot*/*port*]**#no use mac-filter**	Unbinds the MAC filter profile for incoming packets on Ethernet switch port *type slot/port*.

Example: Bind and unbind a MAC filter profile to an Ethernet switch port

Create a MAC filter profile that allows only hosts 00:a0:ba:01:23:45 and 00:a0:ba:01:ab:cd to be connected to the Ethernet port on slot 0 and port 0.

```
device(cfg)#profile switch mac-filter FILTER
device(pf-switch-mac-filter)[FILTER]#permit 00:a0:ba:01:23:45 any
device(pf-switch-mac-filter)[FILTER]#permit 00:a0:ba:01:ab:cd any
device(pf-switch-mac-filter)[FILTER]#exit
device(cfg)#port ethernet 0 0
device(prt-eth)[0/0]#switch
device(switch-port-ethernet)[0/0]#use mac-filter FILTER
```

Unbind a MAC filter profile from a port.

```
device(cfg)#port ethernet 0 0
device(prt-eth)[0/0]#switch
device(switch-port-ethernet)[0/0]#no use mac-filter
```

Note When unbinding a MAC filter, the profile *name* argument is not required since only one MAC filter profile can be active at a time on a certain Ethernet switch port.

Displaying a MAC Filter Profile

The **show profile switch mac-filter** command displays the indicated MAC filter profile. If no specific profile is selected then all installed MAC filter profiles are shown. The command shows the profile's rules and the ports that are using the profile.

This procedure describes how to display a certain MAC filter profile.

Mode: Administrator execution or any other mode, except the operator execution mode

Step	Command	Purpose
1	**device#show profile switch mac-filter** *name*	Displays the MAC filter profile *name*.

Example: Displaying a MAC filter profile

The following example shows how to display the MAC filter profile named *FILTER*.

```
device#show profile switch mac-filter FILTER
FILTER
```

```
===============================================
Source            Destination       VLAN
----------------- ----------------- ----
00:a0:ba:01:23:45
00:a0:ba:01:ab:cd

Port                 : Ethernet 0/0
```

Trunk Configuration

Link aggression control protocol (LACP), a.k.a. Trunk, is a protocol used to determine if two or more physical links are connected to the same device and can be used as a single logical link. Some products include an internal Ethernet switch to which the external ports are connected. This switch provides efficient bridging between the Ethernet and Ethernet-like ports. Up to 8 of these ports may be bonded into a single logical link using IEEE 802.1AX link aggregation. This provides both increased bandwidth and redundancy. This is referred to as "trunking." This chapter describes the tasks involved in configuring the switch's ports for trunking through the CLI.

Note Hardware Switching applies to the ForeFront product series only.

Ethernet Switch Trunk Configuration Task List

To configure an Ethernet switch trunk, perform the tasks described in the following sections:

- Creating a trunk profile and enter configuration mode (see page 197)
- Binding and unbinding a trunk profile to an Ethernet switch port (see page 197)
- Displaying an Ethernet switch trunk (see page 198)
- Debugging an Ethernet switch trunk (see page 198)

Creating a Trunk Profile and Enter Configuration Mode

Mode: Configure

Step	Command	Purpose
1	**device(cfg)#profile switch trunk** *name*	Creates the trunk profile *name* and enter the configuration mode for this profile.

Binding and Unbinding a Trunk Profile to an Ethernet Switch Port

The command **use** is used to bind a trunk profile to an Ethernet switch port.

Mode: Configure

Step	Command	Purpose
1	**device(cfg)#port** *type slot port*	Enters port configuration mode.
2	**device(prt-*type*)[*slot* /*port*]#switch**	Enters Ethernet switch port configuration mode.
3	**device(switch-port-*type*)**[*slot* / *port*]#**use trunk** *name*	Binds trunk profile *name* to Ethernet switch port *type slot/ port*. All ports that are bound to the same trunk profile are grouped together into a single logical link.

The **no** form of the **use** command is used to unbind a trunk profile from an Ethernet switch port. When using this form, the name of a trunk profile represented by the name argument above, is not required.

This procedure describes how to unbind a trunk profile from an Ethernet switch port.

Mode: Ethernet Switch port

Step	Command	Purpose
1	**device(switch-port-*type*)**[*slot / port*]**#no use trunk**	Unbinds the trunk profile from the Ethernet switch port *type slot/port*.

Example: Binding and unbinding a MAC filter profile to an Ethernet switch port

Bond Ethernet 0/1 and Ethernet 0/2 together into a single logical link.

```
device(cfg)#profile switch trunk TRUNK
device(pf-switch-trunk)[FILTER]#exit
device(cfg)#port ethernet 0 1
device(prt-eth)[0/1]#switch
device(switch-port-ethernet)[0/1]#use trunk TRUNK
device(switch-port-ethernet)[0/1]#exit
device(prt-eth)[0/1]#exit
device(cfg)#port ethernet 0 2
device(prt-eth)[0/2]#switch
device(switch-port-ethernet)[0/2]#use trunk TRUNK
```

Displaying an Ethernet Switch Trunk

The **show profile switch trunk** command displays the indicated trunk profile. If no specific profile is selected then all installed trunk profiles are shown. The ports that are using the specified trunk are displayed.

This procedure describes how to display a certain trunk profile.

Mode: Administrator execution or any other mode, except the operator execution mode

Step	Command	Purpose
1	**device#show profile switch acl** *name*	Displays the trunk profile *name.*

Example: Displaying a trunk profile

The following example shows how to display the trunk profile named TRUNK.

```
device#show profile switch trunk TRUNK
TRUNK
===============================================
Port                 : Ethernet 0/1
Port                 : Ethernet 0/2
```

Debugging an Ethernet Switch Trunk

The **trace lacp** command is used to debug link aggregation control protocol negotiation during system operation. Use the **no** form of this command to disable any debug output.

This procedure describes how to debug the trunk profiles.

Mode: Administrator execution of any other mode, except the operator execution

Step	Command	Purpose
1	**device#trace lacp { debug [detail** *detail* **| full-detail] |** *level* **}**	Enables the Ethernet switch trunk debug monitor.

Example: Debugging Ethernet switch trunk profiles

The following example enables the debug monitor for Ethernet switch trunk profiles globally.

```
device#trace lacp debug full-detail
```

The following example enables the debug monitor for Ethernet switch trunk profiles globally.

```
device#no trace lacp
```

Ethernet Service Policy Configuration

Some products include an internal Ethernet switch to which the external ports are connected. This switch provides efficient bridging between the Ethernet and Ethernet-like ports such as G.SHDSL and EFM ports. The switch is able to be configured to provide a differing quality of service (QoS) to packets of different types. This chapter provides an overview of Ethernet switch service policies, which are an integral part of QoS configuration.

In addition, see Chapter 33, “Access Control List Configuration” on page 331 and Chapter 15, “Hardware Switching: QoS Traffic Scheduler” on page 111 for other components that are necessary to configure QoS.

About QoS

There are two main aspects to implementing QoS: packet classification and packet scheduling.

Packet classification is performed by a profile classifier. The classifier is part of the overall Quality-of-Service architecture of Trinity. As depicted in figure 47 on page 316, the classifier groups packet flows into virtual traffic-classes.

Packet Walkthrough

A packet received by an Ethernet switch port is processed as follows:

1. If the port is bound to a profile classifier, that profile classifier assigns the packet to a class of service profile.
2. The class of service profile assigns the packet to a traffic class.
3. The switch selects a port to transmit the packet out of, based on the packets destination MAC address.
4. The port enqueues the packet for transmission based on the packet's traffic class. The port has 8 transmit queues, one for each transmit class. If the port's transmit queue that corresponds to the packet's traffic class is full, then the packet is dropped. This is called tail-dropping.
5. When the port is finished transmitting a packet, it dequeues a packet from one of its transmit queues and begins transmitting it. Which transmit queue it dequeues from is based on its service policy's configuration. The service policy configures each traffic class for one of two scheduling algorithms:
   – **Priority:** Packets will be dequeued and transmitted from the traffic class's queue until there is a packet in a higher priority traffic class's queue. (Traffic class 7 is the highest priority and 0 is the lowest.) This

means that packets assigned to lower priority traffic classes will never be transmitted as long as there are packets assigned to this traffic class. Lower priority traffic classes' queues will fill up and packets assigned to those traffic classes will be dropped if there is enough traffic assigned to the higher priority traffic class.

– **Shared** (shaped deficit weighted round robin): Traffic classes that are configured for shared will share the bandwidth available while the higher priority traffic classes' queues are empty. Each shared traffic class is guaranteed at least its configured percentage of the available bandwidth, but may use more if the other traffic classes require less bandwidth than their configured percentage.

Note Traffic classes configured for shared scheduling should be contiguous or else the behavior will be unexpected. For example, it is acceptable for traffic classes 0 and 6-7 to be configured for priority and traffic classes 1-5 to be configured for shared; however, it is not acceptable for classes 1 and 6-7 to be configured for priority and traffic classes 0 and 2-5 to be configured for shared, because 0 and 2 are not contiguous.

There are 4 different service policies that may each be configured independently. Each Ethernet switch port is assigned to one of these 4 service policies.

Ethernet Switch Service Policy Configuration Task List

To configure a service policy, perform the tasks in the following sections.

Configure Traffic Class for Priority Scheduling

This procedure describes how to configure a traffic class for priority scheduling.

Mode: Configure

Step	Command	Purpose
1	**device(cfg)#profile switch service-policy** *name*	Enters the configuration mode for service policy *name*.
2	**device(pf-srvpl) device#source traf-fic-class** *traffic-class*	Enters the configuration mode for the traffic class *traffic-class* within service policy *name*.
3	**device(src)**[*name.traffic-class*]**#prior-ity**	Configures all ports that use service policy *name* to dequeue from their transmit queue for traffic class *traffic-class* using the priority algorithm.

Configure Traffic Class for Shared Scheduling

This procedure describes how to configure a traffic class for shared scheduling.

Mode: Configure

Step	Command	Purpose
1	**device(cfg)#profile switch service-policy** *name*	Enters the configuration mode for service policy *name*.
2	**device(pf-srvpl)** device**#source traf-fic-class** *traffic-class*	Enters the configuration mode for the traffic class *traffic-class* within service policy *name*.
3	**device(src)**[*name.traffic-class*]**#share** *percent*	Configures all ports that use service policy *name* to dequeue from their transmit queue for traffic class *traffic-class* using the shaped deficit weighted round robin (SDWRR) algorithm with a weight of *percent*.

Binding a Service Policy to an Ethernet Switch Port

This procedure describes how to bind a service policy to an Ethernet switch port. By default all ports use service policy 0.

Mode: Configure

Step	Command	Purpose
1	**device(cfg)#port** *type slot port*	Enters port configuration mode.
2	**device(prt-type)**[*slot*/*port*]**#switch**	Enters Ethernet switch port configuration mode.
3	**device(port-switch-type)**[*slot*/*port*]**#use service-policy** *name*	Configures the Ethernet switch port to schedule transmission according to service policy *name*.

Chapter 17 DSL Port Configuration

Chapter contents

Introduction

This chapter provides an overview of the DSL ports, their characteristics and the tasks involved in the configuration.

G.SHDSL EFM Setup

Configuring the Mode for the G.SHDSL Connection

An EFM DSL device transports Ethernet packets directly over the connection. A standard DSL connection operates over a 2-wire interface. Where hardware will allow it, DSL wire pairs may be bonded together into one link by changing the service-mode. The available modes are listed below:

- 2-wire – Standard 2-Wire mode. Data is byte-interleaved, where subsequent bytes in the data stream are sent on alternating pairs.
- 4-wire – Enhanced 4-Wire mode, meaning the DSL line pairs can be activated and train independently. If one pair droops from Showtime, the other pair drops and they must both be restarted.
- 8-wire – Enhanced 8-Wire mode, meaning the DSL line pairs can be activated and train independently. If one pair drops from Showtime, the other pair drops and they must both be restarted.

Mode: port dsl 0 0

Step	Command	Purpose
1	**device(prt-dsl)[0/0]#service-mode 2-wire | 4-wire | 8-wire**	Define the mode for the DSL connection. Default: 4-wire.

Example: Configuration of the 4-wire service mode

```
port dsl 0 0
service-mode 4-wire
annex-type

 bind interface ETH_DSL router
```

One some devices, G.SHDSL ports can be configured as CO or CPE mode.

Mode: port dsl 0 0

Step	Command	Purpose
1	**device(prt-dsl)[0/0]#mode co | cpe**	Define the mode for the DSL connection.

Configuring the Annex Type for the G.SHDSL Connection

In order for the DSL link to connect, Patton devices supporting a 4-wire G.SHDSL card as a CPE client must configure the DSL annex type to match the annex type on the CO side.

Mode: port dsl 0 0

Step	Command	Purpose
1	***device*(prt-dsl)[0/0]#annex-type a-f | b-g**	Define the annex type for the DSL connection.

Example: Configuration of the 4-wire annex type

```
port dsl 0 0
service-mode 4-wire
annex-type a-b
 bind interface ETH_DSL router
```

Configuring the Payload Data Rate for the G.SHDSL Connection

The payload-rate command allows configuring the bit rate mode to adaptive (commonly used) or specifying a fixed value for the payload data rate. The payload data rate is configured for a single 2-wire pair also in case of a 4-wire connection. The valid range for this command changes based on the "tcpam" command explained later.

Note The payload rate is only valid in 64 Kbps increments as shown in Table 17 on page 206, however the command will allow any integer value in the range to be entered. The entered value will be automatically adjusted to a valid rate.

Mode: port dsl 0 0

Step	Command	Purpose
1	***[device]*(prt-dsl)[0/0]#payload-rate { adaptive [max <192..15296>] | <192..15296> }**	Define the data rate for the DSL connection. Default: adaptive.

Table 17 on page 206 provides an overview of the available payload rates. When the "payload-rate" command is configured as "adaptive", the discovered rate can be adjusted with the "snr-margin" command. This adjusts the acceptable SNR of the link.

Step	Command	Purpose
1	***[device]*(prt-dsl)[0/0]#snr-margin <-10..22>**	Adjust the signal sensitivity of the adaptive payload rate. Default:6

Table 17. Payload Rate Configuration Overview

payload-rate	payload-rate	payload-rate	payload-rate
192	896	1536	2240
256	960	1600	2304
320	1024	1664	2560
384	1088	1728	3392
448	1152	1792	3840
512	1216	1920	5056
576	1280	1984	5696
704	1344	2048	
768	1408	2112	
832	1472	2176	

Configuring the TCPAM for the G.SHDSL connection

The TCPAM setting provides a means to adjust the max configurable data rate. The default setting of “auto(16/32)” allows for the best compatibility with other EFM equipment, and provides a max payload rate of 5696.

Some hardware allows a TCPAM configuration of “auto(64/128)”. This mode is only compatible with some other Patton hardware, but allows for a payload rate of 15296.

Step	Command	Purpose
1	**device(prt-dsl)[0/0]#tcpam { auto(16/32) | auto(64/128) | 4 | 8 | 16 | 32 | 64 | 128 }**	Configure the TCPAM of the link. Default: auto(16/32)

Note TCPAM-4 and TCPAM-8 provide the DSL ports with longer reach and greater stability in noisy environments. The maximum payload rate is 2496 kbps for TCPAM-4, and 5056 kbps for TCPAM-8. These are non-standard extensions to G.SHDSL, so they are only guaranteed to work with two of these products back-to-back.

DSL Emergency Freeze

This feature allows the LINE to survive "micro disruptions" without retraining the link. The performance of this feature improves with lower data rates, distance, and TCPAM settings.

Mode: port dsl <slot> <port>

Step	Command	Purpose
1	**device(prt-dsl)[<slot>/<port>]# [no] emergency-freeze**	Configures the emergency-freeze feature to survive micro breaks in the DSL line.

DSL TX Power Increase

This command may be used to to increase the DSL/line ports transmit power up to 0.7 dB. This is non-standard, but it may provide a more stable link in a noisy environment.

Mode: port dsl <slot> <port>

Step	Command	Purpose
1	**device(prt-dsl)[0/0]#tx-power-increase <dB>**	Increases the ports transmit power dB dBs above what is specified by the standard.
2	**device(prt-dsl)[0/0]#no tx-power-increase**	Sets the ports transmit power to what is specified by the standard.

DSL Suspect Mode

When one of the DSL ports in a 4-wire or 8-wire bundle is connected through a very noisy cable, it may flap repeatedly. Some data will be dropped every time it flaps.

Suspect mode monitors a link for repeated link drops and will suspend the link for a period of time.

Mode: port dsl <slot> <port>

Step	Command	Purpose
1	**device(pf-dsl)[DEFAULT]#suspect-mode [drops <drops> period <period> duration <duration>]**	Suspend a port for duration seconds if it drops drops times within period seconds.
2	**device(prt-dsl)[0/0]#clear suspect-flag**	Re-enable any suspended ports.

Multiport G.SHDSL Devices

The 3310RC and 3310P devices are equipped with multiple G.SHDSL ports that can be configured independently of each other. These devices pass Ethernet traffic directly, and use a sophisticated switching system. The ports on these devices are grouped into slots, with each slot having a set number of ports. (i.e.: The FF3310 has 6 slots with 4 ports.) The **service-mode** command can be used to bond G.S ports within a slot. (The master port in a bond consumes the salve port, i.e.: If dsl 0/0 is configured as **service-mode 4-wire**, dsl 0/1 is no longer configurable independently.)

To make configuring multiple ports easier, the following previously discussed options are available in **profile dsl** instead of **port dsl**, and a **use profile** command is provided in the port: annex-type, mode, payload-rate, snr margin, and tcpam.

Configuring the Profile for the G.SHDSL Connection

In order for the DSL link to connect, Patton devices supporting a 4-wire G.SHDSL card as a CPE client must configure the DSL annex type to match the annex type on the CO side.

Mode: port dsl 0 0

Step	Command	Purpose
1	**device(prt-dsl)[0/0]#use profile** *<name>*	Define the dsl profile to use. (Required) Default: DEFAULT

Example: Configuration of the dsl profile

```
profile del DEFAULT
annex-type a-f
mode co
payload-rate 5064
```

```
    port dsl 0 0
    service-mode 4-wire
    use profile DEFAULT
```

Troubleshooting DSL Connections

Link State

- Verify that the DSL link is established (status LED is continuously on)

Debugging

Step	Command	Purpose
1	**debug dsl [{detail <level>|full-detail|}]**	Enables the DSL debugs.

Chapter 18 Context Bridge

Chapter contents

Introduction

This chapter outlines the Context Bridge. You will obtain a fundamental understanding of how to setup your Patton Trinity Embedded device for bridge configurations. The Context Bridge in Trinity is a high level conceptual entity that is responsible for the management of relaying and filtering of frames from at least two physical Ethernet ports found at the MAC layer of the OSI model.

Bridge group Configuration Task List

The following sections describe how to configure/use the Bridging component:

- Creating a bridge group
- Setting various bridge options
- Bind resources to the bridge-group
- Enable filters on the bridge-group
- Set STP options
- Configure VLANs

Mode: context bridge

Table 18. Creating Bridge Group

Step	Command	Purpose
1	**node# context bridge**	Enters the Bridge configuration context.
2	**node~(ctx-br)# bridge-group <name>**	Creates a new bridge group with the name <name>.

Table 19. Setting Various Bridge Options

Step	Command	Purpose
1	**node~(ctx-br)[TEST}# ageing <seconds>**	Sets the desired ageing options. Default is 299.95secs. The ageing is how long a host information will be kept in the bridge database before updating it.
2	**node~(ctx-br)[TEST}# arp**	Enables ARP protocol on the bridge port
3	**node~(ctx-br)[TEST}# multicast**	Enables IP Multicast on the bridge port
4	**node~(ctx-br)[TEST}# bind**	Binds resources to the bridge group
5	**node~(ctx-br)[TEST}# filter**	Enable various filters on the bridge group
6	**node~(ctx-br)[TEST}# stp**	Set STP Options
7	**node~(ctx-br)[TEST}# vlan <id>**	Enters VLAN Configuration mode
8	**node~(ctx-br)[TEST}# settap**	Taps the interface to mirror communications useful in debugging in situations like using Wireshark.

Table 20. Bind Resources to the Bridge-Group

Step	Command	Purpose
1	**node~(ctx-br)[TEST}# bind interface <IP_Interface>**	Binds this bridge-group to an IP Interface. In other words, it allows the bridge-group to have the IP settings. Remember a bridge-group is just like an Ethernet port for the most part.

Binding an IP Interface to a bridge-group works identical to binding an interface to an Ethernet port.

Table 21. Enable Filters on the Bridge-Group

Step	Command	Purpose
1	**node~(ctx-br)[TEST}# filter mac**	Configures IP Table rules based on mac address.
2	**node~(ctx-br)[TEST}# filter stp**	Configures STP Filters based on interfaces.

Table 22. Set STP Options

Step	Command	Purpose
1	**node~(ctx-br)[TEST}# stp bridgeprio <prio>**	Set bridge priority [0-65535] (Default: 32768).
2	**node~(ctx-br)[TEST}# stp fwdelay <seconds>**	Set bridge forwarding delay in seconds (Default:15s).
3	**node~(ctx-br)[TEST}# stp hello <seconds>**	Set hello time interval in seconds (Default: 20s).
4	**node~(ctx-br)[TEST}# stp maxage <seconds>**	Set maximum hello message age in seconds (Default: 20s).

All VLAN options are identical through VLAN configuration options for normal interfaces. See "HW Switching—VLAN (802.1p/Q)" on page 95 for a detailed guide.

Table 23. Configure VLANs

Step	Command	Purpose
1	**node~(ctx-br)[TEST}# vlan <id>**	Enter VLAN Confirmation mode <id> = [1..4094]
2	**node~(vlan)[br-TEST.1]# arp**	Enables ARP.
3	**node~(vlan)[br-TEST.1]# mtu**	Sets MTU.
4	**node~(vlan)[br-TEST.1]# multicast**	Enables multicast.
5	**node~(vlan)[br-TEST.1]# map**	Maps VLAN class of service (CoS) value to internal traffic class and vice-versa
6	**node~(vlan)[br-TEST.1]# bind**	Binds a resources to a VLAN. You can bind this VLAN to an IP Interface or you can bind it to a bridge-group.

VLANs take one port and split it into several logical Ethernet port, therefore binding resources to a VLAN is the same as binding a resource to an Ethernet Port. See "IP Context Overview" on page 229 for details.

Mode: administrator access

Table 24. Show Bridge-Group Configuration

Step	Command	Purpose
1	**node# show bridge <name>**	Shows bridge-group configuration for <name>.

Chapter 19 Spanning Tree Configuration

Chapter contents

Introduction

Devices that have Ethernet switch ports support the following spanning tree protocols:

- Classic spanning tree (STP)
- Rapid spanning tree (RSTP) as defined in IEEE 802.1w
- Multiple spanning tree (MSTP) as defined in IEEE 802.1s

Note This chapter is intended to provide general, non-product specific information on classic STP, RSTP and MSTP, which is widely available online and in print.

Each of the spanning tree protocols is used to prevent loops in a layer 2 network. Loops must not be present in a layer 2 network because broadcast packets will be continuously rebroadcast and use up all of the network's bandwidth.

However, having multiple connections between switches is desirable because it allows for redundancy. If one of its connections goes down, it may use one of its other connections to reach the other switch.

All of the spanning tree protocols allow for redundancy by allowing multiple connections between switches. One of the connections is put in the *forwarding* state, while the others are put in the *blocking* state. If the connection in the *forwarding* state goes down, one of the connections in the *blocking* state transitions to the *forwarding* state. Rapid spanning tree protocol (RSTP) improves on classic spanning tree by transitioning faster to the *forwarding* state, resulting in less downtime.

In addition, multiple spanning tree protocol (MSTP) allows for load balancing by mapping one set of VLANs to one spanning tree instance, another set of VLANs to another spanning tree instance, and so on. Each of the spanning tree instances may have different ports that are *forwarding*. So VLANs 100 and 101 may be forwarded out of port 1, while VLANs 200 and 201 are forwarded out of port 2. This makes better use of the available bandwidth. Then, if port 2 goes down, VLANs 200 and 201 will be forwarded out of port 1, in addition to VLANs 100 and 101.

Spanning Tree Configuration Task List

By default, the device is configured to run the multiple spanning tree protocol version (MSTP). This is the recommended configuration because MSTP is backwards compatible with the other spanning tree protocol versions. That is, if another device that only supports STP or RSTP is connected to one of this device's ports, this device will automatically fall back to STP or RSTP.

By default, the MST configuration name is empty and the revision is zero. These must be configured or else the device will run the rapid spanning tree protocol version (RSTP). All devices in a MST region must have the same MST configuration name and revision.

By default, there is only one configured spanning tree instance, the common and internal spanning tree (CIST), with all VLANs mapped to it. All devices in a MST region must have the same VLAN to spanning tree instance mapping.

By default, the spanning tree protocol is disabled on all ports. It must be enabled on a per-port basis.

Configuring Global Spanning Tree Parameters

Mode: Configure

Step	Command	Purpose
1	**device(cfg)#switch spanning tree**	Enters global spanning tree configuration mode.
2	**device(stp)#protocol-version {stp | rstp | mstp}**	Configures the spanning tree protocol version.
3	**device(stp)#tx-hold-count** *count*	Optional: Configures the transmit hold count. This should normally be left at the default value.
4	**device(stp)#mst-config name** *name*	N/A for STP and RSTP: Configures the MST configuration name, a string up to 32 characters long.
5	**device(stp)#mst-config revision** *revision*	N/A for STP and RSTP: Configures the MST configuration revision, a number 0 to 65535, inclusive.

Configuring Per-tree Spanning Tree Parameters

Mode: Global spanning tree configuration

Step	Command	Purpose
1	**device(stp)#tree {cist | msti** *instance-id}*	Creates a new spanning tree instance if it doesn't exist, and enters its configuration mode. For STP and RSTP, only the CIST is applicable. For MSTP, *instance-id* may be 1 to 64, inclusive. The no form of this command may be used to destroy any instance other than the CIST.
2	**device(stp)** [*instance-id*]**#vlan** *vlan-id*	Maps a VLAN to this spanning tree instance. For STP and RSTP, this does not apply.
3	**device(stp)** [*instance-id*]**#priority** *priority*	Configures this device's priority in this spanning tree instance's root bridge. The device that you want to be the root must be given the lowest priority.
4	**device(stp)** [*instance-id*]**#max-age** *seconds*	This only applies to the CIST; and configures the max age in seconds. A Bridge Protocol Data Unit (BPDU) will no longer be forwarded after its max age expires. It must be between 6 and 40, inclusive, and also be less than or equal to 2 * (forward-delay - 1). This should normally be left at the default value.
5	**device(stp)** [*instance-id*]**#forward-delay** *seconds*	This only applies to the CIST; and configures the forward delay in seconds. A port will wait this long before entering the forwarding state. It must be between 4 and 30, inclusive, and also be greater than to (max-age / 2) + 1. This should normally be left at the default value.
6	**device(stp)** [*instance-id*]**#max-hops** *count*	Configures the max hops. This is the maximum numbers of bridges a BPDU may be forwarded to before it expires.

Configuring Per-port/Per-tree Spanning Tree Parameters

Mode: Switch port configuration

Step	Command	Purpose
1	**device(switch-port-type)** [*slot/ port*]**#spanning-tree {cist | msti** *instance-id*}	Enters the per-port/per-tree spanning tree configuration mode.
2	**device(stp-type)** [*slot/port*][*instance-id*]**#priority** *priority*	Configures this port's priority in this spanning tree instance, a number 0 to 240, inclusive, in steps of 16. This is used as a tie-breaker if two ports can both reach the root bridge with the same path cost. The port with the lower priority will be the root port, and the other port will be the alternate port.

Enabling Spanning Tree on a Port

Mode: Switch port configuration

Step	Command	Purpose
1	**device(switch-port-type)** [*slot/ port*]**#spanning-tree**	Enables the spanning tree protocol on the given port. The no form of the command disables the spanning tree protocol on this port.

Debugging Spanning Tree

The **show switch spanning-tree** command may be used to show the spanning tree configuration as well as the current topology.

Mode: Operator execution

Step	Command	Purpose
1	**device#show switch spanning-tree**	Show the spanning tree configuration and the current topol-ogy.

Additionally, the **trace ethsw-stp** command may be used to log information about the spanning tree protocol operation.

Mode: Operator execution

Step	Command	Purpose
1	**device#trace ethsw-stp debug detail** *level*	Enables logging of the spanning tree protocol operation; *level* may be 1 (topology changes), 2 (state machine transitions), or 3 (BPDUs send and received).

Spanning Tree Configuration Example

Example: Configure the FF3310RC for the following network:

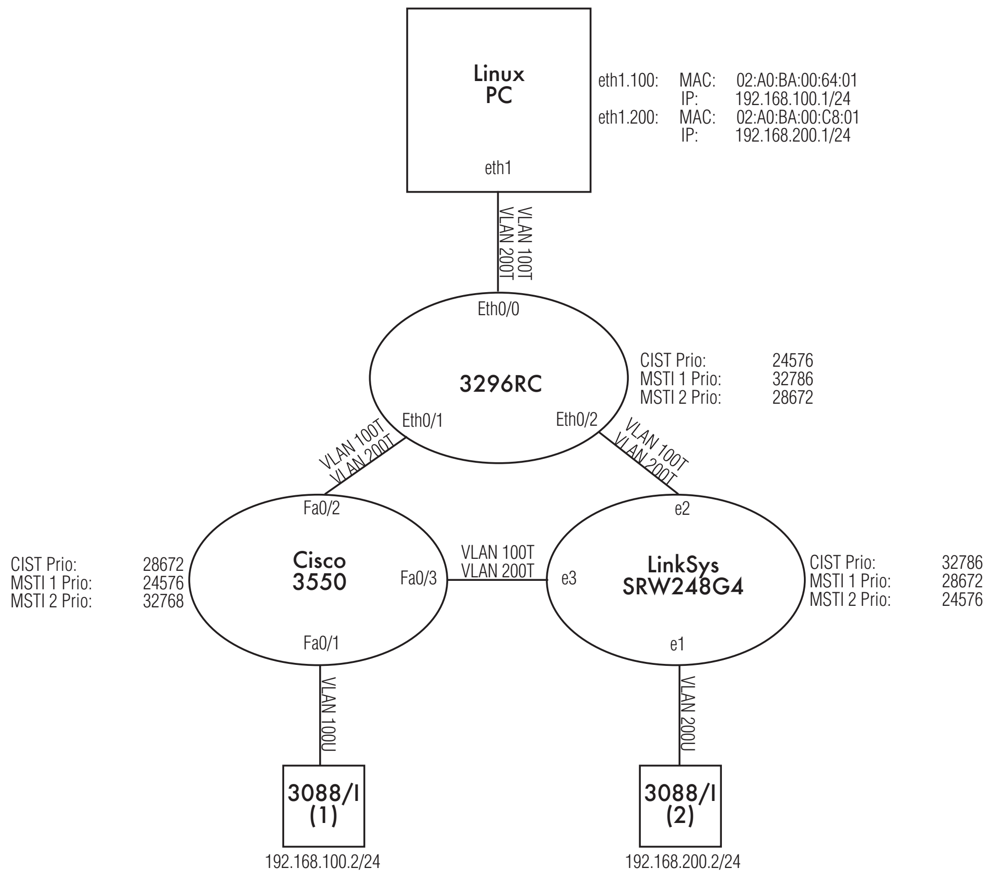


Figure 22. Spanning tree configuration

In this network, there are three switches, the FF3310RC, the Cisco 3550, and the LinkSys SRW248G4. The FF3310RC is the CIST root, the Cisco 3550 is the MSTI 1 root, and the LinkSys SRW248G4 is the MSTI 2 root. Any one of the three links may go down and the Linux PC will still be able to access both 3088/Is.

```
device(cfg)#switch spanning-tree
device(stp)#mst-config name REGION_A
device(stp)#tree cist
device(stp)[0]#priority 24576
device(stp)[0]#exit
device(stp)#tree msti 1
device(stp)[1]#vlan 100
device(stp)[1]#exit
device(stp)#tree msti 2
device(stp)[2]#priority 28672
device(stp)[2]#vlan 200
device(stp)[2]#exit
device(stp)#exit
device(cfg)#port ethernet 0 0
device(prt-eth)[0/0]#no shutdown
device(prt-eth)[0/0]#switch
device(switch-port-ethernet)[0/0]#vlan 100 tagged
device(switch-port-ethernet)[0/0]#vlan 200 tagged
device(switch-port-ethernet)[0/0]#exit
device(prt-eth)[0/0]#exit
device(cfg)#port ethernet 0 1
device(prt-eth)[0/1]#no shutdown
device(prt-eth)[0/1]#switch
device(switch-port-ethernet)[0/1]#vlan 100 tagged
device(switch-port-ethernet)[0/1]#vlan 200 tagged
device(switch-port-ethernet)[0/1]#exit
device(prt-eth)[0/1]#spanning-tree
device(prt-eth)[0/1]#exit
device(cfg)#port ethernet 0 2
device(prt-eth)[0/2]#no shutdown
device(prt-eth)[0/2]#switch
device(switch-port-ethernet)[0/1]#vlan 100 tagged
device(switch-port-ethernet)[0/1]#vlan 200 tagged
device(switch-port-ethernet)[0/1]#exit
device(prt-eth)[0/2]#spanning-tree
device(prt-eth)[0/2]#exit
device(cfg)#
```

Chapter 20 PPP Configuration

Chapter contents

Introduction

This chapter describes how to configure the point-to-point protocol over different link layers.

The Point-to-Point Protocol (PPP) provides a standard method for transporting multi-protocol datagrams over point-to-point links as defined by the RFC1661 etc. Trinity offers PPP over the following link layers:

- PPP over Ethernet (PPPoE)

Figure 23 shows the elements involved in the configuration of PPP. The elements required to configure PPP over the Ethernet are located in the upper left corner of the figure.

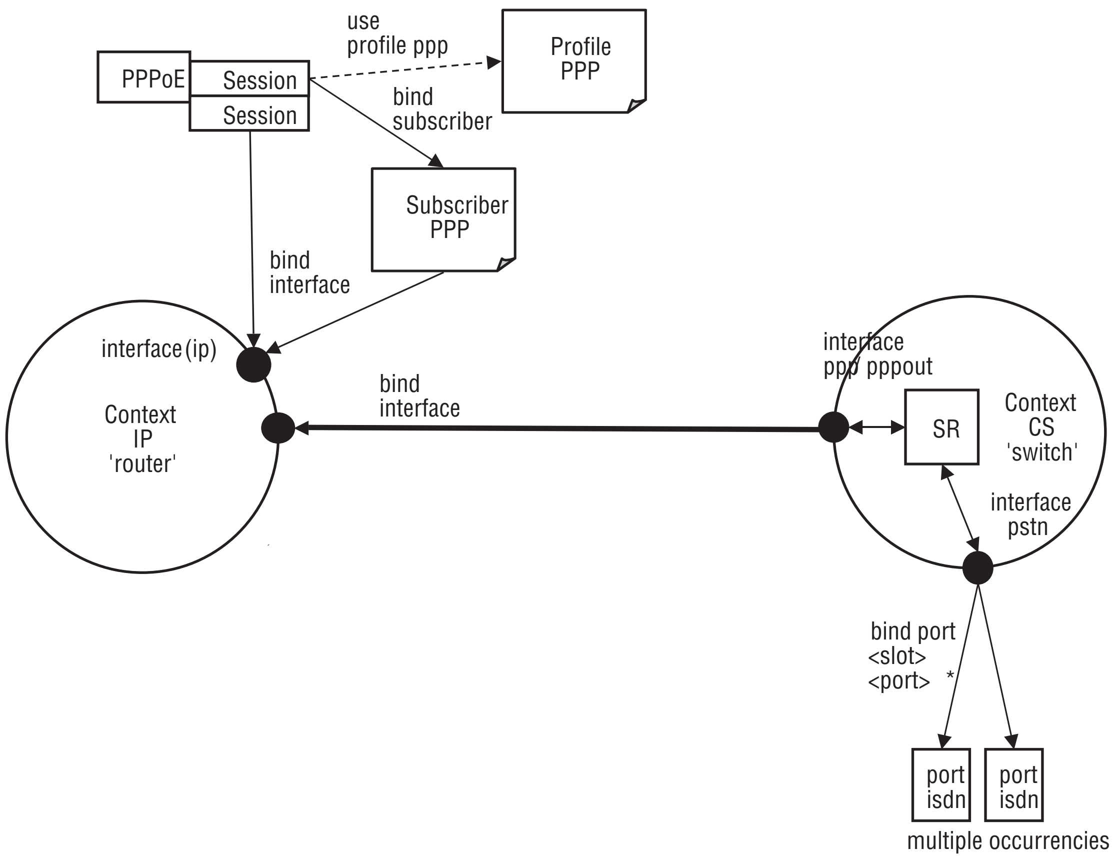


Figure 23. PPP configuration overview

Since the purpose of PPP is providing IP connectivity over different types of link layers, all PPP configuration elements connect to the IP context through an IP interface. This connection is relayed via PPP session.

For PPP over Ethernet, a PPPoE session must be configured on the respective Ethernet port. It is possible to set-up several (limited by the available memory) PPPoE sessions on the same Ethernet port, each session with its own IP interface. In addition to these PPPoE sessions, pure IP traffic can run concurrently over the same Ethernet port. This is achieved by binding the Ethernet port directly to an IP interface.

PPP Configuration Task List

To configure PPP, perform the following tasks:

- Creating an IP interface for PPP
- Configuring for IP address auto-configuration from PPP (see page 224)
- Creating a PPP session (for authentication) (see page 222)
- Configuring a PPPoE session (see page 224)
- Creating a PPP profile (see page 224)
- Displaying PPP configuration information (see page 226)
- Debugging PPP (see page 226)

Creating an IP Interface for PPP

An IP interface is required to link a PPP connection to the IP context. The IP interface must apply a network address port translation (NAPT) if the IP addresses on the LAN shall be private and hidden behind a public IP address (see Chapter 25, “NAT/NAPT Configuration” for more information about NAPT).

This procedure describes how to create an IP interface for PPP.

Mode: Context IP

Step	Command	Purpose
1	***node*****(ctx-ip)[ROUTER]#interface *name***	Creates the new interface *name*, which represents an IP interface.

Step	Command	Purpose
2	**device(if-ip)[<name>]#ipaddress [<label>] ipcp [request] [*ip-address*] [peer [request] [*peer-ip-address*]] [ignore dns]**	• If no address label is specified, the name of the interface will be taken. Negotiate the IP address with the PPP remote peer using PPP's IP control protocol (IPCP). • If *ip-address* is not specified, we will require the PPP remote peer to offer us our local IP address. • If **request** *ip-address* is specified, we will request the PPP remote peer to offer us *ip-address* as our local IP address, but we will accept another local IP address, if it rejects our request. • If *ip-address* is specified, then we will request the PPP remote peer to offer us *ip-address* as our local IP address, and we will fail to connect unless it accepts our request. • If *peer-ip-address* is not specified, we will require the PPP remote peer to assign its own local IP address and tell us what it is. • If **request** *peer-ip-address* is specified, we will request the PPP remote peer to assign itself *peer-ip-address* as its own local IP address, but we will accept another peer IP address if it rejects our request. • If *peer-ip-address* is specified, then we will require the PPP remote peer to assign itself *peer-ip-address* as its own local IP address, and we will fail to connect unless it accepts our request. • By default, we will request up to two DNS servers from the PPP remote peer, but we will not if **ignore dns** is specified.
3 (optional)	***node*(if-ip)[name]#use profile napt** *name*	Assigns the NAPT profile *name* to applied to this IP interface.

Example: Create an IP interface for PPP

The following procedure creates an IP interface that can be used for all three types of link layers. The command lines **tcp adjust-mss** only apply to Ethernet link layers.

```
node(cfg)#context ip ROUTER
node(ctx-ip)[ROUTER]#interface PPP_INTERFACE
node(if-ip)[ROUTER.PPP_INT~]#ipaddress IPCP
```

Creating a PPP Session

One or more PPP session will be configured if either PPP peer requires authentication. This procedure describes how to create a PPP session:

Mode: Configure

Step	Command	Purpose
1	***node*(cfg) # session ppp<session-name>**	Creates the new session *name.*
2	***node*(session-ppp)[name]# [no] authentication { (chap pap) | {chap|pap} }**	Defines the authentication protocol to be used, PAP and/or CHAP.
3 (optional)	***node*(session-ppp)[name]# [no] identification {outbound|inbound} *user* [password** *password*]	Sets the credentials to be provided during the authentication procedure: • the user name: *user* • the password: *password* The keywords ‘inbound’ and ‘outbound’ define the direction of authentication: • ‘inbound’: The local peer checks the credentials that the remote peer sends. • ‘outbound’: The local peer sends its credentials if the remote peer requests them. The following restrictions apply to the direction of authentication: • PPP over Ethernet: ‘outbound’ only
4	***node*(session-ppp)[name]# [no] bind interface [ROUTER]** *interface*	Binds the session to the IP interface to be used for this PPP connection. The IP interface must already exist and shall have the configuration as outlined in section “Creating an IP Interface for PPP” on page 221.
5 **(optional)**	***node* (session-ppp)[name]# [no] use profile ppp <*name*>**	Assigns a PPP profile other than the default profile to this PPP session.
6	***node*(session-ppp)[name]#[no] shutdown**	Initiates the establishment of the PPP session and the PPP connection. **Note** A PPP link, such as a PPPoE session must be bind to this PPP session before it can initiate.

Example: Create a PPP session

The procedure below creates a PPP session for a PAP authentication with some Internet Service Provider.

```
node(cfg)#session ppp JOE_EXAMLE
node(session-ppp)[JOE_EXA~]]#authentication pap
node(session-ppp)[JOE_EXA~]]#identification outbound joeexample@isp.com password
blue4you
node(session-ppp)[JOE_EXA~]]#bind interface ROUTER PPP_INTERFACE
node(session-ppp)[JOE_EXA~]]# no shutdown
```

Configuring a PPPoE Session

PPP can run over Ethernet (PPPoE). The *active discovery* protocol identifies the PPP remote peer on the Ethernet and establishes a PPPoE session with it. The PPPoE session provides a logical point-to-point link that runs PPP as if it was a physical point-to-point link (e.g. a serial link).

This procedure describes how to configure an Ethernet port and a session for PPPoE:

Mode: Configure

Step	Command	Purpose
1	***node*(cfg) #port ethernet** *slot port*	Enters Ethernet port configuration mode for the interface on *slot* and *port*
2 (optional)	***node* (prt-eth)[slot/port]# [no] bind interface** *name* **[ROUTER]**	Binds the Ethernet port to the IP interface to be used for the direct IP traffic.
3	***node*(prt-eth)[slot/port]#[no] shutdown**	Enables the ethernet port
4	***node*(prt-eth)[slot/port]#session pppoe** *name*	Creates PPPoE session with the name: *name*
5	***node*(session)[name]# [no] bind session ppp** *name*	Binds the PPPoE session to the PPP session *name*.
6 (optional)	***node*(session)[name]#service** *Service-Name*	Defines the tag 'Service-Name' to be supplied with Active Discovery in order to identify the desired remote peer (check whether the remote peer supports this feature)
7 (optional)	***node*(session)[name]#access-concentrator** *AC-Name*	The Active Discovery only accepts the PPPoE session if the remote peer provides tag 'AC-Name' with its Active Discovery Offer as specified. This allows to identify the desired remote peer

Example: Configure a PPPoE session

The procedure below configures a PPPoE session for the connection to a DSL provider using the credentials specified in the PPP session profile above.

```
node(cfg)#port ethernet 0 0
node(prt-eth)[0/0]#no shutdown
node(prt-eth)[0/0]#session pppoe green
node(session)[GREEN]#bind session ppp JOE_EXAMPLE
```

Creating a PPP Profile

A PPP profile allows to adjust additional PPP parameters like the maximum transmit unit (MTU) and maximum receive unit (MRU). Only the most important parameters are listed here.

The profile *DEFAULT* is always present and supplies the parameters if no other profile has been created or a profile cannot be used with a certain type of PPP connection. Profiles created by the user can only be used with PPP over Ethernet connections. For all other types of PPP connections the default profile applies.

The procedure bellow describes how to create a PPP profile or to modify the default PPP profile.

Mode: Configure

Step	Command	Purpose
1	***node*(cfg) #[no] profile ppp { name | DEFAULT }**	Creates the new PPP profile *name* and enters the PPP profile configuration. The profile 'DEFAULT' already exists.
2 (optional)	***node*(pf-ppp)[name]#mtu min** *max*	Defines the minimum and maximum size of IP packets (in Bytes) allowed on the outbound PPP connection. Outbound packets larger than the maximum size are fragmented into smaller ones if allowed. The default value is 1492 Bytes. On the IP interface over which the PPP connection runs, the minimum of the IP interface MTU and PPP MTU applies.
3 (optional)	***node*(pf-ppp)[name]#mru** *max*	Defines the minimum and maximum size of IP packets (in Bytes) allowed on the inbound PPP connection. The default value is 1492 Bytes. Inbound packets larger than the maximum size are fragmented into smaller ones if allowed. The default value is 1492 Bytes.
4 (optional)	***node*(pf-ppp)[name]#[no] van-jacobson {compression|decompression} max-slots** *max-slots*	Allows PPP to use Van Jacobson header compression for TCP packets. Only the negotiation between the PPP peers determines whether this header compression is really used. *max-slots* determines the maximum number of concurrent TCP sessions for which header compression shall be done. The default is 31.

Example: Create a PPP profile

The procedure below creates a PPP profile, sets some of its parameters, and assigns it to a PPPoE session.

```
node(cfg)#profile ppp PPPoE
node(pf-ppp)[PPPoE]#mtu min 68 max 1492
node(pf-ppp)[PPPoE]#mru min 68 max 1492
node(pf-ppp)[PPPoE]#van-jacobson
node(pppoe)[0/0]#session ppp JOE_EXAMPLE
node(session-ppp)[JOE_EXA~]#use profile ppp PPPoE
```

Displaying PPP Configuration Information

This section shows how to display and verify the PPP configuration information.

Mode: Configure

Step	Command	Purpose
1	***node*(cfg) #show running-config**	Gives the best overview of all PPP related configuration information. The following parts of interest are: • profile ppp DEFAULT • profile ppp *name* • interface *name* • session ppp *name* • port Ethernet *slot port* • session pppoe *name*
2	***node*(cfg) #show session ppp [** *name* **]**	Displays configuration information of the PPP session *name* or of all PPP sessions.

Example: Display PPP session configuration information

```
node#show session ppp JOE_EXAMPLE

ppp JOE_EXAMPLE
===============================================
Administrative State : Up
Bound IP Interface   : ROUTER.PPP_INTERFACE
IP Addresses         : IPCP (IPCP): up
                          Requested: (none)
                        Operational: 10.67.15.1/32 peer 10.0.0.1
Bound Bridge         : (none)
Operational State    : Up
Hardware Address     : 00:00:0c:16:3b:43
MTU                  :
ARP                  : enabled
Multicast            : enabled
Rx Statistics        : 1058 bytes in 16 packets
                      0 errors 0 drops, 0 overruns
                      0 multicast packets
Tx Statistics        : 1058 bytes in 16 packets
                      0 errors 0 drops, 0 collisions 0 carrier errors
```

Debugging PPP

A set of commands is available to check the status of the PPP connection and the PPPoE session. Furthermore, two debug monitors help to analyze the dynamic behavior. The commands are listed in the order which you should follow in case you encounter problems with PPP. This procedure describes how to display PPP configuration information:

Mode: Configure

Step	Command	Purpose
1	***node*(cfg) #show pppoe** [*name*]	Displays configuration information of the PPPoE session(s). *level* specifies to level of details displayed (1..4, default is 1).
2	***node*(cfg) #show port ethernet** *slot port*	Displays status and configuration information of the Ethernet/serial port over which a PPP connection/ PPPoE sessions runs. Check whether operation state of the port is 'Up'.
3	***node*(cfg) # [no] trace ppp debug detail *level***	Enables all or a particular PPP debug monitor.
4	***node*(cfg) # [no] trace pppoe debug detail *level***	Enables all or a particular PPPoE debug monitor.

Example: Display PPPoE information

```
GREEN
===============================================
Interface            : ethernet 0 0
Service              :
Access Concentrator  :
PPP Session          : JOE_EXAMPLE
```

Sample Configurations

PPP Over Ethernet (PPPoE)

Without authentication, encapsulation multi, with NAPT

```
profile napt WAN

context ip ROUTER

  interface NORMAL_IP_INTERFACE
    ipaddress 172.16.1.1 255.255.0.0

  interface PPP_INTERFACE
    ipaddress IPCP
use profile napt WAN

context ip ROUTER
routing-table DEFAULT
route 0.0.0.0 0.0.0.0 interface PPP INTERFACE metric 0

session ppp NO AUTHENTICATION
bind interface ROUTER PPP INTERFACE
no shutdown

port ethernet 0 0
bind interface normal_ip_interface
```

```
    no shutdown

  session pppoe GEEEN
  bind session ppp NO AUTHENTICATION
```

With authentication, encapsulation PPPoE

```
  context ip ROUTER

    interface PPP_INTERFACE
      ipaddress IPCP

  session ppp JOE_EXAMPLE
  authentication pap
    identification outbound <user> password <password>
    bind interface ROUTER PPP_INTERFACE

  port ethernet 0 0
    no shutdown

      session pppoe GREEN
        bind session ppp JOE_EXAMPLE
```

Chapter 21 IP Context Overview

Chapter contents

Introduction

This chapter outlines the Trinity *Internet protocol (IP) context* and its related components. You will get the fundamental understanding on how to set up your Patton device to make use of IP related services.

The following sections describe the configuration steps necessary to put together certain IP services and the references to the related chapters that explain the issue in more details.

Trinity also supports Fast-Path routing, which can be found in Chapter 24 on page 262.

To understand the information given in the following chapters, carefully read to the end of the current chapter. Before proceeding, make sure that you feel comfortable with the underlying Trinity configuration concept by reading Chapter 2, "Configuration Concepts" on page 47.

The IP context in Trinity is a high level, conceptual entity that is responsible for all IP-related protocols and services for data and voice. The IP context performs much of the same functions as a standalone IP router, and since every context is defined by a name, the IP context is named *ROUTER* by default.

In figure 24 below, the IP context with all its related elements is contained within the area on the left, which has a gray fill (find a short description of those elements below). The right side displays the related CS context, which communicates with the IP context via gateways. Since the CS context and its related components are not the subject of this chapter, they are illustrated in figure 24 with gray lines instead of black ones.

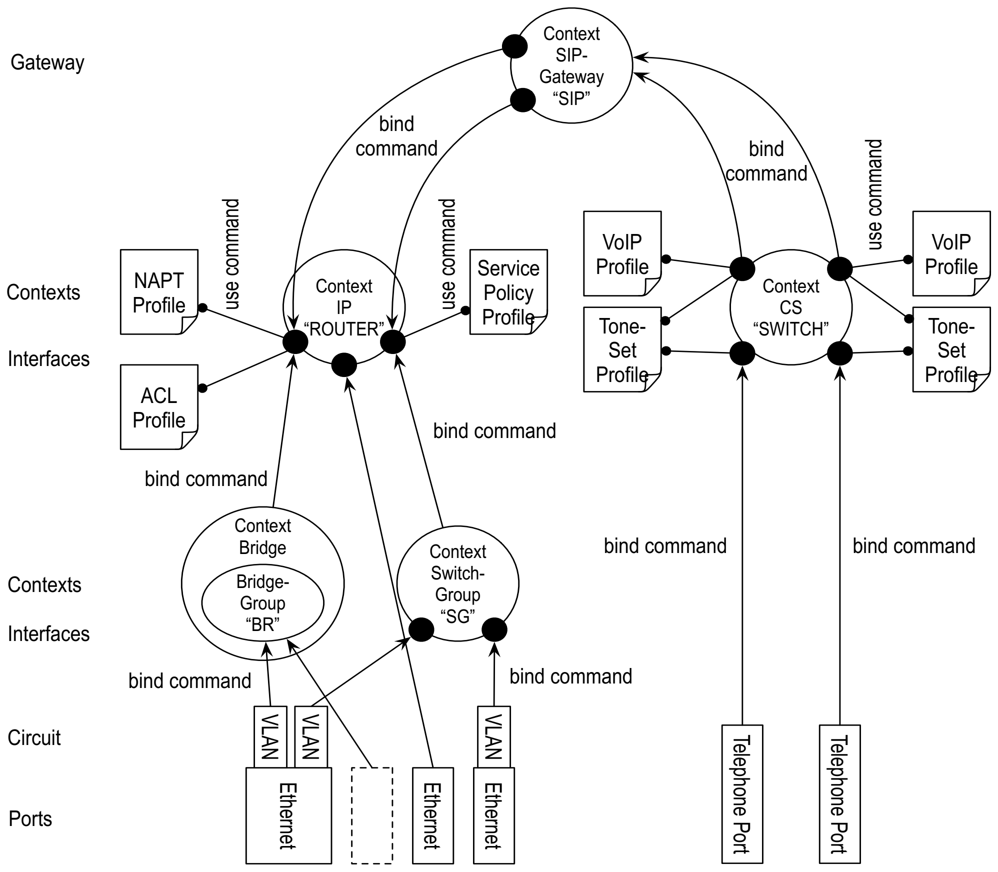


Figure 24. IP context and related elements

The IP context contains the following entities:

- Routing tables
- Logical IP interface
- Links to service profiles

Since the IP context represents a virtual IP4 and IPv6 dual-stack router, it contains up to 251 routing tables for static routes (not depicted in figure 24 on page 230). The routing tables decide whether received packets are delivered to a local application (example, CLI, web server, SIP gateway) or routed via another IP interface to a remote network host.

The IP context may contain an arbitrary number of logical interfaces. Unlike other operating systems where a network interface is identical to a physical port, we distinguish physical ports from logical interfaces. A logical interface contains all IP-related configuration parameters that are common to all ports, such as the IP address, for example. As depicted in figure 24 on page 230, a physical port or circuit is bound bottom-up to one logical IP interface. Hence, each IP interface reflects the IP-protocol of a physical port or circuit.

Applications such as SIP gateways may also be bound to an IP interface. A top-down binding defines over which IP interface (and hence over which physical port or circuit) an application communicates.

Packet Processing in the IP Context

Several IP service profiles can be assigned to the individual logical interfaces in the context (see figure 24). These profiles control the flow of packets through the router. They classify packet streams, control which packets may enter/leave the device via Access Control Lists (ACL), perform Network and Port Address Translation (NAPT) and deal with Quality-of-Service (QoS) information in packet headers.

Note that there is a different packet-processing chain for each interface depending on its configuration, i.e., each interface maintains its own configuration of how the packets are classified, a different ACL, etc. However, to make having the same configuration on multiple interfaces easier, we moved the configuration parameters to profiles. The **use** command attaches a profile to an interface, such that the same profile can be used by different interfaces.

Figure 25 shows the journey of a packet through the IP context and the order in which the attached profiles process the packet.

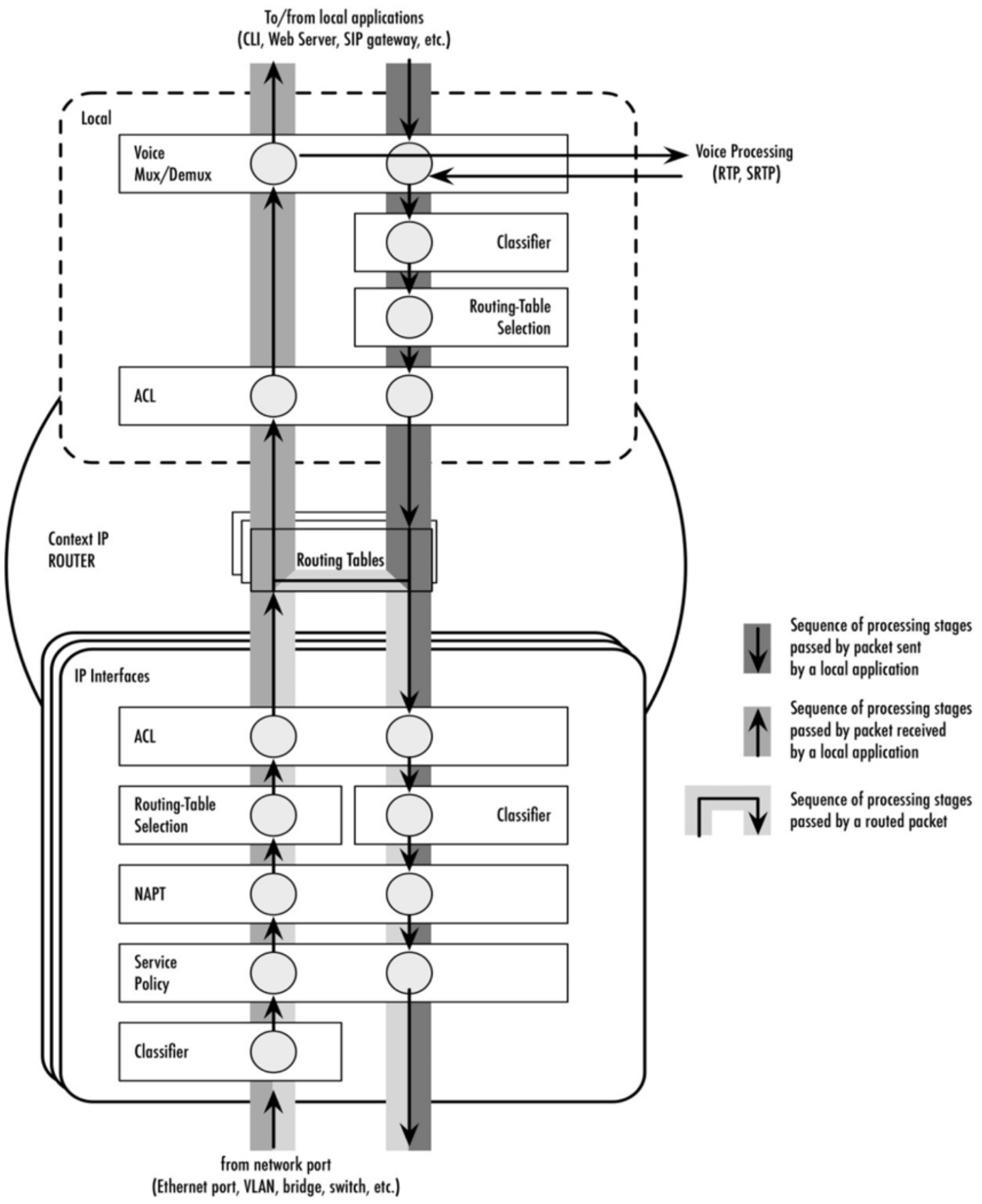

To/from local applications
(CLI, Web Server, SIP gateway, etc.)
Local
Voice Mux/Demux
Voice Processing (RTP, SRTP)
Classifier
Routing-Table Selection
ACL
Context IP ROUTER
Routing Tables
IP Interfaces
ACL
Routing-Table Selection
Classifier
NAPT
Service Policy
Classifier
from network port
(Ethernet port, VLAN, bridge, switch, etc.)
Sequence of processing stages passed by packet sent by a local application
Sequence of processing stages passed by packet received by a local application
Sequence of processing stages passed by a routed packet

Figure 25. Processing order of IP services attached to an IP interface

Classifier

The classifier is the first profile that inspects an incoming packet. The classifier assigns a traffic class to each packet. You can think of the traffic-class as if every packet in the router has a tag attached to it, on which the classification can be noted. The traffic-class tags exist only inside the router, but layer 2 priority bits (802.1pq class-of-service) and IP header type-of-service bits (TOS field) can be used to mark a specific packet type for the other network devices. By default the traffic-class tag is *DEFAULT*.

A powerful packet-matching filter in the classifier profile lets you inspect any combination of IP, UDP, TCP or ICMP header fields and assign a traffic-class to the matching packet flow. For example, you may configure to tag all UDP packets to a destination port between 5000 and 8000, and shorter than 500 bytes with the traffic-class *VOICE*. The traffic-class tag can later be used in other IP service profiles, e.g., to filter packets in the ACL or to do policy routing by selecting a routing-table based on the traffic-class.

Network Address Port Translation (NAPT)

After classification is done, the packet is handed over to the NAPT profile-if one is used on the current interface. Network Address Port Translation (NAPT), which is an extension to NAT, uses TCP/UDP ports in addition to network addresses (IP addresses) to map multiple private network addresses to a single outside address. Thus the NAPT profile may change the destination address and port of an incoming packet.

Routing-table Selection

You may configure policy routing by selecting a different routing table based on some header fields of the incoming packet. You may also use the traffic-class (tagged before in the Classifier) to make a routing-table decision. For example, you may direct all packets tagged with the VOICE traffic-class to a separate routing table while processing the other traffic with the default routing table.

Note The routing-table selection for an incoming packet is performed after NAPT, i.e., you will see the translated (private) addresses and ports

Access Control Lists (ACL)

An access control list is a sequential collection of permit and deny conditions that apply to packets on a certain interface. You can use the same packet-matching mechanism as in the classifier and the routing-table selection to decide whether the specified packet flow is permitted to enter the router or is rejected.

The ACL filter is passed after the routing decision has been made. This allows you to apply an ACL to an input-output interface pair. For example, you may use a specific profile for all packets entering the router via the LAN interface and leaving it over the DMZ interface.

Routing

Once a packet traversed all ingress packet filters (controlled by the attached profiles), the router decides whether the packet is destined to an application of the gateway itself or shall be routed to a remote host. For this purpose it performs a best-prefix match on the destination IP address in the routing-table, which was previously selected. If no routing-table has been selected explicitly, the DEFAULT table is consulted.

If the packet is to be sent to a remote host, it traverses the egress filters of the IP interface (depicted in figure 25), an egress ACL, another possibility to classify the packet, NAPT translations and finally, a service-policy profile, which can be used to map an internal traffic-class to IP TOS field values.

Packet Processing To/From Local Applications

If the packet is not sent to a remote host, and is destined for a local application (e.g. CLI, the web server, or SIP signaling packets), another set of packet-processing filters is traversed after the routing decision has been made. In particular, another ACL profile dedicated only for locally-terminated flows is passed. This allows you to create specific ACL profiles to protect the local device while having different ACL profiles for routed traffic.

After passing the ACL, voice data packets (RTP/SRTP) are diverted to the voice processing engine whereas the remaining traffic reaches one of the running service applications.

Packets that have been generated by applications on the device also traverse a set of packet-processing filters-a classifier to tag packets with a traffic-class, routing-table selection, and another outbound ACL for locally-generated traffic.

As shown at the top of figure 25 on page 232, the local packet-processing filters are not attached to a specific logical IP interface. All packets to/from a local application rather pass the same set of filters. There is a special local mode within the IP context in which classifier and ACL profiles for local applications can be attached (see chapter 34, "Classifier Configuration" on page 341 and chapter 33, "Access Control List Configuration" on page 331 for more information). The local mode also hosts routing-selection commands for locally-generated traffic (see chapter 23, "IP Routing" on page 248 for more information).

IP Context Overview Configuration Task List

As previously described, this chapter outlines the IP context configuration. It does not give you all the details of a configuration task, but refers you to the chapters in which you will find the full description.

- To view additional information for configuring an IP Interface, refer to chapter 22, "IP Interface Configuration" on page 238
- To configure a physical port and bind it to a logical IP interface, refer to chapter 14, "Ethernet Port Configuration" on page 156
- For information on configuring the classifier to tag packets with a traffic-class, refer to chapter 34, "Classifier Configuration" on page 341
- For information regarding network address port translation (NAPT), refer to chapter 25, "NAT/NAPT Configuration" on page 265
- For information on setting up the IP router contained in Trinity, refer to chapter 23, "IP Routing" on page 248
- For essential knowledge related to network security requirements, refer to chapter 33, "Access Control List Configuration" on page 331
- If your network provides better service to selected network traffic, chapter 35, "Service Policy Configuration" on page 349 will help you to get in-depth knowledge about quality of service (QoS) management.

The following sections describe the basic tasks involved in IP context configuration. Many parameters have acceptable default values, which in most cases do not need to be explicitly configured. Hence not all of the configuration tasks below are required. Depending on your application scenario, some tasks are mandatory while others are optional. The following tasks use a bottom-up approach, starting from the ports, followed by the interfaces and up to the services running on the device. Read through the tasks in order to learn a general understanding of the whole network before moving onto more detailed instructions.

- Planning your IP configuration (see page 235)
- Configuring physical ports (see page 235)
- Creating and configuring IP interfaces (see page 236)
- Configuring packet classification (see page 236)
- Configuring Network Address Port Translation (NAPT) (see page 236)
- Configuring static IP routing (see page 236)
- Configuring Access Control Lists (ACL) (see page 237)
- Configuring quality of service (see page 237)

Planning Your IP Configuration

The following subsections provide network connection considerations for Ethernet ports. Patton recommends that you draw a network overview diagram displaying all neighboring IP devices. Do not begin configuring the IP context until you have completed the planning of your IP environment.

IP Interface Related Information

Setting up the basic IP connectivity for your device requires at least the following information:

- IP addresses used for Ethernet LAN and WAN ports
- IP Subnet mask used for Ethernet LAN and WAN ports
- Length for Ethernet cables
- IP addresses of the central SIP registrar
- IP addresses of the central PSTN gateway for SIP-based calls

QoS Related Information

Check with your access service provider if there are any QoS-related requirements, which you need to know prior to configuring Trinity QoS management. Check the following with your access service provider:

- What is the dedicated bandwidth, which you have agreed with your access service provider?
- How does your provider perform packet classification, e.g. which ToS bits have to be used to define the supported classes of service?

Configuring Physical Ports

Port configuration includes parameters for the physical and data link layer, such as framing and encapsulation formats or media access control. Before any higher-layer user data can flow through a physical port, you must associate that port with an interface within the IP context. This association is referred to as a binding. For information and examples on how to configure ports, refer to the respective port type’s chapter.

Creating and Configuring IP Interfaces

The number and names of IP interfaces depend upon your application scenario. An interface is a logical construct that encapsulates network-layer protocol and service information, such as IP addressing. Therefore, interfaces are configured as part of the IP context (the virtual router) and represent logical entities that are only usable if a physical port (Ethernet) or circuit (VLAN) is bound to them.

An interface name can be any arbitrary string, but for ease of identification use self-explanatory upper-case names that describe the use of the interface, e.g. LAN, WAN.

Several IP-related configuration parameters are necessary to define the behavior of such an interface. The most obvious parameters are one or multiple IP addresses and the IP net masks that belong to them. Several profile types can also be attached to an IP interface to define how packets arriving on the interface or leaving over it are processed.

For information and examples on how to create and configure an IP interface, refer to #<IP interface configuration>#. The configuration of each profile type is described in a dedicated chapter, and is briefly introduced below.

Configuring Packet Classification

A classifier profile can be attached to each IP interface. It contains rules to match packet flows based on the header fields of the packets and tag them with an internal traffic-class. This traffic-class is usually used in conjunction with other services. For example, an ACL may have filter rules that drop all packets tagged with a certain traffic-class, or policy routing may be configured to select a dedicated routing-table for a packet flow of a given traffic-class. Trinity tests packets against the classifier rules one by one. The first match determines the traffic-class. Because Trinity stops testing rules after the first match, the order of the classifier rules is critical. If no conditions match or if there is no classifier profile attached to an interface, the software tags receive packets with the DEFAULT traffic-class, whereas all packets generated by local applications are tagged with the LOCAL-DEFAULT traffic-class, except generated RTP/SRTP packets, which are tagged as LOCAL-VOICE.

Classifier profiles can be attached to several entities in Trinity-on any local IP interface and in the local mode of the IP context. In both places classifier profiles can be attached separately for inbound and outbound packets.

For an in-depth elaboration of how to configure and use classifier profiles, refer to chapter 34, "Classifier Configuration" on page 341. To get a detailed understanding of how to build packet matching rules, consult chapter 36, "Packet Matching" on page 353.

Configuring Network Address Port Translation (NAPT)

You can configure NAPT by creating a profile that is afterwards used on an explicit IP interface. In Trinity terminology, an IP interface uses a NAPT profile, as shown in figure 24 on page 230. For information and examples on how to configure NAPT, refer to chapter 25, "NAT/NAPT Configuration" on page 265.

Configuring Static IP Routing

Trinity allows you to define static routing entries, which are destination-address-to-egress-interface mappings established by the network administrator prior to the beginning of routing. These mappings do not change unless the network administrator alerts them. Algorithms that use static routes are simple to design, and work well in environments in which network traffic is relatively predictable and where network design is relatively simple.

Routing entries are grouped in routing-tables. A set of route commands in the IP interface can be used to select the routing-table for inbound traffic for different packet-header fields. The route command in the local mode, within the IP context configures the routing-table to consult for locally-generated traffic. Trinity tests packets against the routing-table-selection rules one by one. The first match determines the routing-table to use. Because Trinity stops testing rules after the first match, the order of the routing-selection rules is critical. If no conditions match or if there is no route command in the interface, the software uses the DEFAULT routing table.

For information and examples on how to configure static IP routing, refer to #< IP routing configuration>#. To get a detailed understanding of how to build packet matching rules, consult #<Packet matcher overview>#.

Configuring Access Control Lists (ACL)

Packet filtering helps to control packet movement through the network. Such control can help to limit network traffic and restrict network use by certain users or devices. An access control list is a sequential collection of permit and deny conditions that apply to packets on a certain interface. Access control lists can be configured for all routed network protocols (IP, ICMP, TCP, UDP, and SCTP) to filter the packets of those protocols as the packets pass through a device. Trinity tests packets against the conditions in an access list one by one. The first match determines whether Trinity accepts or rejects the packet. Because Trinity stops testing conditions after the first match, the order of the conditions is critical. If no conditions match, the software rejects the address.

For information and examples on how configure access control lists, refer to chapter 33, "Access Control List Configuration" on page 331. To get a detailed understanding of how to build packet matching rules, consult chapter 36, "Packet Matching" on page 353.

Configuring Quality of Service (QoS)

A service-policy profile can be attached to an IP interface to manage QoS for network traffic, as shown in Figure 24 on page 230. QoS refers to the ability of a network to provide improved service to selected network traffic over various underlying technologies including Ethernet and 802.x type networks, as well as IP-routed networks. In particular, QoS features provide improved and more predictable network service by providing the following features:

- Supporting dedicated bandwidth
- Improving loss characteristics
- Avoiding and managing network congestion
- Shaping network traffic
- Setting traffic priorities across the network

The QoS features described in chapter 35, "Service Policy Configuration" on page 349 address these diverse and common needs.

Chapter 22 IP Interface Configuration

Chapter contents

Introduction

This chapter provides a general overview of IP interfaces and describes the tasks involved in their configuration. An interface is a logical entity that provides higher-layer protocol and service information, such as Layer 3 addressing. Interfaces are configured as part of a context and are independent of physical ports and circuits.

The separation of the interface from the physical layer allows for many advanced features. For higher layer protocols to become active, a physical port or circuit must be bound to an interface. IP interfaces can be bound physically to Ethernet or DSL ports, or to VLANs, according to the appropriate transport network layer.

IP Interface Configuration Task List

To configure interfaces, perform the tasks in the following sections:

- Creating an IP interface (see page 239)
- Deleting an IP interface (see page 240)
- Setting the static IP address (see page 241)
- Deleting an IP address (see page 241)
- Requesting an IP address via DHCP (see chapter 26, "DHCP Configuration" on page 273)
- Displaying IP interface information (see page 242)
- Testing connections with the **ping** command (see page 243)
- Traceroute command (see page 246)
- Debugging the IP configuration (see page 246)

Creating an IP Interface

Interface names can be any arbitrary string. Use self-explanatory names for your interfaces, which reflect their usage, e.g. LAN, WAN, DMZ.

Mode: configure

Step	Command	Purpose
1	***device*(cfg)#context ip [ROUTER]**	Enters the IP context configuration mode for the default virtual router.
2	***device*(ctx-ip)[ROUTER]#interface** *name*	Creates the new interface *name*, which represents an IP interface. This command also places you in interface configuration mode for the interface just created.
3	***device*(if-ip)[ROUTER.*name*]#**	You are now in the interface configuration mode, where you can enter specific configuration parameters for the IP interface *name*, e.g., create an IP address.

Example: Create IP interface

The procedure illustrated below assumes that you would like to create an IP interface named *WAN*. Use the following commands in *operator exec* mode.

```
device>enable
```

```
device#configure
device(cfg)#context ip ROUTER
device(ctx-ip)[ROUTER]#interface WAN
device(if-ip)[ROUTER.WAN]#
```

Deleting an IP Interface

Almost every configuration command has a **no** form. In general, use the **no** form to disable a feature or function. Use the command without the **no** keyword to re-enable a disabled feature or to enable a feature that is disabled by default.

Deleting an existing interface in the IP context is often necessary. You can only delete an IP interface if it is not bound from a physical port or circuit. Go to the configuration mode of the physical port or circuit and unbind the interface first before deleting it.

Mode: configure

Step	Command	Purpose
1	**device(cfg)#context ip [ROUTER]**	Enters the IP context configuration mode for the default virtual router.
2	***device*(ctx-ip)[ROUTER]#no interface** *name*	Deletes the existing interfaces *name.*

Example: Delete IP interface

The procedure below assumes that you would like to delete an IP interface named *WAN*, which is currently bound by Ethernet port 0/1. Use the following commands in *operator exec* mode to unbind and delete the IP interface.

```
device>enable
device#configure
```

Unbind port Ethernet 0/1 from the IP interface WAN:

```
device(cfg)#port ethernet 0 1
device(prt-eth)[0/1]#no bind interface
device(prt-eth)[0/1]#exit
```

List the existing interfaces:

```
device(cfg)#context ip ROUTER
device(ctx-ip)[ROUTER]#no interface <?>
LAN    Existing IP interface: LAN
WAN    Existing IP interface: WAN
```

Delete the IP interface named *WAN* with the no interface command:

```
device(ctx-ip)[ROUTER]#no interface WAN
```

List the interfaces again to check if the appropriate interface was deleted:

```
device(ctx-ip)[ROUTER]#interface <?>
<interface>New IP interface
LAN             Existing IP interface: LAN
```

Setting the Static IP Address and Network Mask

Each IP interface needs at least one IP address and an appropriate network mask to be operational. You can use the ipaddress interface configuration command to create, change or delete an IP address.

An IP interface may host more than one IP address. Each IP address is, therefore identified by a unique *label*, i.e., a non-empty string. This label must be specified as a parameter to ipaddress when creating, changing or deleting the IP address. The same label may be reused in a different IP interface.

An IP address must be unique within the same IP context. That is, you cannot configure the same IP address for the same or different IP interfaces belonging to the same virtual router. It is, however, possible to configure multiple IP addresses within the same subnet, e.g. 10.1.1.1/24 and 10.1.1.2/24.

To learn how to obtain a dynamic IP address via DHCP, see chapter 26, "DHCP Configuration" on page 273.

The **ipaddress** command offers the following options:

Parameter	Explanation
label	Name of the IP address. An IP interface may host more than one IP address. The label identifies the address when changing or deleting it. The label must be unique within the IP interface; you may reuse the same label in a different IP interface. **Note** The label parameter is optional. If you don't provide it, Trinity automatically generates a default label, which is identical to the name of the IP interface. If you specify *DHCP* as label, you create a DHCP address (see chapter 26, "DHCP Configuration" on page 273).
address	The network address and mask size in dotted-decimal format a.b.c.d/m for IPv4 or in the colon format a:b:c::x/m for IPv6. Alternatively, you may enter the network address and full network mask with two consecutive parameters, a.b.c.d e.f.g.h or a:b:c::x e:f:g::y, respectively.

Mode: configure

Step	Command	Purpose
1	***device*(cfg)#context ip [ROUTER]**	Enters the IP context configuration mode for the default virtual router.
2	***device*(ctx-ip)[ROUTER]#interface** *name*	Enters the configuration mode of an existing IP interface or creates a new IP interface.
3	***device*(ip-if)[ROUTER.*name*]#ipaddress** *label address*	Creates a new IP address and network mask on the IP interface *name* or changes the IP address and network mask of an existing IP address (identified by its label).

Example: Create a new static IP address

To create an IP address to *192.168.1.3* with network mask to *255.255.255.0* and label *MAIN* on the IP interface *LAN*, use the following commands in *administrator exec* mode.

```
device>enable
device#configure
device(cfg)#context ip ROUTER
device(ctx-ip)[ROUTER]#interface LAN
```

```
    device(ip-if)[ROUTER.LAN]#ipaddress MAIN 192.168.1.3 255.255.255.0
```

Example: Modify an existing static IP address

List the IP addresses within the current interface:

```
    device(ip-if)[ROUTER.LAN]#ipaddress <?>
    <label>    Unique label of the address within the interface
    MAIN       Existing IP address: 192.168.1.3/24
    ALT        Existing IP address: 10.0.0.1/8
```

Change the IP address MAIN to 192.168.1.4 and give it a network mask of 255.255.255.0, or a mask size of 24, respectively:

```
    device(ip-if)[ROUTER.LAN]#ipaddress MAIN 192.168.1.4/24
```

Deleting an IP Address

Since an IP interface host multiple IP address, Trinity supports to delete them with the no ipaddress command, specifying the address label as the only argument. After the IP address has been deleted, the device is no longer reachable over that address.

Mode: configure

Step	Command	Purpose
1	**device(cfg)#context ip [ROUTER]**	Enters the IP context configuration mode for the default virtual router.
2	**device(ctx-ip)[ROUTER]#interface** name	Enters the configuration mode of an existing IP interface or creates a new IP interface.
3	**device(ip-if)[ROUTER.name]#no ipaddress label**	Deletes an existing IP address.

Example: Delete an existing IP address

To delete the IP address with label *MAIN* on the IP interface *LAN*, follow the procedure below, starting in *administrator exec* mode.

```
    device>enable
    device#configure
    device(cfg)#context ip ROUTER
    device(ctx-ip)[ROUTER]#interface LAN
```

List the IP addresses within the current interface:

```
    device(ip-if)[ROUTER.LAN]#ipaddress <?>
    <label>    Unique label of the address within the interface
    MAIN       Existing IP address: 192.168.1.3/24
    ALT        Existing IP address: 10.0.0.1/8
```

Delete the IP address with label MAIN:

```
    device(ip-if)[ROUTER.LAN]#no ipaddress MAIN
```

List the IP addresses again to check if the appropriate address was detected.

```
device(ip-if)[ROUTER.LAN]#ipaddress <?>
<label>    Unique label of the address within the interface
ALT        Existing IP address: 10.0.0.1/8
```

Displaying IP Interface Information

The show ip interface command displays a list of all IP interfaces and their configured IP addresses. The command is available in operator exec mode or in any of its sub-modes (administrator exec or any configuration mode). By specifying the context and/or IP-interface parameter you can selectively display information for a single context/interface.

Parameter	Explanation
context	Name of the IP context. If the user neither specifies a context nor an interface, the command lists all interfaces in all contexts. If the user specifies a context but no interface, the command lists all interfaces in the specified context.
interface	Name of the IP interface. If the user specifies an interface, information for all interfaces in the specified or default (ROUTER) context is displayed.
detail	Level of detail of the output. If not specified, minimal information is printed in a compact form. The detail levels 2 and 3 provide more information about the status of the IP interface(s) addresses.
full-detail	Prints the maximum amount of information for the IP interface(s) and addresses (=detail level 3)
continuously	Prints an update of the information every second. Press <ctrl><c> to abort the command.

Mode: any

Step	Command	Purpose
1	**device>show ip interface [context] [interface] [detail detail | full-detail] [continuously]**	Displays information about the configuration and state of the selected IP interface(s) and addresses.

Example: Displaying compact information for all IP interfaces

The following example shows how to display information for all IP interfaces by using the show ipinterface command from *operator exec* mode.

```
device>show ip interface
IP Context: ROUTER
==================
IP Interface  Status  Address  Type    Status  Configured   Operational
---------------------------------------------------------------------------
LAN           up      LAN      static  up      10.1.1.1/24  10.1.1.1/24
                      DHCP     DHCP    up      (none)       172.168.1.10/24
WAN           up      WAN      static  up      10.1.2.1/24  10.1.2.1/24
```

The columns have the following meaning:

Column	Explanation
IP interface	Name of the IP interface
Status	State of the IP interface: • **unbound:** No physical port or circuit is bound to the IP interface • **shutdown:** The bound physical port or circuit is administratively shut down; use the `no shutdown` command on the port/circuit to bring it up. • **down:** The link of the bound physical port or circuit is down • **up:** The link of the bound physical port or circuit is up; the IP interface is ready to send/receive packets.
Address	Label of an IP address on the IP interface
Type	IP address type (static or DHCP)
Status	State of the IP address: • **down:** The IP interface is not ready, i.e., not in the up state • **FAILED:** Requesting a dynamic address or applying a static address failed; execute the command `show ip interface full-detail` to display the reason of the problem • **released:** The DHCP address is waiting for a new lease from the DHCP server • **applying:** The IP address is being applied to the system • **revoking:** The IP address is being removed from the system, e.g., prior to applying a newly configured address. • **up:** The device is reachable via the IP address.
Configured	The configured static IP address or requested DHCP address
Operational	The active IP address of the device

Example: Displaying detailed information for a specific IP interface.

The following example shows how to display detailed information for a specific IP interfaces by using the show ipinterface command from operator exec mode:

```
device>show ip interface ROUTER LAN full-detail
IP Interface: LAN
=================
  Status:                                  up
  Port/Circuit:                            ethernet 0 0 (dev: eth0)
  Active/Total IPv4 Addresses:             2/2

  Address: LAN
  ------------
    Type:                                  static
    Status:                                up
    Configured:                            172.16.46.10/19
    Operational:                           172.16.46.10/19

  Address: DHCP
```

```
    -------------
      Type:                                 DHCP
      Status:                               up
      Configured:                           (none)
      Operational:                          172.16.60.4/19
```

In addition to the compact form, the detailed version shows the bound physical port or circuit as well as the active (state=up) and total number of IP addresses on the interface.

Displaying Dynamic ARP Entries

The following command can be used to display both the statically-configured as well as the dynamically-learned ARP entries of the device.

Mode: any

Step	Command	Purpose
1	**device>show arp**	Displays the ARP entries of the system

Testing Connections with the Ping Command

As an aide to diagnosing basic network connectivity, many network protocols support an echo protocol. The protocol involves sending a special ICMP datagram to the destination host, then waiting for a reply datagram from that host. Results from this echo protocol can help in evaluating the path-to-host reliability, delays over the path and whether the host can be accessed or is functioning.

Mode: any

Step	Command	Purpose
1	**device>ping address [count]**	Sends *count* ICMP ECHO_REQUEST packets to the network host at IP address *address*. If the *count* parameter is not specified, five requests are sent. The user may abort the command by pressing <ctrl><c>.

When using **ping** for fault isolation, you should first run it on the respective local IP interface to verify that the local LAN or WAN interface is up and running. Then, you should “ping” hosts and gateways farther away. The command computes round-trip times and packet loss statistics when it terminates. If duplicate packets are received, they are not included in the packet loss calculation, although the round-trip time of these packets is used to calculate the minimum/average/maximum round-trip time numbers. When the command terminates, a brief summary of the calculation is displayed.

Example: Testing connections with the **ping** command

The following example shows how to invoke the echo protocol to the destination host at IP address 172.16.1.10 by using the **ping** command from *operator exec* mode.

```
device>ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=45 time=15.019 ms
64 bytes from 8.8.8.8: seq=1 ttl=45 time=52.068 ms
64 bytes from 8.8.8.8: seq=2 ttl=45 time=83.353 ms
64 bytes from 8.8.8.8: seq=3 ttl=45 time=92.690 ms
```

```
64 bytes from 8.8.8.8: seq=4 ttl=45 time=32.056 ms
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 15.019/55.037/92.690 ms
```

Traceroute Command

The traceroute command allows you to determine the path a packet takes in order to get to a destination from a given source by returning the sequence of hops the packet has traversed.

Mode: operator

Step	Command	Purpose
1	**>traceroute <host> [destination-port <port>] [max-ttl <hops>] [first-ttl <hops>] [packet-size <bytes>] [probe-count <probes>] [source-address <ipaddress>] [timeout <seconds>] [verbose]**	Find the path a packet with a size of bytes takes to a host. Traceroute will list all hops after first-ttl until max-ttl and it will stop if a hop does not answer after timeout. For each probe a packet will be sent.

Debugging the IP Configuration

When a configured IP address cannot be applied to the system, use the show ip interface full-detail command to inspect the reason. The debug ip command may also be helpful to obtain a trace of all internal actions while the IP interface is re-configured. In addition, the debug dynif command traces link state changes of physical port and circuits.

Mode: any

Step	Command	Purpose
1	**device>[no] debug ip**	Enables or disables the IP debug monitor
2	**device>[no] debug dynif**	Enables or disables the link state debug monitor

Example: Debug output while reconfiguring an IP address

The following example switches on the IP debug monitor and then changes the IP address with label WAN on IP interface WAN from 10.1.1.1/24 to 10.1.1.2/24.

```
device>debug ip
device>configure
device#context ip ROUTER
device(ctx-ip)[ROUTER]#interface WAN
device(ip-if)[ROUTER.WAN]#ipaddress WAN 20.1.1.2/24
04:40:24  IP  # [DBG] [ROUTER.WAN.WAN] onUpdate: cfgAddress
04:40:24  IP  # [INF] [ROUTER.WAN.WAN] State: up (normal) | Event: configuration
update
04:40:24  IP  # [INF] [ROUTER.WAN.WAN] State: up (normal) | Action: revoke address
                      20.1.1.1/24 from dev eth1.1
04:40:24  IP  # [INF] [eth1.1] Kernel << ip addr del 20.1.1.1/24 dev eth1.1
04:40:24  IP  # [INF] [eth1.1] Kernel >> Address departed: WAN (20.1.1.1/24)
04:40:24  IP  # [INF] [ROUTER.WAN.WAN] State: up (normal) -> revoking (reconfigure)
04:40:24  IP  # [INF] [ROUTER.WAN.WAN] State: up (normal) | Action: decrement
active
```

```
                      address counter
04:40:24  IP  # [DBG] [ROUTER.WAN] onUpdate: activeV4AddrCount
04:40:24  IP  # [INF] [ROUTER.WAN.WAN] State: revoking (reconfigure) | Event: down
04:40:24  IP  # [INF] [ROUTER.WAN.WAN] State: revoking (reconfigure) | Action: 
restart
                      (reconfigure)
04:40:24  IP  # [INF] [ROUTER.WAN.WAN] Restart Mode: reconfigure -> normal
04:40:24  IP  # [INF] [ROUTER.WAN.WAN] State: revoking (normal) | Action: apply 
address
                      20.1.1.2/24 to dev eth1.1
04:40:24  IP  # [INF] [eth1.1] Kernel << ip addr add 20.1.1.2/24 broadcast + label
                      eth1.1:WAN dev eth1.1
04:40:24  IP  # [INF] [ROUTER.WAN.WAN] State: revoking (normal) -> applying
04:40:24  IP  # [DBG] [ROUTER.WAN.WAN] onUpdate: address, restartMode, state
04:40:24  IP  # [INF] [eth1.1] Kernel >> Address arrived: WAN (20.1.1.2/24)
04:40:24  IP  # [INF] [ROUTER.WAN.WAN] State: applying (normal) | Event: up
04:40:24  IP  # [INF] [ROUTER.WAN.WAN] State: applying (normal) -> up
04:40:24  IP  # [INF] [ROUTER.WAN.WAN] State: applying (normal) | Action: increment 
active
                      address counter
04:40:24  IP  # [DBG] [ROUTER.WAN.WAN] onUpdate: state
04:40:24  IP  # [DBG] [ROUTER.WAN] onUpdate: activeV4AddrCount
```

Chapter 23 IP Routing

Chapter contents

Introduction

The Trinity IP Routing facility consists of the two major functionalities *Basic Routing* and *Policy Routing*.

Basic Routing

Under Basic Routing is to be understood the destination IP address based next-hop determination. The next-hop or gateway selection is done by matching a set of routing rules entered by the user (static-route), received through a routing-protocol (dynamic-route) or added by the system (system-route). Routing entries which specify a *gateway* as next-hop are also called **gateway-routes**. Networks that are directly reachable through a device's network-port are specified through **interface-routes**. Instead of a gateway they specify an outgoing interface.

In the *context ip* configuration mode exists a system pre-created routing-table called *DEFAULT*. This table contains all *Basic Routing* information and cannot be deleted by the user. Actually it is possible to created additional routing-tables with a user defined name but such user-created tables are part of the *Policy Routing* and do not have any use in *Basic Routing*.

All Basic Routing features are available for IPv4 as well as for IPv6.

Static Routes

These are user managed gateway and interface routes and are getting exported in the running-config. In the output of the *show route* command they are flagged with an '**R**'. Another flag '**U'** indicates if the route is **up** or not. A static gateway-route is becoming active (up) if the gateway is reachable. For this we need the following conditions:

- At least one IP address in the gateway's network has to be configured.
- The IP interface which owns the IP address has to be bound from a network-port.
- The network-port's link state has to be **up**.

A static interface-route is becoming active (up) if:

- The specified outgoing interface is bound from a network-port.
- The specified outgoing interface has at least one IP address configured.
- The network-port's link state has to be **up**.

Configuring static routes

A route is clearly identified by its destination address/mask combination and the metric. That means it is allowed to configure several time the same destination, using the same or different gateways, but with a different metric value. The metric in a static route has the meaning of a priority where lower value means higher priority.

Static route differentiation by metric is useful if a destination network is reachable through different gateways. Usually gateways are located in the same network as the device itself. If the link to the gateway with lowest metric is going down, this static-route is becoming unavailable. In that case the device's router will select the route to the destination with the next higher metric and another gateway is going to be used.

Mode: configure

Step	Command	Purpose
1	**[device](cfg)#context ip [ROUTER]**	Enters the context IP ROUTER configuration mode.
2	**[node](ctx-ip)[ROUTER]#routing-table [DEFAULT]**	Enters the routing-table DEFAULT configuration mode.
3	**[node][ROUTER.DEFAULT]#route { <network>/<mask-size> | <network> <mask> | default | default-v6 } { gateway <gw-address> | interface <if-name> } [metric <metric>]** **OR** **[node][ROUTER.DEFAULT]#no route <network>/<mask-size> [metric <metric>]**	Adds a static route. Removes a static route.

Syntax:

Parameter	Explanation
network	The destination network address in the dot-format **a.b.c.d** for IPv4 and in the colon-format **a:b:c::x** for IPv6.
mask-size	Number of mask-bits defining the destination network.
mask	The destination network mask in the dot-format **a.b.c.d** for IPv4 and in the colon-format **a:b:c::x** for IPv6.
default	Short form for defining a default IPv4 route. It configures **network/mask-size** with **0.0.0.0/0**.
default-v6	Short form for defining a default IPv6 route. It configures **network/mask-size** with **::/0.**
gw-address	The address of the next-hop router that can access the destination network. In the dot-format **a.b.c.d** for IPv4 and in the colon-format **a:b:c::x** for IPv6.
interface	The name of the outgoing interface to be used for reaching the destination network.
metric	Metric value of the route. Default: 0

System Routes

For each assigned IP address the system automatically creates route entries for the belonging network into the *DEFAULT* routing-table. That means, all directly available networks are known by the system and don't have to be configured. The system-routes are of type *interface-route* means, only the outgoing interface is specified and do not have the gateway parameter. In the output of the show route command they are flagged with an '**S**'.

Dynamic Routes

This kind of routes is assigned to the system either by a routing-protocol (RIP, BGP …) or through a device configuration protocol (DHCP, PPP …). In the output of the *show route* command they are flagged with a '**D**'. A *dynamic-route* is active under the same condition as a *static-route*.

Show Routes

Execution of *show running-config* command only displays the static-routes which have been added to the system. Neither dynamic-routes nor system-routes are shown there. To get an overview of all routes actually known by the system the *show route* command has to be executed.

Mode: Operator execution

Step	Command	Purpose
1	**[node]#show route [<detail>]**	Displays route information Default: 0

Output:

```
Routing Tables
===============================================
Flags: C - dhCp, D - Dynamic, G - use Gateway, H - target is a host
       R - useR, U - route is Up, S - System

Routing Table DEFAULT, ID = 254
Destination         Gateway             Flags Metric Interface     Source
172.16.32.0/19                          SU    0      WAN           172.16.45.7
30.30.30.0/24       172.16.45.4         RU    0
172.16.32.0/19                          SU    0      WAN           172.16.60.7
0.0.0.0/0           172.16.32.1         CDGU  0                    eth0
```

Basic Static Routing Example

The picture below shows an Internetwork consisting of three routers, a Trinity device in the middle, and the four autonomous networks, with network addresses 10.1.5.0/16, 172.16.40.0/24, 172.17.100.0/24 and 10.2.5.0/16. The Trinity device shall be configured for the following IP routing scenario:

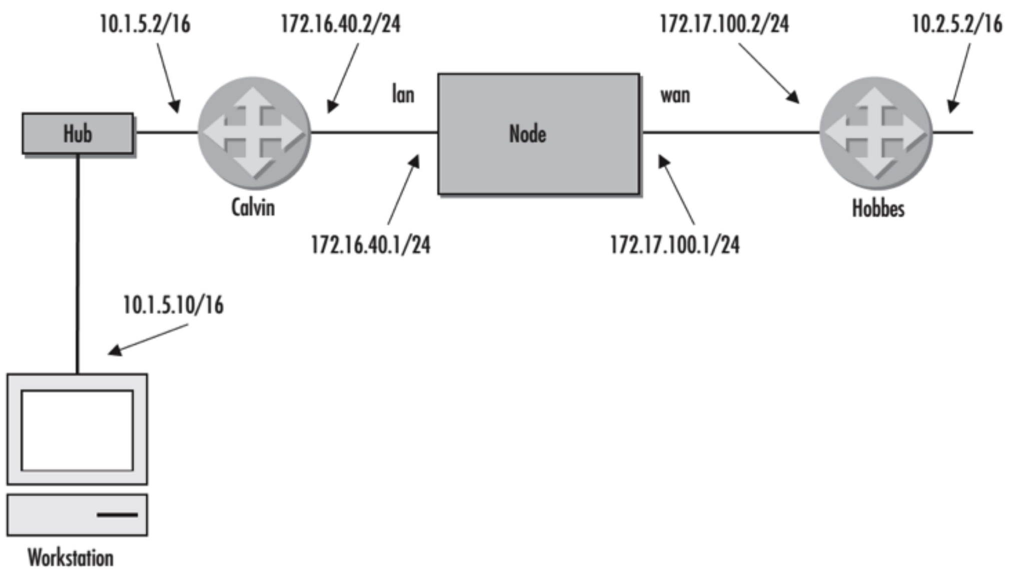


Figure 26. Static route example

All packets for the Workstation with IP address 10.1.5.10 shall be forwarded to the next-hop router Calvin. All packets for network 10.2.5.0/16 shall be forwarded to the next-hop router Hobbes.

Example Configuration:

```
context ip ROUTER

  routing-table DEFAULT
    route 10.1.5.10/32 gateway 172.16.40.2 metric 0
    route 10.2.0.0/16 gateway 172.17.100.2 metric 0
```

Show Route Output:

```
Routing Tables
===============================================
Flags: C - dhCp, D - Dynamic, G - use Gateway, H - target is a host
       R - useR, U - route is Up, S - System

Routing Table DEFAULT, ID = 254
Destination         Gateway             Flags Metric Interface     Source
172.16.40.0/24                          SU    0      LAN           172.16.40.1
172.17.100.0/24                         SU    0      WAN           172.17.100.1
10.1.5.10/32        172.16.40.2         RU    0
10.2.0.0/16         172.17.100.2        RU    0
```

Policy Routing

IP routing makes decisions based on IP addresses. Policy Routing allows the user to configure IP routing based on more criteria than only the destination IP address.

The heart of the Trinity policy-routing is the ability of traffic assignment to different routing-tables based on packet-matching at two different observation points. By making use of Trinity's Packet Matcher functionality it is possible to select traffic by more than 20 different criteria in an 'and' conjunction and to assign it to one of up to 251 user defined routing-tables. Several matching rules can be added in a prioritized order.

User defined routing-tables have the same routing features as the system created *DEFAULT* table has. Forwarding is done by destination IP address based next-hop determination (See "Basic Routing" on page 249). No additional functionality is needed due to assumption the traffic has been separated through packet-matching before it reaches the routing-table.

Observation points for Policy Routing's packet-matching are the IP interfaces and the *local* pseudo interface in context IP.

Figure 27 on page 253 shows an example of a context IP configuration with the Observation Points at which traffic from local applications and from LAN is dispatched to different routing-tables. Each routing-table is then forwarding the packets according to its routing-entries.

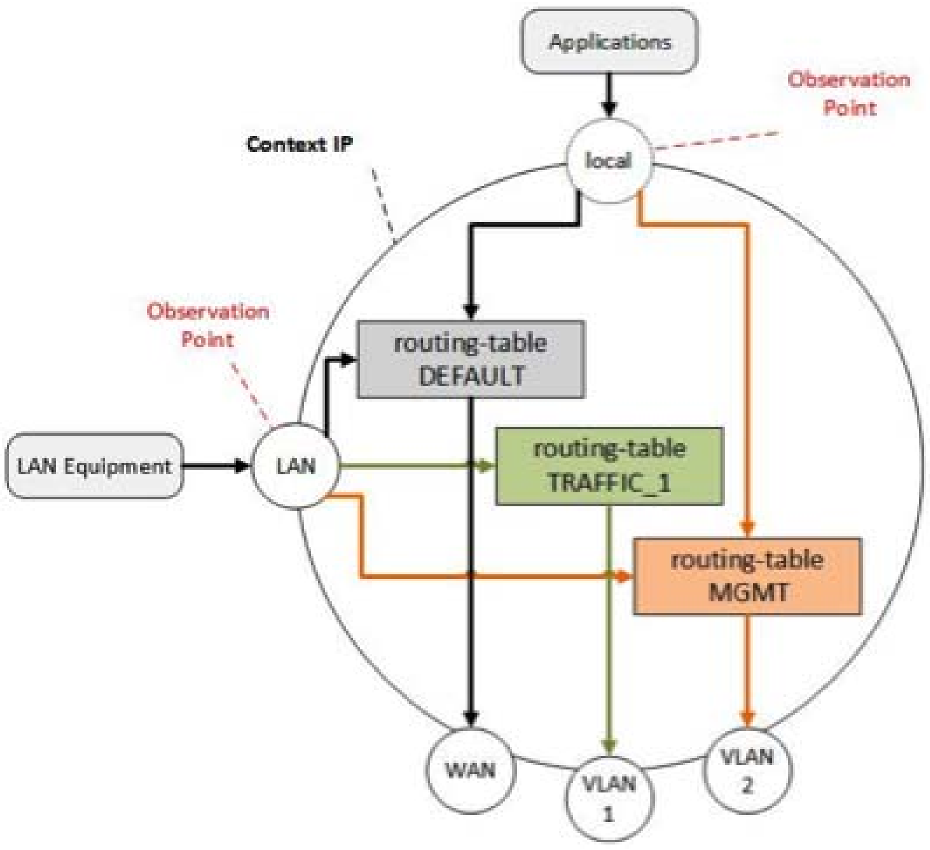


Figure 27. Policy-routing observation points

The different elements in the Policy-Routing Observation Points are:

- local: Pseudo interface in context IP where local application traffic can be matched.
- LAN: IP interface bound to LAN's network port.
- WAN, VLAN1, VLAN2: IP interfaces bound to the WAN's network and VLAN ports.
- DEFAULT: System created routing-table forwards all unspecified traffic.
- TRAFFIC_1, TRAFFIC_2: User-Created routing-tables forwarding a given kind of traffic.

Routing Tables

Besides the system-created *DEFAULT* routing-table it is possible to create up to 251 additional tables. Each user created routing-table has the same possibilities and behavior as the *DEFAULT* table (see also "Basic Routing" on page 249). The difference is their responsibility. User-Created tables are not involved in the routing process as long as there is not explicit traffic assigned to them (see "Traffic Assignment" on page 254). On the other hand the *DEFAULT* table is responsible for all traffic that is not explicitly assigned to another routing-table.

On creation of a new routing-table all system-routes (see "System Routes" on page 250) of all available IP addresses and all dynamic-routes (see "Dynamic Routes" on page 250) are automatically assigned to the table.

Also upcoming system- and dynamic-routes are replicated to already existing routing-tables. Therefore, user-created tables know the same base networks as the *DEFAULT* table.

Creating a table

Mode: configure

Step	Command	Purpose
1	**[device](cfg)#context ip [ROUTER]**	Enters the context ip ROUTER configuration mode.
2	[node](ctx-ip)[ROUTER]#routing-table <name> OR [node](ctx-ip)[ROUTER]#no routing-table <name>	Creates a new routing-table if it doesn't yet exist or enters configuration mode if it already exists. Removes a routing-table.

Configuring static routes

Because the static-route configuration syntax is exactly the same as for the DEFAULT table, please take a look at Configuring static routes.

Show routes

Because the syntax to display all available routes and the output is the same as for the DEFAULT table, please take a look at Show routes.

Traffic Assignment

As mentioned in the introduction, there exists two observation points for assigning traffic to a routing-table. Each IP interface of context IP and the local pseudo interface can be configured to route traffic according to specified criteria to a specific routing table.

All unmatched traffic is automatically sent to the routing-table *DEFAULT.* This is also the default behavior if the user doesn't care about Policy Routing.

In the *interface ip* or the *local* configuration mode explicit rules can be defined to route a specific type of traffic to a routing-table.

Assign an IP Interface

This is the simplest form for assigning traffic to a specific routing-table. All traffic received from an ip interface is sent to a configured routing-table. As visible in figure 28 such a configuration makes sense if the traffic on an interface is just of one category and is accessing always the same domain.

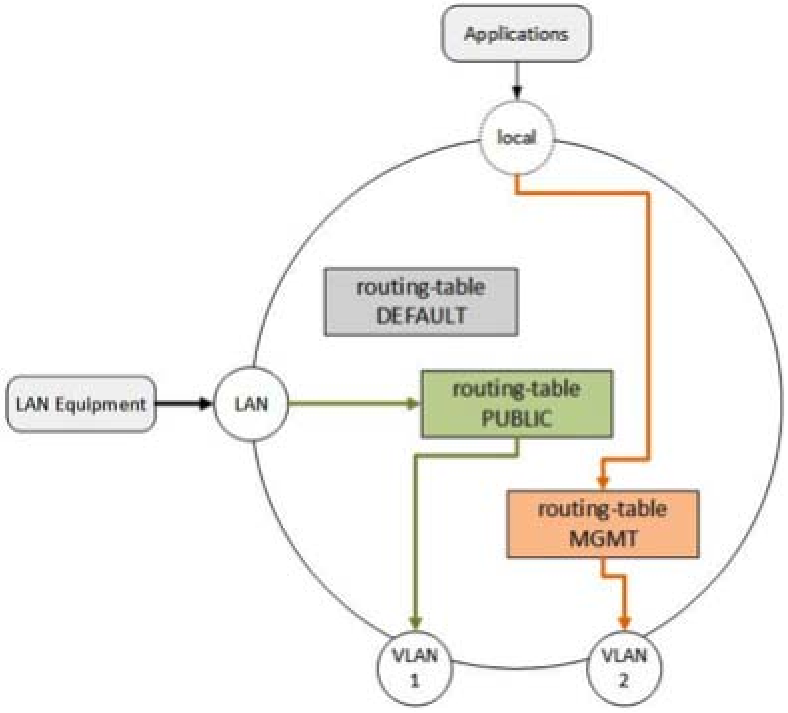


Figure 28. Whole ip interface assignment

Configuration Syntax

Mode: configure

Step	Command	Purpose
1	**[device](cfg)#context ip [ROUTER]**	Enters the context ip ROUTER configuration mode.
2	**[device](ctx-ip)[ROUTER]#interface** *<if-name>* OR **[device](ctx-ip)[ROUTER]#local**	Enters the ip interface configuration mode Enters the local configuration mode.
3	**[device](ctx-ip)[ROUTER.name]#route dest-table** *<table-name>*	Assigns the whole traffic to a routing-table.

Example Configuration:

```
context ip ROUTER

  local
    route 1 dest-table MGMT

  interface LAN
    ipaddress LAN 10.10.10.1/24
    route 1 dest-table PUBLIC

  interface VLAN1
    ipaddress VLAN1 192.168.100.1/24

  interface VLAN2
    ipaddress VLAN2 192.168.200.1/24

  routing-table DEFAULT

  routing-table PUBLIC
    route 0.0.0.0/0 gateway 192.168.100.2 metric 0

  routing-table MGMT
    route 0.0.0.0/0 gateway 192.168.200.2 metric 0
```

Assignment by Rules

With Trinity's Packet-Matcher functionality it is possible to create user-defined rules to define which traffic has to be sent to which routing-table. The entered rules have a prioritized order where lower order means higher priority. Actually the rules are applied from top to down as they are listed in *show running-config*.

If a packet doesn't match a rule's criteria the next rule is applied to the packet. If a rule matches the packet it is sent to the specified routing-table and the matching process stops. If a packet doesn't match any rule it is forwarded to the *DEFAULT* table.

In figure 29 on page 257 a scenario is shown where traffic is separated into three different categories. All VoIP packets (SIP, RTP) either coming from LAN clients or generated locally have to be handled by routing-table *VOICE*. For managing the device itself a further separation is done through the *MGMT* table. Finally, all traffic from the LAN that doesn't belong to the VoIP category is assumed to access the public internet and is sent to the *PUBLIC* table. Take a look at Example Configuration which is also referring to figure 29.

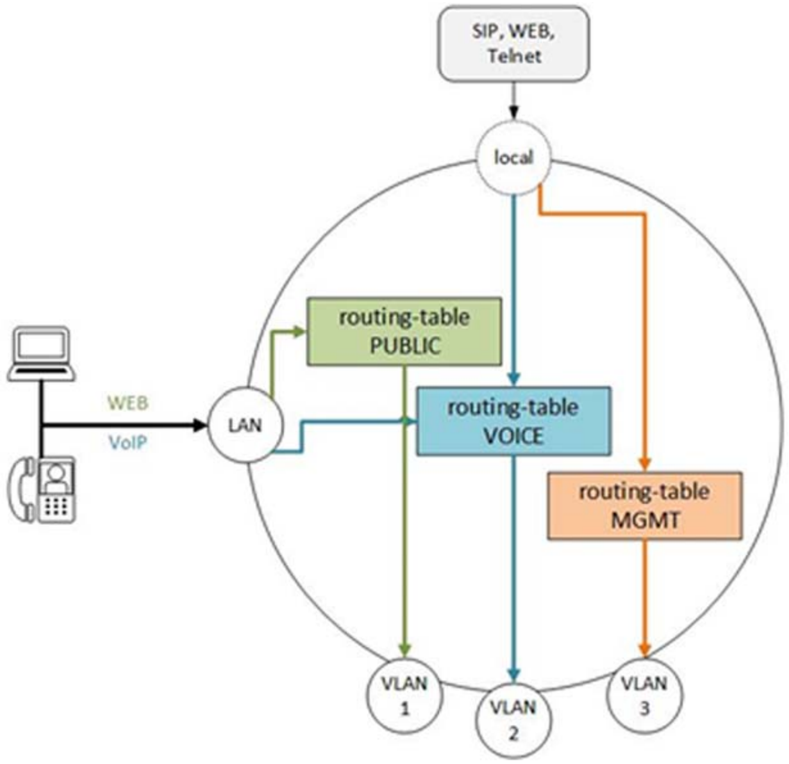


Figure 29. Explicit traffic assignment

Configuration Syntax

For all packet-matcher-options please consult chapter 36, "Packet Matching" on page 353.

Mode: configure

Step	Command	Purpose
1	**[node](cfg)context ip [ROUTER]**	Enters the context ip ROUTER configuration mode.
2	**[device](ctx-ip)[ROUTER]#interface** *<if-name>* OR **[device](ctx-ip)[ROUTER]#local**	Enters the ip interface configuration mode Enters the local configuration mode.
3	[node](if-ip)[ROUTER.name]#route [<index> | { after | before <index> }] <packet-matcher-options> dest-table <table-name> [node](if-ip)[ROUTER.name]#no route <index>	Adds a new routing rule or is moving an existing rule. Deletes the rule at *index*.

Example Configuration

The configuration below is referring to figure 29 on page 257. In the local configuration SIP signaling originated with UDP or TCP is sent to the *VOICE* table. Local generated RTP traffic is expected from the source ports 4864 to 5375 and is also sent to table *VOICE*. All traffic from the WEB server (Port 80) and the Telnet server (Port 23) is sent to table *MGMT*.

SIP traffic generated on LAN is expected to be handled the same as local generated SIP traffic. Therefore also SIP signaling (Port 5060) is sent to *VOICE*. On LAN the RTP packets are expected in the UDP source port range of 4000 to 4099. The last rule in the interface LAN configuration forces all traffic that didn't match any rules before to be sent to *PUBLIC* table.

```
context ip ROUTER

  local
    route 1 protocol udp src-port 5060 dest-table VOICE
    route 2 protocol tcp src-port 5060 dest-table VOICE
    route 3 protocol udp src-port 4864..5375 dest-table VOICE
    route 4 protocol tcp src-port 80 dest-table MGMT
    route 5 protocol tcp src-port 23 dest-table MGMT

  interface LAN
    ipaddress LAN 10.10.10.1/24
    route 1 protocol udp src-port 5060 dest-table VOICE
    route 2 protocol tcp src-port 5060 dest-table VOICE
    route 3 protocol udp src-port 4000..4099 dest-table VOICE
    route 4 dest-table PUBLIC

  interface VLAN1
    ipaddress VLAN1 192.168.100.1/24

  interface VLAN2
    ipaddress VLAN2 192.168.200.1/24

  interface VLAN3
    ipaddress VLAN3 192.168.220.1/24

  routing-table DEFAULT

  routing-table PUBLIC
    route 0.0.0.0/0 gateway 192.168.100.2 metric 0

  routing-table MGMT
    route 0.0.0.0/0 gateway 192.168.200.2 metric 0

  routing-table VOICE
    route 0.0.0.0/0 gateway 192.168.220.2 metric 0
```

Assignment by Traffic-Class

In section "Assignment by Rules" on page 256 is explained how to categorize traffic in ip *interface* or in *local* for sending it to a specific routing-table. There the categorization is explicitly done for policy-routing. Because other services probably are also assigned to a specific type of traffic it makes sense to categorize traffic only once and all service can operate on that classification. Trinity provides that feature in framework of chapter 34, "Classifier Configuration" on page 341. With the Classifier it is possible to tag packets with a *traffic-class* in an early stage of the system's packet receive path. Tagging the packets is done through chapter 36, "Packet Matching" on page 353 rules which are configured in a *Classifier Profile*.

The system knows three pre-created and assigned traffic-classes which can directly be used in the IP interface and local route rules.

- local-voice: All locally generated real-time traffic (RTP, SRTP, T.38 etc.)
- local-default: All locally generated application traffic (WEB, SIP, Telnet etc.)
- default: All other traffic (Routed traffic, System messages like ICMP-Redirect etc.)

To enable classifier profile it has to be attached to the interfaces (IP & local) where the packets to be marked are entering or leaving the system. Attaching the profile is done through the use *profile classifier command*. This command has a direction attribute which can be set to *in* or *out*. To use a classifier profile for policy-routing only one direction makes sense. Dependent on the use location it is different:

- IP interface: *in*
- Local: *out*

In figure 30 the classifier directions are shown with the in/out labeled arrows. Because *local* applications are part of the device, their generated traffic is sent *out* and needs to be marked in direction *out* for policy-routing. On the ip interface the traffic that is coming *in* the device has to be marked for policy-routing.

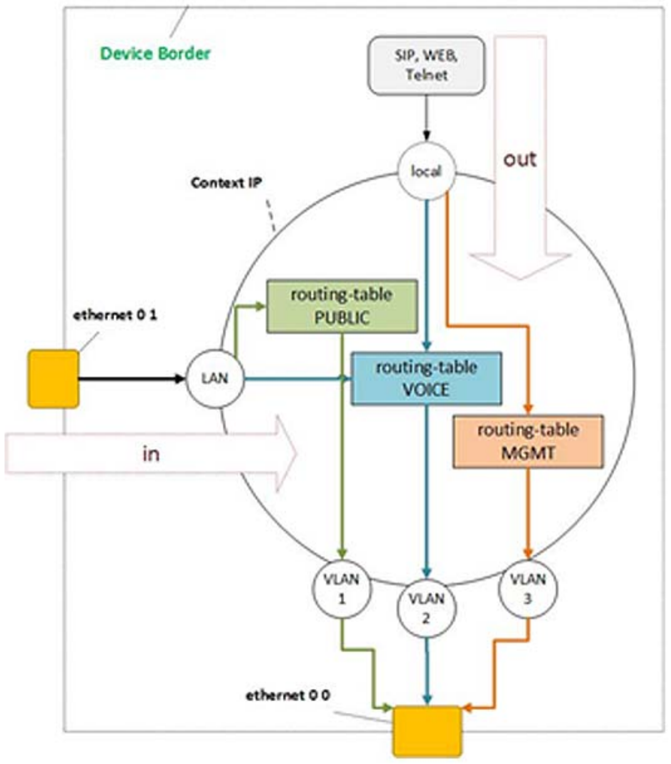


Figure 30. Classifier Directions

Configuration Syntax

For the configuration syntax of a classifier profile, see chapter 34, “Classifier Configuration” on page 341.

Mode: configure

Step	Command	Purpose
1	**[node](cfg)#context ip [ROUTER]**	Enters the context ip ROUTER configuration mode.
2	**[node](ctx-ip)[ROUTER]#interface <if-name>** **OR** **[node](ctx-ip)[ROUTER]#local**	Enters the ip interface configuration mode. Enters the local configuration mode.
3	[node](if-ip)[ROUTER.name]#use profile classifier { in | out } [<index> | { after | before <index> }] <profile-name>	Adds a new classifier-profile or is moving an existing one.
	[node](if-ip)[ROUTER.name]#no use profile classifier { in | out } <index>	Deletes the classifier-profile at *index*.

Example Configuration

The following example is referring to scenario figure 29 on page 257. In difference to the previous Example Configuration the same scenario is implemented with the system's known traffic-classes and with classifier profile that is tagging another traffic-class.

One special thing is the processing of the SIP messages. Per default all traffic from local applications is tagged with *local-default*. But because we want to send that kind of traffic to the *VOICE* routing-table it is re-tagged as *sip-sig* through the used classifier profile. The same classifier profile is the re-usable in *local* as well as in LAN interface.

```
profile classifier CL_MGMT
  match 1 protocol tcp src-port 80 set traffic-class TC_MGMT
  match 2 protocol tcp src-port 23 set traffic-class TC_MGMT

profile classifier CL_SIP_SIG
  match 1 protocol udp src-port 5060 set traffic-class TC_SIP_SIG
  match 2 protocol tcp src-port 5060 set traffic-class TC_SIP_SIG

context ip ROUTER

  local
    use profile classifier out 1 CL_SIP_SIG
    use profile classifier out 2 CL_MGMT
    route 1 traffic-class local-voice dest-table VOICE
    route 2 traffic-class TC_SIP_SIG dest-table VOICE
    route 3 traffic-class TC_MGMT dest-table MGMT

  interface LAN
    ipaddress LAN 10.10.10.1/24
    use profile classifier in 1 CL_SIP_SIG
    route 1 traffic-class TC_SIP_SIG dest-table VOICE
    route 2 protocol udp src-port 4000..4099 dest-table VOICE
    route 3 dest-table PUBLIC

  interface VLAN1
    ipaddress VLAN1 192.168.100.1/24

  interface VLAN2
    ipaddress VLAN2 192.168.200.1/24

  interface VLAN3
    ipaddress VLAN3 192.168.220.1/24

  routing-table DEFAULT

  routing-table PUBLIC
    route 0.0.0.0/0 gateway 192.168.100.2 metric 0

  routing-table MGMT
    route 0.0.0.0/0 gateway 192.168.200.2 metric 0

  routing-table VOICE
    route 0.0.0.0/0 gateway 192.168.220.2 metric 0
```

Chapter 24 Fast-Path

Chapter contents

Introduction

Fast Path is a new feature to speed-up the routing performance of Trinity devices. Fast Path only routes the first packet of a UDP/TCP flow and learns to which interface it is routed and what packet header manipulations are being performed (e.g. by ACL, NAT, etc.). All subsequent packets of the same layer 4 flow bypass the router and take the fast path.

This is currently an experimental feature and therefore, the routing fast path is disabled by default. If you experience routing performance limitations you can switch it on with the CLI commands described below.

Note Currently, the fast path feature only works for UDP and TCP flows over IPv4. All other protocols (IPv6 or other transport protocols) always take the (slower) routing path.

Fast-Path Configuration

To enable/disable the routing fast path, enter the following command(s):

Mode: configure

Step	Command	Purpose
1	**node(cfg)# context ip ROUTER**	Enter the IP context configuration mode.
2	**node(ctx-ip)[Device]# [no] fast-path**	Enables/Disables? fast-path processing for the IPv4 router.

Optionally—usually only when requested by our technical supporters—you may tweak the routing fast path by entering one or multiple of the following commands:

Mode: configure

Step	Command	Purpose
1	**node(cfg)# context ip ROUTER**	Enter the IP context configuration mode.
2	**node(ctx-ip)[Device]# fast-path**	Enter the configuration mode of the routing fast path.
3	**node(fp)[Device]# flow idle-lifetime [<interval>]**	Enter the time interval in ms after which idle UDP/TCP flows are considered closed. Since only 1024 UDP and 1024 TCP can be tracked simultaneously by the fast path, this interval should be short. On the other hand, it must be long enough to keep UDP flows that are rarely sending packets open (TCP flows are closed when routing the FIN packet). The default idle-lifetime interval is 5 minutes (300000 ms).

Step	Command	Purpose
4	**node(fp)[Device]# flow route-interval [<interval>]**	Enter the time interval in ms after which a packet of a UDP/TCP connection again passes the router too keep the stateful firewall and NAPT happy. The default route interval is 30 seconds (30000 ms).
5	**node(fp)[Device]# garbage-collection interval [<interval>]**	Enter the time interval in ms after which the fast path destroys idle flows and makes memory available for new connections. The default value is 10 seconds (10000 ms).

Chapter 25 NAT/NAPT Configuration

Chapter contents

Introduction

This chapter provides a general overview of Network Address (Port) Translation and describes the tasks involved in its configuration.

For further information about the functionality of Network Address Translation (NAT) and Network Address Port Translation (NAPT), consult the RFCs 1631 and 3022. This chapter applies the terminology defined in RFC 2663.

Trinity provides four types of NAT/NAPT:

- Dynamic NAPT (Cisco terminology: NAT Overload)
- Static NAPT (Cisco terminology: Port Static NAT)
- Dynamic NAT
- Static NAT

You can combine these types of NAT/NAPT without any restriction. One type of profile, the 'NAPT Profile', holds the configuration information for all four types where configuration is required. The remainder of this Section shortly explains the behavior of the different NAT/NAPT types.

Dynamic NAPT

Dynamic NAPT is the default behavior of the NAT/NAPT component. It allows hosts on the local network to access any host on the global network by using the global interface address as source address. It modifies not only the source address, but also the source port, so that it can tell different connections apart (NAPT source ports are in the range 8,000 to 16,000). UDP and TCP connections from the local to the global network trigger the creation of a dynamic NAPT entry for the reverse path. If a connection is idle for some time (UDP: 2 minutes, TCP: 12 hours) or gets closed (only TCP), the dynamic NAPT entry is removed.

An enhancement of the Dynamic NAPT allows to define subsets of hosts on the local network that shall use different global addresses. Up to 20 subsets with their respective global addresses are possible. Such a global NAPT address can be any IP address as long as the global network routes the traffic to the global interface of the NAT/NAPT component.

Figure 31 illustrates the basic and enhanced behavior of the Dynamic NAPT. The big arrows indicate the direction of the connection establishment. Although only a local host can establish a connection, traffic always flows in both directions.

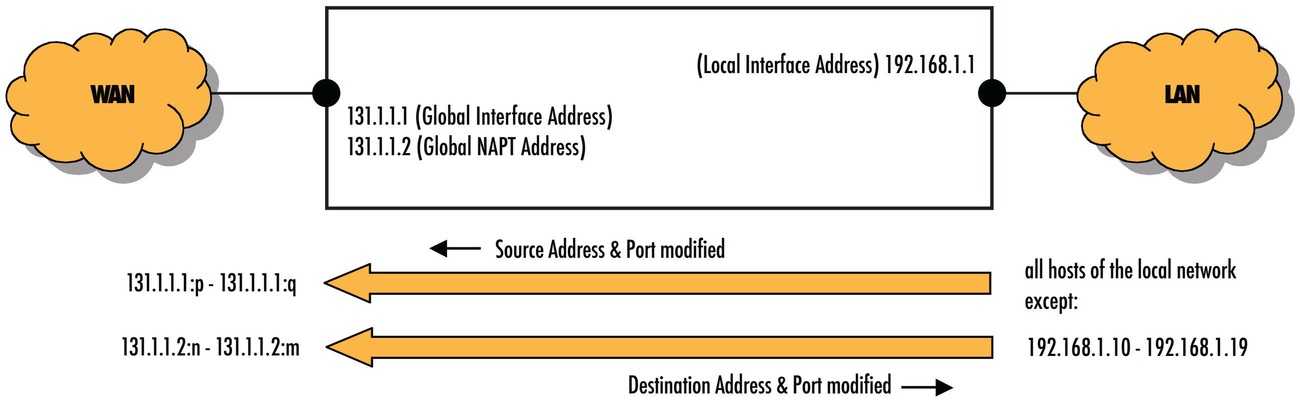

WAN
(Local Interface Address) 192.168.1.1
131.1.1.1 (Global Interface Address)
131.1.1.2 (Global NAPT Address)
LAN
Source Address & Port modified
131.1.1.1:p - 131.1.1.1:q
all hosts of the local network except:
131.1.1.2:n - 131.1.1.2:m
192.168.1.10 - 192.168.1.19
Destination Address & Port modified

Figure 31. Dynamic NAPT

Static NAPT

Static NAPT makes selected services (i.e. ports) of local hosts globally accessible. Static NAPT entries map global addresses/ports to local addresses/ports. The global address can either be the address of the global interface or a configured global NAPT address. Usually, the local and the global port of a static NAPT entry are the same; however, they may be different.

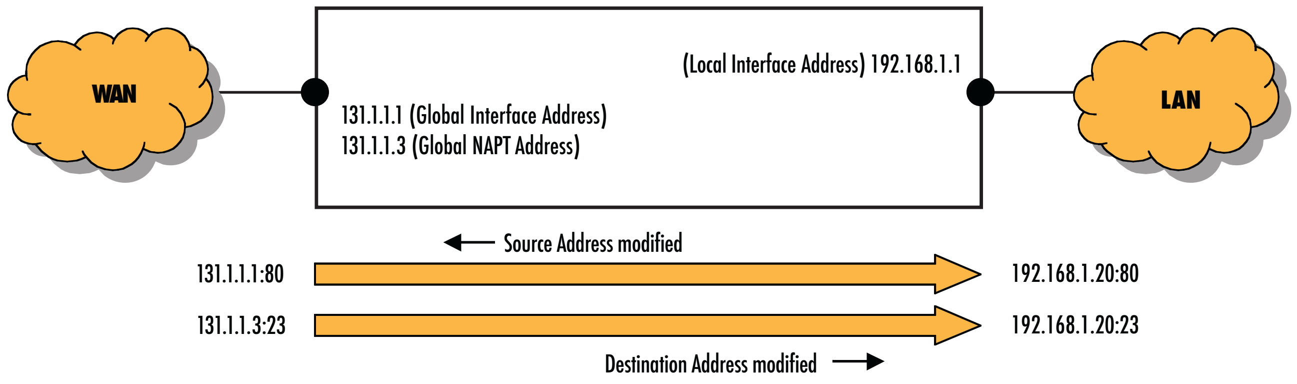

WAN
(Local Interface Address) 192.168.1.1
131.1.1.1 (Global Interface Address)
131.1.1.3 (Global NAPT Address)
LAN
Source Address modified
131.1.1.1:80
192.168.1.20:80
131.1.1.3:23
192.168.1.20:23
Destination Address modified

Figure 32. Static NAPT

Note Be careful when mapping ports the Patton device uses itself (e.g. Telnet, TFTP) because the device might become inaccessible.

Dynamic NAT

NAT only modifies addresses but not ports. Dynamic NAT assigns a global address from a global NAT address pool each time a local host wants to access the global network. It creates a dynamic NAT entry for the reverse path. If a connection is idle for some time (2 minutes), the dynamic NAT entry is removed. Should Dynamic NAT run out of global addresses, it lets Dynamic NAPT handle the connection (which may lead to an unexpected behavior).

Dynamic NAT is particularly useful for protocols that do not build on UDP or TCP but directly on IP (e.g. GRE, ESP). See also section "NAPT traversal" on page 269.

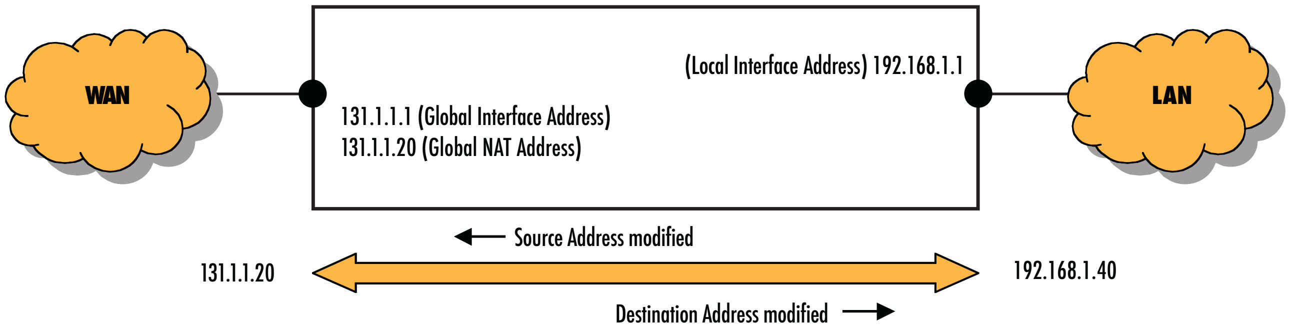

WAN
LAN
(Local Interface Address) 192.168.1.1
131.1.1.1 (Global Interface Address)
131.1.1.20 (Global NAT Address)
Source Address modified
131.1.1.20
192.168.1.40
Destination Address modified

Figure 33. Dynamic NAT

Static NAT

Static NAT makes local hosts globally accessible. Static NAT entries map global addresses to local addresses. The global address must be a configured global NAT address. It cannot be the address of the global interface since this would break connectivity to the Patton device itself.

Static NAT is particularly useful for protocols that do not build on UDP or TCP but directly on IP (e.g. GRE, ESP). See also section "NAPT traversal" on page 269.

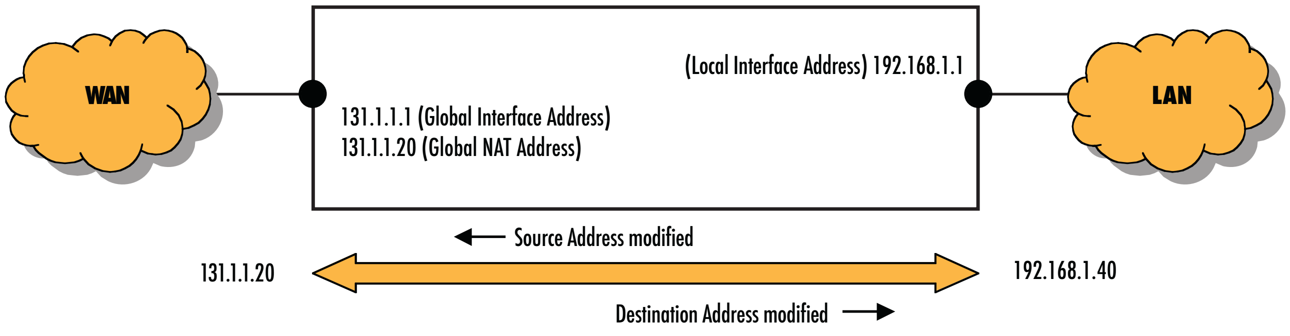

WAN
LAN
(Local Interface Address) 192.168.1.1
131.1.1.1 (Global Interface Address)
131.1.1.20 (Global NAT Address)
Source Address modified
131.1.1.20
192.168.1.40
Destination Address modified

Figure 34. Static NAT

NAPT traversal

Protocols that do not build on UDP or TCP but directly on IP (e.g. GRE, ESP), and protocols that open additional connections unknown to the NAT/NAPT component (e.g. FTP, SIP), do not easily traverse a NAPT.

The Trinity NAPT can handle one GRE (Generic Routing Encapsulation) connection and one ESP (Encapsulating Security Payload) connection at a time. It also routes ICMP messages back to the source of the concerned connection or to the source of an ICMP Ping message.

To enable NAPT traversal of protocols that open additional connections, the NAPT component must analyze these protocols at the Application Level in order to understand which NAPT entries for additional connections it should create and which IP addresses/ports it must modify (e.g. for voice connections in addition to signaling connections). It performs this task for the protocol FTP. Other protocols such as SIP cannot traverse the Trinity NAPT.

NAT/NAPT Configuration Task List

To configure the NAT/NAPT component, perform the tasks in the following sections:

- Creating a NAPT profile (see page 269)
- Activating NAT/NAPT (see page 269)
- Displaying NAT/NAPT configuration information (see page 271)

Creating a NAPT Profile

A NAPT profile defines the behavior of the NAT/NAPT, comprising all four types of NAT/NAPT (this profile is called ‘NAPT profile’ and not ‘NAT/NAPT profile for historical reasons). Several NAPT profiles are admissible but there is only one NAT/NAPT component.

Procedure: To create a NAPT profile and to configure the required types of NAT/NAPT

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#profile napt** *name*	Creates the NAPT profile *name* and activates the basic behavior of the Dynamic NAPT
2 (optional)	***device*(pf-napt)device#range** *local-ip-range-start local-ip-range-stop global-ip*	Configures and activates the enhanced behavior of the Dynamic NAPT: *local-ip-range-start* and *local-ip-range-stop* define the subset of local hosts that use the global NAT address *global-ip* to access to global network. (max. 20 entries) The IP ranges of different Dynamic NAPT entries must not overlap each other.
3 (optional)	***node*(pf-napt)[<*name*>]# static {udp|tcp}** *<local-ip> <local-port> [<global-ip>] [<global-port>]*	Creates a Static NAPT entry: *local-ip/local-port* is mapped to *global-ip/global-port*. If *global-port* is omitted, *local-port* is used on both sides. If *global-ip* is omitted, the global address is the address of the global interface. (*local-port* to be either a single port as before, or a range of ports, e.g. "5000..5999") (max. 20 UDP and 20 TCP entries)

Step	Command	Purpose
4 (optional)	***device*(pf-napt)device#range** *local-ip-range-start local-ip-range-stop global-ip-start global-ip-stop*	Configures and activates the Dynamic NAT: *local-ip-range-start* and *local-ip-range-stop* define the subset of local hosts that use an address from the global NAT address pool to access to global network. *global-ip-start* and *global-ip -stop* define the global NAT address pool. (max. 20 entries) The IP ranges of different Dynamic NAT entries must not overlap each other.
5 (optional)	***device*(pf-napt)device#static** *local-ip global-ip*	Creates a Static NAT entry: *local-ip* is mapped to *global-ip*. (max. 20 entries)
6 (optional)	***device*(pf-napt)device#static { ah|esp|gre|ipv6 }** *local_ip* [*global_ip*].	Creates a static NAT entry: traffic of the IP protocol AH, ESP, GRE, or IPv6 respectively directed to the *global_ip* is forwarded to the *local_ip*.

Use no in front of the above commands to delete a specific entry or the whole profile.

Note The command icmp default is obsolete.

Example: Creating a NAPT Profile

The following example shows how to create a new NAPT profile *access* that contains all settings necessary to implement the examples in section "Introduction" on page 266.

```
device(cfg)#profile napt access
device(pf-napt)[access]#range 192.168.1.10 192.168.1.19 131.1.1.2
device(pf-napt)[access]#static tcp 192.168.1.20 80
device(pf-napt)[access]#static tcp 192.168.1.20 23 131.1.1.3
device(pf-napt)[access]#range 192.168.1.30 192.168.1.39 131.1.1.10 131.1.1.15
device(pf-napt)[access]#static 192.168.1.40 131.1.1.20
device(pf-napt)[access]static ah 192.168.1.41 131.1.1.120
```

Configuring a NAPT DMZ host

The NAPT allows a DMZ host to be configured, which receives any inbound traffic on the global NAPT interface, which:

- Is not translated by any static or dynamic NAPT entry and
- Is not handled by the device itself.

The following procedure shows how a DMZ host can be configured.

Mode: NAPT profile

Step	Command	Purpose
1	***device*(pf-napt)device#[no] dmz-host** *dmz-host-ip-address [global-ip-address]*	Configures a DMZ host. The global-ip-address must only be specified, if the DMZ host shall handle the inbound traffic for a different NAPT global IP address than the gateways global interface IP address.

Activate NAT/NAPT

To enable NAPT on a WAN port, the user has to "use" a NAPT profile on the corresponding IP interface. In order to work, NAPT requires a global IP address. Since there now can be multiple IP addresses per IP interface, the user has to specify which of the IP interface addresses shall be used as NAPT global address.

Step	Command	Purpose
1	**node(cfg)#context ip [ROUTER].**	Enters the IP context mode for the default virtual router.
2	node(ctx-ip)[ROUTER]#interface <name>	Enters the IP interface that is bound to the WAN port.
3	node(ip-if)[ROUTER.<name>]#use profile napt <pfName> <label>	Enables NAPT on the WAN interface and uses its IP address with label <label> as global address.

The <pfName> parameter specifies the NAPT profile to be used for this IP interface. The <label> parameter specifies the name of the IP address on the same IP interface.

Note If the <label> parameter refers to a dynamic address (such as DHCP), the global NAPT address changes automatically when we got a new DHCP lease. If that DHCP address is released, all NAPT translations that require a global interface address are not active while translation rules that explicitly specify a global IP address still work.

Note You can use the same NAPT profile on multiple IP interfaces. In this case, the linked IP interface address will be used as global IP address on the corresponding port.

Example: Create an IP interface with one static an a DHCP address and bind the NAPT profile to the dynamic DHCP address.

```
node>enable
node#configure
node(cfg)#profile napt DYNAMICNAPT
node(cfg)#context ip ROUTER
node(ctx-ip)[ROUTER]#ip interface WAN
node(ip-if)[ROUTER.WAN]#ipaddress STATIC 10.1.1.1/24
node(ip-if)[ROUTER.WAN]#ipaddress DHCP
node(ip-if)[ROUTER.WAN]#use profile napt DYNAMICNAPT DHCP
node(ip-if)[ROUTER.WAN)#port ethernet 0 0
node(prt-eth)[0/0]#bind interface ROUTER WAN
node(prt-eth)[0/0]#no shutdown
```

Displaying NAT/NAPT Configuration Information

Two commands are available to display an existing NAPT profile. There is no command yet to display the dynamic entries of a NAT/NAPT component.

Procedure: To display NAT/NAPT configuration information

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#show profile NAPT**	Displays the available NAPT profiles
2	***device*(cfg)#show profile NAPT** *name*	Displays the NAPT profile *name*

Example: Display NAT/NAPT configuration information

```
device(pf-napt)[access]#show profile NAPT access
NAPT profile access:
--------------------------
STATIC NAPT MAPPINGS
  Protocol     Local IP             Local Port       Global IP        Global Port
  --------    ---------------      -----------      ---------------   -----------
  tcp          192.168.1.20         80               0.0.0.0           80
  tcp          192.168.1.20         23               131.1.1.3         23

STATIC NAT PROTOCOL MAPPINGS
  Protocol Local IP        Global IP
  -------- --------------- ---------------
  ah       192.168.1.41    131.1.1.120

STATIC NAT MAPPINGS
  Local IP        Global IP
  --------------- ---------------
  192.168.1.40    131.1.1.20

STATIC NAPT RANGE MAPPINGS
  Local IP Start  Local IP Stop   Global IP
  --------------- --------------- ---------------
  192.168.1.10    192.168.1.19    131.1.1.15

STATIC NAT RANGE MAPPINGS
  Local IP Start  Local IP Stop   Global IP Start Global IP Stop
  --------------- --------------- --------------- ---------------
  192.168.1.30    192.168.1.39    131.1.1.10      131.1.1.15
```

Chapter 26 **DHCP Configuration**

Chapter contents

Introduction

This chapter provides an overview of the Dynamic Host Configuration Control Protocol (DHCP) and describes the tasks involved in its configuration. This chapter includes the following sections:

- DHCP-client configuration tasks (see page 275)

The Dynamic Host Configuration Protocol (DHCP) automates the process of configuring new and existing devices on TCP/IP networks. DHCP performs many of the same functions a network administrator carries out when connecting a computer to a network. Replacing manual configuration by a program adds flexibility, mobility, and control to networked computer configurations.

The tedious and time-consuming method of assigning IP addresses was replaced by automatic distributing IP addresses. The days when a network administrator had to manually configure each new network device before it could be used on the network are in the past.

In addition to distributing IP addresses, DHCP enables configuration information to be distributed in the form of DHCP options. These options include, for example, the default router address, domain name server addresses, the name of a boot file to load etc.

A new expression in DHCP is lease. Rather than simply assigning each DHCP-client an IP address to keep until the client is done with it, the DHCP-server assigns the client an IP address with a lease; the client is allowed to use the IP address only for the duration of that lease. When the lease expires, the client is forced to stop using that IP address. To prevent a lease from expiring, which essentially shuts down all network access for the client, the client must renew its lease on its IP address from time to time.

The DHCP-server and DHCP-client are illustrated in figure 35.

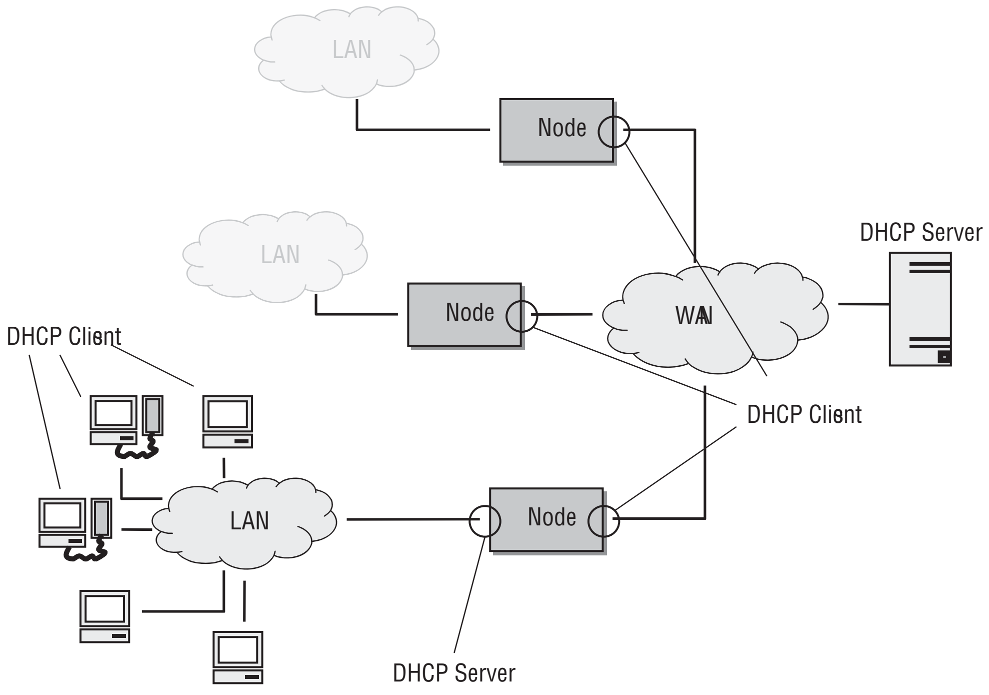


Figure 35. DHCP-client and DHCP-server

DHCP-client Configuration Tasks

To configure the device as DHCP-client, perform the steps mentioned below.

- Configure an IP interface for DHCP
- Release or renew a DHCP lease manually (advanced) (see page 276)
- Remove a DHCP address from an IP interface
- Capture debug output from the DHCP-client (see page 276)

Configure an IP interface for DHCP

On every created IP interface a DHCP-client could be enabled. If enabled, the device gets one IP address for this interface from a DHCP-server. Additionally, other configuration information is received for this IP interface, i.e. the default gateway, DNS server IP addresses, and the host name.

Note Next to a single DHCP address, an IP interface may foster an arbitrary number of static (manually-configured) IP addresses. The DHCP address distinguishes from the static addresses by its *label* set to *DHCP*.

To enable the DHCP-client on an IP Interface, configure the DHCP value with the option "ipaddress"' by performing the steps described below.

Mode: configure

Step	Command	Purpose
1	***device*(cfg)#context ip [ROUTER]**	Enters the IP context configuration mode for the default virtual router.
2	***device*(ctx-ip)[ROUTER] #interface** <*name*>	Enters the configuration mode of an existing IP interface or creates a new IP interface.
3	**device(if-ip)[<name>]#ipaddress [<label>] dhcp [<req-addr>] [route-metric] [ignore { route | dns | host }]**	If no address label is specified the name of the interface will be taken. Creates a DHCP IP address and requests a specific <req-addr> or any address from the DHCP server; potentially ignores information received from the server such as route (default gateway), dns (DNS server IP addresses), and host (host name). If the route-metric is omitted then that device will have a metric of 0 and become the default route.

Release or Renew a DHCP Lease Manually (advanced)

After enabling the DHCP-client, the interface receives a DHCP lease from the DHCP-server. To manually release and/or renew this DHCP lease use the command described below.

Mode: interface

Step	Command	Purpose
1	***device*(ip-if)[ROUTER *name*]#ipaddress DHCP release**	Releases DHCP lease. (See note)
2	***device*(ip-if)[ROUTER *name*]#ipaddress DHCP renew**	Gets a DHCP lease from the DHCP-server

Note If you are connected by Telnet or SSH over the IP interface on which you release the DHCP lease, the connection is lost after entering the command **ipaddress DHCP release**. You need another way (i.e. another static IP address or another IP interface) to connect to the device again and to enter the command **ipaddress DHCP renew.**

Remove a DHCP address from an IP interface

To completely disable the DHCP client on an IP interface, remove the IP address with the “no ipaddress” command.

Mode: interface

Step	Command	Purpose
1	***device*(ip-if)[ROUTER *name*]#[no] ipaddress DHCP**	Disables the DHCP-client on the current IP interface.

Note If you are connected by Telnet or SSH over the IP address you delete, the connection is lost after entering the command no ipaddress. You need another way (e.g. another static IP address, another IP interface, or console access) to connect to the device again.

Capture Debug Output from DHCP-client

This procedure describes how to enable/disable the IP debug monitor, which shows traces of the DHCP client.

Mode: Any

Step	Command	Purpose
1	***device*>#[no] debug ip <detail> <detail level 0-5>**	Enables/disables the DHCP-client debug monitor

Example: Tracing DHCP renew and release requests

This example shows how to enable the IP debug monitor and the debug output generated by the consecutive commands ipaddress DHCP release and ipaddress DHCP renew on IP interface *LAN*.

```
host(if-ip)[ROUTER.LAN]#debug ip
```

```
23:05:02  IP   # [INF] [ROUTER.LAN.DHCP] State: up (normal) | Event: release
23:05:02  IP   # [INF] [ROUTER.LAN.DHCP] Restart Mode: normal -> release
23:05:02  IP   # [INF] [ROUTER.LAN.DHCP] State: up (release) | Action: release
dynamic DHCP address on dev eth0
23:05:02  IP   # [DBG] [eth0] Releasing DHCP address
23:05:02  IP   # [DBG] [ROUTER.LAN.DHCP] onUpdate: restartMode
23:05:02  IP   # [INF] [eth0] DHCP Client >> Unicasting a release of 172.16.60.4 to
172.16.20.10
23:05:02  IP   # [INF] Sending release...
23:05:02  IP   # [INF] [eth0] DHCP Client >> deconfig: interface="eth0"
23:05:02  IP   # [INF] [eth0] DHCP Client: Release DNS server 172.16.20.20
23:05:02  IP   # [INF] [eth0] DHCP Client: Entering released state
23:05:02  IP   # [INF] [eth0] DHCP Client: Release DNS server 172.16.20.10
23:05:02  IP   # [INF] [eth0] DHCP Client: Release default route
23:05:03  IP   # [INF] [eth0] DHCP Client: Release IP address
23:05:03  IP   # [INF] [ROUTER.LAN.DHCP] State: up (release) | Event: dynamic DHCP
address released, reason=DHCP RELEASED
23:05:03  IP   # [INF] [ROUTER.LAN.DHCP] State: up (release) | Action: revoke
address 172.16.60.4/19 from dev eth0
23:05:03  IP   # [INF] [eth0] Kernel << ip addr del 172.16.60.4/19 dev eth0
23:05:03  IP   # [INF] [ROUTER.LAN.DHCP] State: up (release) -> revoking
23:05:03  IP   # [INF] [ROUTER.LAN.DHCP] State: up (release) | Action: decrement
active address counter
23:05:03  IP   # [DBG] [ROUTER.LAN.DHCP] onUpdate: recvAddress, state
23:05:03  IP   # [DBG] [ROUTER.LAN] onUpdate: activeV4AddrCount
23:05:03  IP   # [INF] [eth0] Kernel >> Address departed: DHCP (172.16.60.4/19)
23:05:03  IP   # [INF] [ROUTER.LAN.DHCP] State: revoking (release) | Event: down
23:05:03  IP   # [INF] [ROUTER.LAN.DHCP] State: revoking (release) | Clear address
23:05:03  IP   # [INF] [ROUTER.LAN.DHCP] State: revoking (release) | Action: restart
(release)
23:05:03  IP   # [INF] [ROUTER.LAN.DHCP] State: revoking (release) | Action: release
dynamic DHCP address on dev eth0 forced
23:05:03  IP   # [DBG] [eth0] Releasing DHCP address
23:05:03  IP   # [INF] [ROUTER.LAN.DHCP] Restart Mode: release -> normal
23:05:03  IP   # [INF] [ROUTER.LAN.DHCP] State: revoking (normal) -> released
23:05:03  IP   # [DBG] [ROUTER.LAN.DHCP] onUpdate: address, restartMode, state
host(if-ip)[ROUTER.LAN]#ipaddress DHCP renew
23:05:15  IP   # [INF] [ROUTER.LAN.DHCP] State: released (normal) | Event: renew
23:05:15  IP   # [INF] [ROUTER.LAN.DHCP] State: released (normal) | Action: request
dynamic DHCP address on dev eth0
23:05:15  IP   # [INF] [eth0] Starting DHCP client
23:05:15  IP   # [INF] [eth0] DHCP Client << udhcpc -f -R -i eth0 -s /usr/bin/
DHCPInsert -C host -H host
23:05:15  IP   # [DBG] [ROUTER.LAN.DHCP] onUpdate
23:05:15  IP   # [INF] [eth0] DHCP Client >> udhcpc (v1.19.3) started
23:05:15  IP   # [INF] [eth0] DHCP Client >> deconfig: interface="eth0"
23:05:15  IP   # [INF] [eth0] DHCP Client: Release default route
23:05:15  IP   # [INF] [eth0] DHCP Client >> Sending discover...
23:05:15  IP   # [INF] [eth0] DHCP Client >> Sending select for 172.16.60.4...
23:05:15  IP   # [INF] Lease of 172.16.60.4 obtained, lease time 86400
23:05:15  IP   # [INF] [eth0] DHCP Client: Release IP address
23:05:15  IP   # [INF] [ROUTER.LAN.DHCP] State: released (normal) | Event: dynamic
DHCP address released, reason=DHCP RELEASED
23:05:15  IP   # [DBG] [ROUTER.LAN.DHCP] onUpdate
23:05:15  IP   # [INF] [eth0] DHCP Client >> bound: interface="eth0",
ip="172.16.60.4", mask="19", router="172.16.32.1", dns="172.16.20.10 172.16.20.20"
23:05:15  IP   # [INF] [eth0] DHCP Client: Apply IP address 172.16.60.4/19
23:05:15  IP   # [INF] [eth1.1] DHCP Client >> Sending discover...
23:05:15  IP   # [INF] [ROUTER.LAN.DHCP] State: released (normal) | Event: dynamic
DHCP address received: address=172.16.60.4/19, broadcast=
23:05:15  IP   # [INF] [ROUTER.LAN.DHCP] State: released (normal) | Action: apply
address 172.16.60.4/19 to dev eth0
23:05:15  IP   # [INF] [eth0] Kernel << ip addr add 172.16.60.4/19 broadcast + label
eth0:DHCP dev eth0
23:05:15  IP   # [INF] [ROUTER.LAN.DHCP] State: released (normal) -> applying
23:05:15  IP   # [DBG] [ROUTER.LAN.DHCP] onUpdate: address, recvAddress, state
23:05:15  IP   # [INF] [eth0] DHCP Client: Apply default route via gateway
172.16.32.1
23:05:16  IP   # [INF] [eth0] DHCP Client: Apply DNS server 172.16.20.10
23:05:16  IP   # [INF] [eth0] DHCP Client: Apply DNS server 172.16.20.20
23:05:16  IP   # [INF] [eth0] Kernel >> Address arrived: DHCP (172.16.60.4/19)
23:05:16  IP   # [INF] [ROUTER.LAN.DHCP] State: applying (normal) | Event: up
23:05:16  IP   # [INF] [ROUTER.LAN.DHCP] State: applying (normal) -> up
23:05:16  IP   # [INF] [ROUTER.LAN.DHCP] State: applying (normal) | Action: incre-
ment active address counter
23:05:16  IP   # [DBG] [ROUTER.LAN.DHCP] onUpdate: state
23:05:16  IP   # [DBG] [ROUTER.LAN] onUpdate: activeV4AddrCount
```

Chapter 27 DNS Configuration

Chapter contents

Introduction

The domain name system (DNS) enables users to contact a remote host by using easily remembered text labels (www.patton.com, for example) instead of having to use the host's numeric address (209.45.110.15, for example). When DNS names are entered as part of configuration commands or CLI exec mode commands in applications like Ping, Traceroute, or Tftp, the Patton device uses a DNS resolver component to convert the DNS names into the numeric address.

The Patton device can be configured as a caching DNS relay server to speed data transfers, acting as the DNS server for a private network. In this configuration, hosts in the network send their DNS queries to the Patton device, which checks to see if the DNS name is in its DNS resolver cache. If it finds the name in cache, the device uses the cached data to resolve the DNS name into a numeric IP address. If the name is not in cache, the query is forwarded on to a DNS server. When the device receives the answer from the server, it adds the name to the cache, and forwards it on to the host that originated the query. This process enables the Patton device to provide answers more quickly to often-queried DNS names, reducing the number of DNS queries that must be sent across the access link.

DNS Configuration Task List

The following sections describe how to configure the DNS component:

- Enabling the DNS resolver
- Enabling the DNS relay

Enabling the DNS Resolver

To enable the Patton device DNS resolver for manually configured DNS upstream servers, you must configure it with the address of one or more DNS servers that will be used to resolve DNS name queries. If multiple DNS servers are configured, the device will query each server in turn until a response is received.

There are two configuration modes: "dns-server" and "dns-client".

- The "dns-server" mode is meant to configure the own DNS server of the Patton device, defining its own port, domain and host -> address static translations. For specific domains (which can be defined too), it can also work as a relay, forwarding the DNS queries to the configured DNS servers. This mode can be shut down to disable the DNS server.
- The "dns-client" mode is meant to define the DNS servers the Patton device will query when the "dns-server" mode cannot provide the required DNS resolution. This can happen when the "dns-server" mode is shut down, it does not have the required host-address translation set up, the domain is not listed in the ones configured in the DNS-relay, or those servers are not accessible. The "dns-client" mode cannot be shut-down.

For DNS servers being received through DHCP or PPP, no configuration is needed. These DNS servers will be stored regardless of the dns-server config, and will be used for DNS queries originating on the Patton device.

Manual entries for DNS servers are configured as follows:

Mode: dns-server

Step	Command	Purpose
1	node(dns-srv)#relay name-server domain <name> <IPv4|IPv6> [<port>]	Configures a DNS server used by the relay for the specific domain name.
Repeat step 1 for each additional DNS server you want to add		

Example: Configuring DNS servers

The following example shows how to add DNS servers to the Patton device DNS resolver and increase the size of the DNS cache to 100 entries.

```
node>enable
node#configure
node#dns-server name-server address 62.2.32.5
node#dns-server no shutdown
```

Mode: dns-client

Step	Command	Purpose
1	node(dns-client)#name-server <IPv4|IPv6>	Defines a DNS server used by the Patton device with the specific IP address.

Enabling the DNS Relay

To use DNS relay on the device, the DNS server has to be enabled ("no shutdown").

The DNS resolver automatically learns domain name servers if it receives them through PPP or DHCP protocols, regardless if dns-server is set to shutdown or not.

You can verify that the DNS resolver has received domain name servers as well as the manually configured upstream DNS servers, by using the `show dns-client`command as follows:

```
node(cfg)#show dns
DNS Configuration
===============================================
Shutdown             : false
Name                 :
Port                 : 53

Name Server Table
===============================================
IP                   Domain               Port
192.168.1.1                               53

DNS Host Information
===============================================
Host IP              Host Name
```

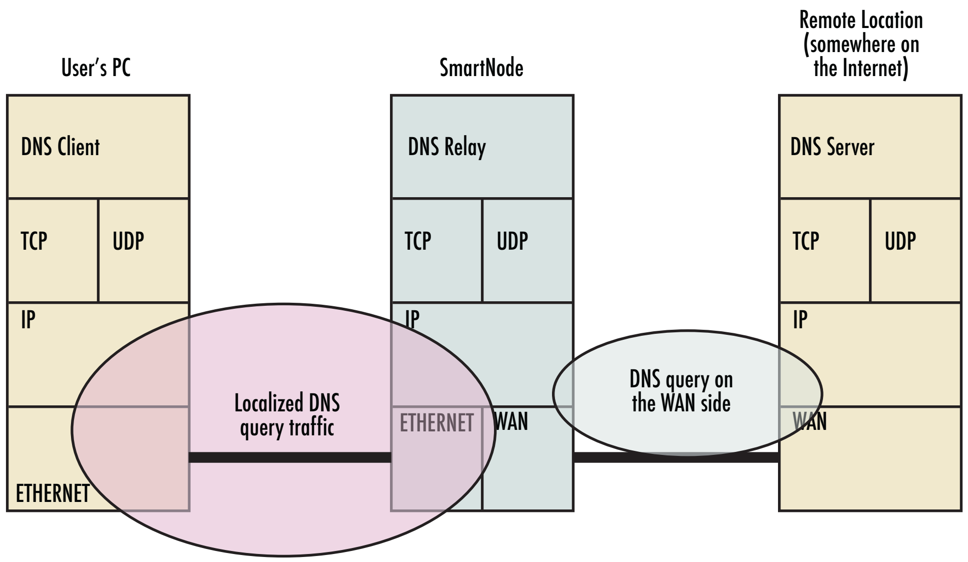

User's PC
SmartNode
Remote Location (somewhere on the Internet)
DNS Client
TCP
UDP
IP
ETHERNET
DNS Relay
TCP
UDP
IP
ETHERNET
WAN
DNS Server
TCP
UDP
IP
WAN
Localized DNS query traffic
DNS query on the WAN side

Figure 36. DNS relay diagram

Chapter 28 SNTP Client Configuration

Chapter contents

Introduction

This chapter describes how to configure Network Time Protocol (NTP) client. The NTP Management component enables users to setup an NTP time updating, given they are connected to the Internet. The NTP allows you to set several NTP Servers to allow for precise time keeping on the device. The NTP also allows you to listen to the NTP Broadcasts.

NTP Client Configuration Task List

The following sections describe how to configure the NTP component:

- Enabling/Disabling NTP Management component
- Configuring NTP options

Enabling/Disabling the NTP Management Component

Mode: Configure

Step	Command	Purpose
1	**device(cfg)# ntp no shutdown**	Enables the management component
2	**device(cfg)# ntp shutdown**	Disables the management component

Enabling NTP Options

Mode: Configure

Step	Command	Purpose
1	**device(cfg)# ntp broadcastclient**	Enables the broadcast client. NTP component supports NTP Broadcasts. The inverted command 'no' will disable the NTP server
2	**device(cfg)# ntp** <*server*>	Adds NTP Server to the DB.
3	**device(cfg)# ntp** <*server*> **multicast**	Optional multicast flag that will enable multicast NTP capabilities

Mode: Configure

Step	Command	Purpose
1	**device(cfg)# show ntp config**	Shows NTP Configuration
2	**device(cfg)# show ntp status**	Shows NTP Status

Examples

Run the following to see the NTP configuration settings

```
device(cfg)# show ntp config

NTP Status and Information
===============================================
Broadcast Client     : false
Shutdown             : false

NTP Servers
===============================================
192.168.0.2          : Unicast
96.47.67.105         : Unicast
```

Run the following to see the NTP status

The Active server will be shown with an arrow. The Server name is what the NTP config entry has resolved to. It may be different from what you've configured. That is how the protocol is meant to be operating:

```
device(cfg)# show ntp status

Active      Server                  Delay     Offset    Jitter
================================================================
            hit-nxdomain.op         0.000     0.000     0.000
  --->      nist1-nj.ustimi         23.244    0.575     1.592
```

Chapter 29 SNMP Configuration

Chapter contents

Introduction

This chapter provides overview information about Simple Network Management Protocol (SNMP) and describes the tasks used to configure those of its features supported.

Note SNMP v1/v2/v3 are supported in Trinity from version 3.7 onwards.

This chapter includes the following sections:

- Simple Network Management Protocol (SNMP)
- SNMP tools (see page 289)
- SNMP configuration task list (see page 289)
- Using the ManageEngine SNMP utilities (see page 295)
- Standard SNMP version 1 traps (see page 299)

Simple Network Management Protocol (SNMP)

The Simple Network Management Protocol (SNMP) is an application-layer protocol that facilitates the exchange of management information between network devices. It is part of the Transmission Control Protocol/Internet Protocol (TCP/IP) suite. SNMP enables network administrators to manage network performance, find and solve network problems, and plan for network growth.

SNMP Basic Components

An SNMP managed network consists of three key components: managed devices, agents, and network-management systems (NMSs).

A managed device is a network SN that contains an SNMP agent and resides on a managed network. Managed devices collect and store management information and make this information available to NMSs using SNMP. Managed devices, sometimes called network elements, can be routers and access servers, switches and bridges, hubs, computer hosts, or printers.

An agent is a network-management software module that resides in a managed device. An agent has local knowledge of management information and translates that information into a form compatible with SNMP.

An NMS executes applications that monitor and control managed devices. NMSs provide the bulk of the processing and memory resources required for network management. One or more NMSs must exist on any managed network.

SNMP Basic Commands

Managed devices are monitored and controlled using four basic SNMP commands: read, write, trap, and traversal operations.

- The read command is used by an NMS to monitor managed devices. The NMS examines different variables that are maintained by managed devices.
- The write command is used by an NMS to control managed devices. The NMS changes the values of variables stored within managed devices.
- The trap command is used by managed devices to asynchronously report events to the NMS. When certain types of events occur, a managed device sends a trap to the NMS.

- Traversal operations are used by the NMS to determine which variables a managed device supports and to sequentially gather information in variable tables, such as a routing table.

SNMP Management Information Base (MIB)

A Management Information Base (MIB) is a collection of information that is organized hierarchically. MIBs are accessed using a network-management protocol such as SNMP. They are comprised of managed objects and are identified by object identifiers.

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. Objects in the MIB are defined using the subset of abstract syntax notation one (ASN.1) defined in the SMI. In particular, an *object identifier*, an administratively assigned name, names each object type. The object type together with an object instance serves to uniquely identify a specific instantiation of the object. For human convenience, a textual string, termed the descriptor, to refer to the object type, is often used.

An object identifier (OID) world-wide identifies a managed object in the MIB hierarchy. The MIB hierarchy can be depicted as a tree with a nameless root, the levels of which are assigned by different organizations.

Network Management Framework

This section provides a brief overview of the current SNMP management framework. An overall architecture is described in RFC 2571 "An Architecture for Describing SNMP Management Frameworks." The SNMP management framework has several components:

- Mechanisms for describing and naming objects and events for the purpose of management. The first version, Structure of Management Information (SMIv1) is described in RFC 1155 "Structure and Identification of Management Information for TCP/IP-based Internets", RFC 1212 "Concise MIB Definitions", RFC 1213 "Management Information Base for Network Management of TCP/IP-based Internets: MIB-II", and RFC 1215 "A Convention for Defining Traps for use with the SNMP". The second version, SMIv2, is described in RFC 2233 "The Interfaces Group MIB using SMIv2", RFC 2578 "Structure of Management Information Version 2 (SMIv2)", RFC 2579 "Textual Conventions for SMIv2", and RFC 2580 "Conformance Statements for SMIv2".
- Message protocols for transferring management information. The first version, SNMPv1, is described in RFC 1157 "A Simple Network Management Protocol (SNMP)." The second version, SNMPv2, which is not an Internet standards track protocol, is described in RFC 1901 "Introduction to Community-Based SNMPv2" and RFC 1906 "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)".
- Protocol operations for accessing management information. The first set of protocol operations and associated protocol data unit (PDU) formats is described in RFC 1157. The second set of protocol operations and associated PDU formats is described in RFC 1905 "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)".
- A set of fundamental applications described in RFC 2573 "SNMP Applications" and the view-based access control mechanism described in RFC 2575 "View-Based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)".

Identification of a Patton Device via SNMP

All product models have assigned sysObjectID.

Refer to the getting started guide of your product, or see the MIB definition file (.my) for sysObjectIDs.

SNMP Tools

Patton recommends the ManageEngine.

Refer to section "Using the ManageEngine SNMP Utilities" on page 295 for more detailed information on how to use these tools.

SNMP Configuration Task List

To configure SNMP, perform the tasks described in the following sections. The tasks in the first three sections are required; the tasks in the remaining sections are optional, but might be required for your application.

- Setting basic system information (required) (see page 289)
- Setting access community information (required) (see page 292)
- Setting allowed host information (required) (see page 293)
- Specifying the default SNMP trap target (optional) (see page 294)
- Displaying SNMP related information (optional) (see page 294)

Setting Basic System Information

The implementation of the MIB-II system group is mandatory for all systems. By default, an SNMP agent is configured to have a value for any of these variables and responds to get commands from a NMS.

The following MIB II panels should be set:

- sysContact
- sysLocation
- sysName

The system sysContact object is used to define the contact person, together with information on how to contact that person.

Assigning explanatory location information to describe the system physical location (e.g. server room, wiring closet, 3rd floor, etc.) is very supportive. Such an entry corresponds to the MIB II system sysLocation object.

The name used for sysName should follow the rules for ARPANET host names. Names must start with a letter, end with a letter or digit, and have as interior characters only letters, digits, and hyphens. Names must be 63 characters or fewer. For more information, refer to RFC 1035.

This procedure describes how to set these MIB-II system group objects.

Mode: Administrator execution

Step	Command	Purpose
1	***device*(cfg)#system contact** *name*	Sets the contact persons name

Step	Command	Purpose
2	***device*(cfg)#system location** *location*	Sets the system location
3	***device*(cfg)#system hostname** *hostname*	Sets the system hostname and command line prompt

If any of the command options *name*, *location*, or *hostname* has to be formed out of more than one word, the information is put in “double quotes”.

Note Enter an empty string “” to get rid of any of the system settings.

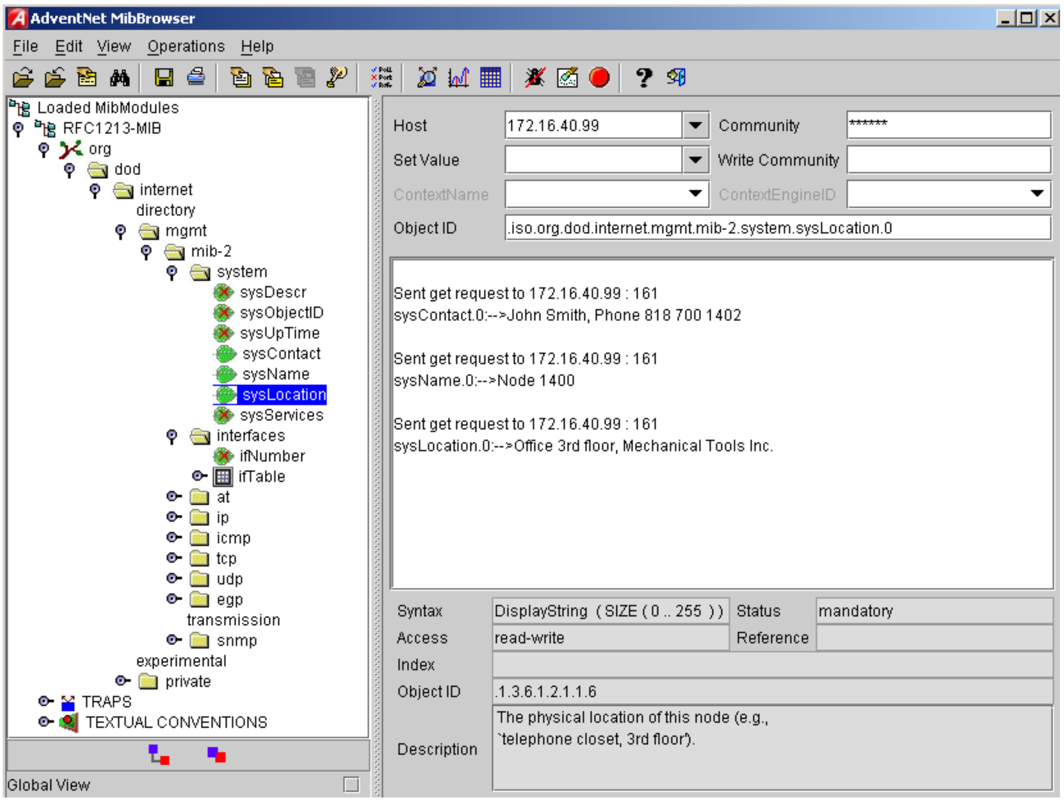
The MIB-II system group values are accessible for reading and writing via the following SNMP objects:

- .iso.org.dod.internet.mgmt.mib-2.system.sysContact
- .iso.org.dod.internet.mgmt.mib-2.system.sysName
- .iso.org.dod.internet.mgmt.mib-2.system.sysLocation

After setting these values according to 1 through 3 any SNMP MIB browser application should read the values using a get or get-next command as shown in figure 37.

The procedure to use the SNMP MIB browser is:

- Enter the community string public into the Community field in the upper right corner of the window. For safety reasons each entered character is displayed with a “*”.
- Access any of the supported MIB system group object by using the GetNext button from the button bar of the window.



Host	172.16.40.99	Community	******
Set Value		Write Community	
ContextName		ContextEngineID	
Object ID	.iso.org.dod.internet.mgmt.mib-2.system.sysLocation.0		

```
Sent get request to 172.16.40.99 : 161
sysContact.0:-->John Smith, Phone 818 700 1402

Sent get request to 172.16.40.99 : 161
sysName.0:-->Node 1400

Sent get request to 172.16.40.99 : 161
sysLocation.0:-->Office 3rd floor, Mechanical Tools Inc.
```

Syntax	DisplayString (SIZE (0 .. 255))	Status	mandatory
Access	read-write	Reference	
Index			
Object ID	.1.3.6.1.2.1.1.6		
Description	The physical location of this node (e.g., `telephone closet, 3rd floor').		

Figure 37. ManageEngine MibBrowser displaying some of the System Group objects

Example: Setting the system group objects

In the following example the system information is set for later access via SNMP. See figure 37 for a typical MIB browser application accessing these MIB-II system group objects representing the system information.

```
device>enable
device#configure
device(cfg)#system contact "Bill Anybody, Phone 818 700 1504"
device(cfg)#system location "Wiring Closet, 3rd floor"
device(cfg)#system hostname "device"
(cfg)#
```

After entering a host name the prompt on the CLI no longer displays the IP address of the Ethernet port over which the Telnet session is running but shows the newly entered host name.

Setting Access Community Information

SNMP uses one or more labels called *community strings* to delimit groups of *objects* (variables) that can be viewed or modified on a device. The SNMP data in such a group is organized in a tree structure called a Management Information Base (MIB). A single device may have multiple MIBs connected together into one large structure, and various community strings may provide read-only or read-write access to different, possibly overlapping portions of the larger data structure. An example of a read-only variable might be a counter showing the total number of octets sent or received through an interface. An example of a read-write variable might be the speed of an interface, or the hostname of a device.

Community strings also provide a weak form of access control in earlier versions of SNMP version 1 and 2. SNMP version 3 provides much improved access control using strong authentication and should be preferred over SNMP version 1 and 2 wherever it is supported. If a community string is defined, then it must be provided in any basic SNMP query if the requested operation is to be permitted by the device. Community strings usually allow read-only or read-write access to the entire device. In some cases, a given community string will be limited to one group of read-only or read-write objects described in an individual MIB.

In the absence of additional configuration options to constrain access, knowledge of the single community string for the device is all that is required to gain access to all objects, both read-only and read-write, and to modify any read-write objects.

Note Security problems can be caused by unauthorized individuals possessing knowledge of read-only community strings so they gain read access to confidential information stored on an affected device. Worse can happen if they gain access to read-write community strings that allow unauthorized remote configuration of affected devices, possibly without the system administrators being aware that changes are being made, resulting in a failure of integrity and a possible failure of device availability. To prevent these situations, define community strings that only allow read-only access to the MIB objects should be the default.

Choosing community names is like choosing a password. Do not use easily guessed ones; do not use commonly known words, mix letters and other characters, and so on. If you do not intend to allow anyone to use SNMP write commands on your system, then you probably only need one community name.

This procedure describes how to define your own SNMP community.

Mode: Configure

Step	Command	Purpose
1	***node*(cfg)#snmp-server**	Enters snmp-server mode

Use the no command option to remove a SNMP community setting.

Example: Setting access community information

In the following example the SNMP communities for the default community public with read-only access and the undisclosed community Not4evEryOne with read/write access are defined. Only these valid communities have access to the information from the SNMP agent.

```
device(cfg)#snmp community public ro
device(cfg)#snmp community Not4evEryOne rw
```

Note If no community is set on your Patton device accessing any of the MIB objects is not possible!

Setting Allowed Host Information

If a host has to access SNMP MIB objects on a certain device, it explicitly needs the right to access the SNMP agent. Therefore a host needs an entry, which allows accessing the device. The host is identified by its IP address and has to use a certain community string for security precautions.

Note The community which is to be used as security name to access the MIB objects has to be defined prior to the definition of allowed hosts.

This procedure describes adding a host that is allowed to access the MIB of this system.

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#snmp host** *IP-address-of-device* **security-name** *community*	Configures a host that with IP address *IP-address-of-device* can access the MIB, using the security name *community*.

Use the no command option to remove a SNMP allowed host setting.

Example: Setting allowed host information

In the following example the host with IP address *172.16.224.45* shall be able to access the MIB using community *public* as security name.

```
device(cfg)#snmp host 172.16.224.45 security-name public
```

Authentication and Encryption

The authentication and encryption protocols can be configured globally for all users. The users have the option of SHA or MD5 for authentication and AES or DES for packet encryption. (If either are disabled, SNMPv3 will not work.) (NOTE: SNMPv1 and SNMPv2 cannot be explicitly disabled, however the service cannot be accessed if no HOSTS are defined.)

By default, all users in the local AAA database can assess SNMP; however this can be limited when defining the users.

Mode: snmp-server

Step	Command	Purpose
1	**[no] authentication-protocol {sha|md5}**	Authentication protocol used to access service.
2	**[no] encryption-protocol {aes|des}**	Encryption protocol used to access service.

Mode: configure

Step	Command	Purpose
1	**[no] {superuser|administrator|operator} <user-name> {no-password|password <password> [encrypted]} [terminal-type {console|http|snmp|ssh|telnet}]**	Optionally select what type of service a user can access from.

Specifying the Default SNMP Trap Target

An SNMP trap is a message that the SNMP agent sends to a network management station. For example, an SNMP agent would send a trap when an interface's status has changed from up to down. The SNMP agent must know the address of the network management station so that it knows where to send traps. It is possible to define more than one SNMP trap target.

The SNMP message header contains a *community* field. The SNMP agent uses a defined community name, which is inserted in the trap messages header sent to the target. In most cases the target is a NMS, which only accepts a SNMP message header of a certain community.

This procedure describes how to define a SNMP trap target and enter community name.

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#snmp target** *IP-address-of-device* **security-name** *community*	Configures a SNMP trap target with IP-address-of-hostname *device* that receives trap messages using the security name *community* on the target.

Use the no command option to remove s SNMP trap target setting.

Example: Specifying the default SNMP trap target

In the following example the NMS running on host with IP address 172.16.224.44 shall be defined as SNMP trap target. Since the NMS requires that SNMP message headers have a community of *Not4evEryOne* the security-name argument is set accordingly.

```
device(cfg)#snmp target 172.16.224.44 security-name Not4evEryOne
```

Displaying SNMP Related Information

Displaying the SNMP related configuration settings is often necessary to check configuration modifications or when determining the behavior of the SNMP agent.

This procedure describes how to display information and configuration settings for SNMP.

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#show snmp**	Displays information and configuration settings for SNMP

Example: Displaying SNMP related information

This example shows how to display SNMP configuration information.

```
device(cfg)#show snmp

 Hosts:
   172.16.224.44 security-name public

 Targets:
   172.16.224.44 security-name Not4evEryOne

 Communities:
   public access-right ro
   Not4evEryOne access-right rw
```

Using the ManageEngine SNMP Utilities

The ManageEngine SNMP utilities are a set of cross-platform applications and applets for SNMP and Web-based network management. These utilities can be used for device, element, application and system management. The following tools are the most useful:

- MibBrowser—used to view and operate on data available through a SNMP agent on a managed device
- TrapViewer—used to parse and view the received traps

The ManageEngine is a complete SNMP MibBrowser that enables the loading of MIBs, MIB browsing, walking a MIB tree, searching MIBs and performing all other SNMP-related functions to users.

Viewing and operating the data available through an SNMP agent on a managed device, e.g. a router, switch, hub etc., is made possible by using the MibBrowser.

The TrapViewer is a graphical tool to view the Traps received from one or more SNMP agents. The Trap viewer can listen to one or more port at a time and the traps can be sent from any host. Moreover the TrapViewer contains a Trap parser editor, which is a tool to create a trap parser file. The Trap viewer parses the file created using Trap parser editor to match each incoming traps with certain criteria. Since Traps typically contain cryptic information, which is not easily understandable to the users, trap parsers are required to translate or parse traps into understandable information.

Using the MibBrowser

Figure 38 on page 296 depicts the primary window of the ManageEngine MibBrowser. It consists of a menu bar, a toolbar, a left frame and a right frame.

The operations that can be performed by the MibBrowser are available in a series of buttons in the toolbar on top of the MibBrowser's main window. The toolbar can be hidden or made visible using the options available.

The menu bar has various options that perform the same operations as the options available in the toolbar.

The left frame holds the MIB tree. A MIB tree is a structure through which all the MIBs loaded can be viewed. The MIB tree component enables us to traverse through the tree, view the loaded MIBs and learn the definition for each SN. The ManageEngine MibBrowser allows loading additional MIB files in the text format (the “my” file contains enterprise specific MIB definitions).

The right frame has labeled text fields to specify the basic parameters like host, community etc. and a Result text area display to view the results.

There are three ways in which the primary window of the MibBrowser can be viewed. It can be viewed with the result display, MIB description panel or multi-variable bind panel in the right frame. The view can be altered in three ways.

- The desired view can be set by the options provided in the display menu item under the view menu. (View I Display I).
- The other way of altering the view is through the general settings panel in the settings menu item in the edit menu. (Edit I Settings)
- The same can be done through clicking the MibBrowser settings button on the toolbar. See figure 38.

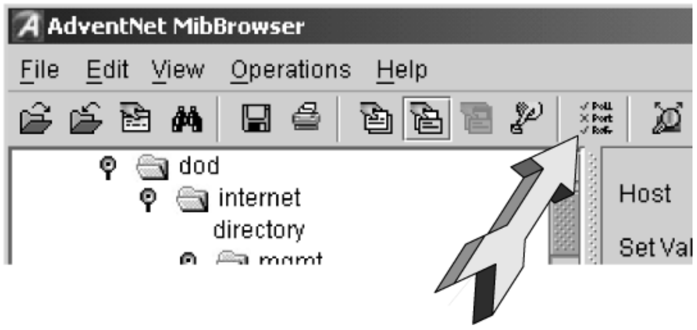


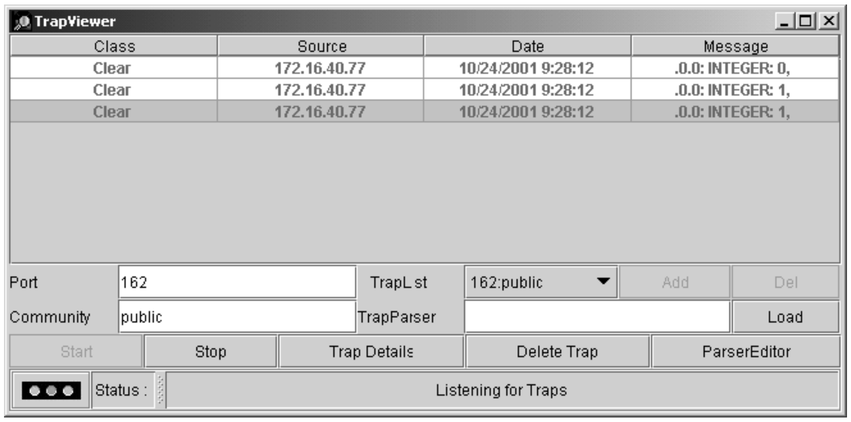
Figure 38. ManageEngine MibBrowser Settings Button on the Toolbar

By default the MIB description display and the result display are visible in the MibBrowser.

Using the TrapViewer

TrapViewer is a graphical tool to view the traps received from one or more SNMP agents. The TrapViewer can listen to one or more port at a time and the traps can be sent from any host.

Invoke the TrapViewer through the usage of the MibBrowser. To get to know more about the MibBrowser refer to section "Using the MibBrowser" on page 295. Figure 39 is a screen shot of the TrapViewer.



Class	Source	Date	Message
Clear	172.16.40.77	10/24/2001 9:28:12	.0.0: INTEGER: 0,
Clear	172.16.40.77	10/24/2001 9:28:12	.0.0: INTEGER: 1,
Clear	172.16.40.77	10/24/2001 9:28:12	.0.0: INTEGER: 1,

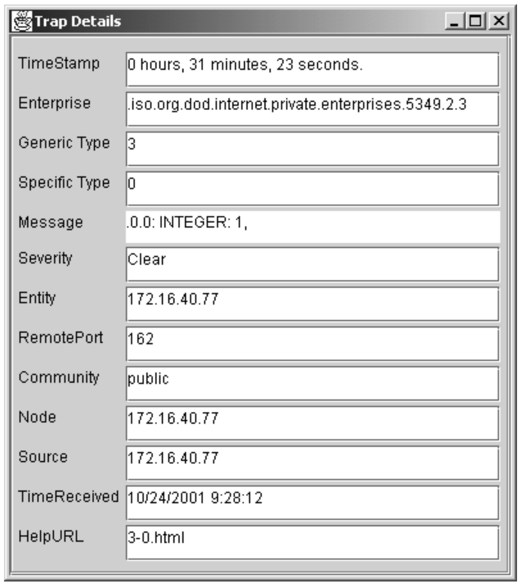
Port: 162
Community: public
TrapList: 162:public
Status : Listening for Traps

Figure 39. ManageEngine TrapViewer displaying received traps

The TrapViewer has a table that displays the trap information, the common parameters text fields where necessary information has to be entered and other options such as Start, Stop, Trap Details, Delete Trap and ParserEditor.

Follow these steps to work on the Trap Viewer and to know more about the available options:

- By default the value in the *Port* text field is 162. Enter the desired port in the field on which the viewer will listen.
- The default value in the *Community* text field is public. Set the community of the incoming traps as desired, depending on the SNMP configuration.
- Click on *Add* button to add the port and community list on which the trap has to listen to. This is visible in the *TrapList* combo box.
- The port and community list can be deleted by clicking on the *Del* button.
- When you need to load a trap parser file, click on the *Load* button, which will open up a dialog box, from which you can load the parser file.
- In order to receive the traps now, click on the *Start* button. Upon clicking this button, TrapViewer begins to receive traps according to the as-specified port and community.
- Once received, the traps are listed in the trap table of the TrapViewer. By default, the trap table has the following four columns:
  - *Class* that defines the severity of the trap.
  - *Source* that displays the IP address of the source from where the traps were sent.
  - *Date* that shows the date and time when the trap was received.
  - *Message* that by default has the object identifier format (sequence of numeric or textual labels on the SNs along a path from the root to the object) of the trap if any, or it is blank.
- The details of the traps can be viewed by clicking the *Trap Details* button or right click the trap in the trap table and select the option *View Trap Details*. figure 40 show the screen of such a trap details window.



TimeStamp	0 hours, 31 minutes, 23 seconds.
Enterprise	.iso.org.dod.internet.private.enterprises.5349.2.3
Generic Type	3
Specific Type	0
Message	.0.0: INTEGER: 1,
Severity	Clear
Entity	172.16.40.77
RemotePort	162
Community	public
Node	172.16.40.77
Source	172.16.40.77
TimeReceived	10/24/2001 9:28:12
HelpURL	3-0.html

Figure 40. ManageEngine Trap Details window of TrapViewer

The various details available in the Trap Details window are listed in table 25:

Table 25. Details available in the Trap Details window

Trap Details	Description
TimeStamp	The TimeStamp is a 32-bit unsigned value indicating the number of hundredths-of-a-second that have elapsed since the (re)start of the SNMP agent and the sending of the trap. This field shows the value stored in the MIB-II sysUpTime variable converted into hours, minutes and seconds.
Enterprise	This field shows the OID of the management enterprise that defines the trap message. The value is represented as an OBJECT IDENTIFIER value and has a variable length.
Generic Type	The Generic type value is categorized and numbered 0 to 6. They are 0-coldStart, 1-warmStart, 2-linkDown, 3-linkUp, 4-authenticationFailure, 5-egpNeighborLoss. The trap type value 6 is identified as enterprise-specific value. This field shows the value based on the type of trap.
Specific Type	The specific trap type indicates the specific trap as defined in an enterprise-specific MIB. If the Generic type value is 6, then this field shows a value greater than 0. If the generic type value is a value other than 6, then the field shows a value 0. This field can have values from 0 to 2147483647.
Message	This is a text field. By default, this field will always contain the Varbinds in the Trap PDU. This can be substituted with text.
Severity	This field shows the Severity or the intensity of the trap. They could be 0-All, 1-Critical, 2-Major, 3-Minor, 4-warning, 5-Clear and 6-info.
Entity	The source IP address from which the Trap was sent is displayed here.
RemotePort	This field reveals the port on which the Trap was sent by the originator.
Community	The Community string is displayed here.
device	Source

Table 25. Details available in the Trap Details window (Continued)

Trap Details	Description
TimeReceived	This displays the Date and Time when the trap was received.
HelpURL	The URL shown here gives more details of the received trap. By default, the URL file name is <generic-type value> - <specific-type value>.html

You can stop the listening by clicking the *Stop* button.

When you need to delete the trap, select the trap to be deleted and click the *Delete Trap* button or right click on the trap in the trap table and select option *Delete the Selected Rows*.

Yet another option in the Trap Viewer is the *ParserEditor*. The TrapViewer can filter incoming traps according to certain criteria called the parser criteria. The configuration of the criteria is made possible by using the parser editor. Refer to the ManageEngine SNMP Utilities documentation for a detailed description of the parser editor configuration and its use.

Standard SNMP Version 1 Traps

The following standard SNMP version 1 traps are supported. The descriptions are taken from RFC 1215 “Convention for defining traps for use with the SNMP”.

```
warmStart TRAP-TYPE
ENTERPRISE snmp
DESCRIPTION
"A warmStart trap signifies that the sending protocol entity is reinitializing
itself such that neither the agent configuration nor the protocol entity implemen-
tation is altered."
::= 1

linkDown TRAP-TYPE
ENTERPRISE snmp
VARIABLES   { ifIndex }
DESCRIPTION
"A linkDown trap signifies that the sending protocol entity recognizes a failure in
one of the communication links represented in the agent's configuration."
::= 2
```

Note The linkDown trap is not sent if any of the ISDN ports has gone down.

```
linkUp TRAP-TYPE
ENTERPRISE snmp
VARIABLES   { ifIndex }
DESCRIPTION
"A linkUp trap signifies that the sending protocol entity recognizes that one of
the communication links represented in the agent's configuration has come up."
::= 3
```

Note The linkUp trap is not sent if any of the ISDN ports has come up.

```
authenticationFailure TRAP-TYPE
ENTERPRISE snmp
DESCRIPTION
"An authenticationFailure trap signifies that the sending protocol entity is the
addressee of a protocol message that is not properly authenticated. While implemen-
tations of the SNMP must be capable of generating this trap, they must also be capa-
ble of suppressing the emission of such traps via an implementation-specific
mechanism."
::= 4
```

Note The authenticationFailure trap is sent after trying to access any MIB object with a SNMP community string, which does not correspond to the system setting.

```
coldStart TRAP-TYPE
ENTERPRISE snmp
DESCRIPTION
"A coldStart trap signifies that the sending protocol entity is reinitializing
itself such that the agent's configuration or the protocol entity implementation
may be altered."
::= 0
```

Note The standard SNMP version 1 trap coldStart as listed below is *not* supported. After powering up, a warmStart trap message is sent if any trap target host is defined.

SNMP Interface Traps

The Patton device sends Interface Traps (*linkUp*, *linkDown*) when the status of logical or physical interfaces change. Logical interfaces are interfaces defined in the IP context and CS context. Physical interfaces are ports.

The Patton device adds an entry to event log for each Interface Traps it sends:

```
device(cfg)#show log event

...
2002-09-06T14:54:35 : LOGINFO  : Link up on interface sip_60.
2002-09-06T14:54:35 : LOGINFO  : Link up on interface sip_30.
2002-09-06T14:54:35 : LOGINFO  : Link up on interface isdn20.
2002-09-06T14:54:38 : LOGINFO  : Link up on interface ETH00.
2002-09-06T14:54:38 : LOGINFO  : Link up on interface ETH01.
2002-09-06T14:54:39 : LOGINFO  : Link up on interface eth00.
2002-09-06T14:54:39 : LOGINFO  : Link up on interface eth01.
2002-09-06T14:56:02 : LOGINFO  : Link up on interface SLOT2:00 ISDN D
2002-09-10T14:21:20 : LOGINFO  : Link down on interface SLOT2:00 ISDN
...
```

Chapter 30 Public-Key Infrastructure (PKI)

Chapter contents

Introduction

This chapter provides an overview on how to set up the public-key infrastructure (PKI) on a Patton device. PKI deals with the creation, management and deployment of keys and certificates, which is an intricate task. Therefore, this chapter first gives an introduction on general PKI concepts, before it discusses in detail how to configure a Patton device.

Overview

As business is moving forward the expectation for secure communication over the public Internet is getting more and more paramount. There are three primary security vulnerabilities of communications over a publicly accessible network:

- Eavesdropping: an intruder captures the data transmission between two parties during communications.
- Identity theft: an intruder gains illegal access by posing as an individual who actually can access secured resources.
- Man-in-the-Middle: an intruder interrupts a dialogue and modifies the data between the two parties. The intruder could take over the entire session in the worst case.

Public Key Infrastructure (PKI) strongly reduces these risks. It provides a hierarchical framework for managing the digital security attributes of entities that will engage in secured communications.

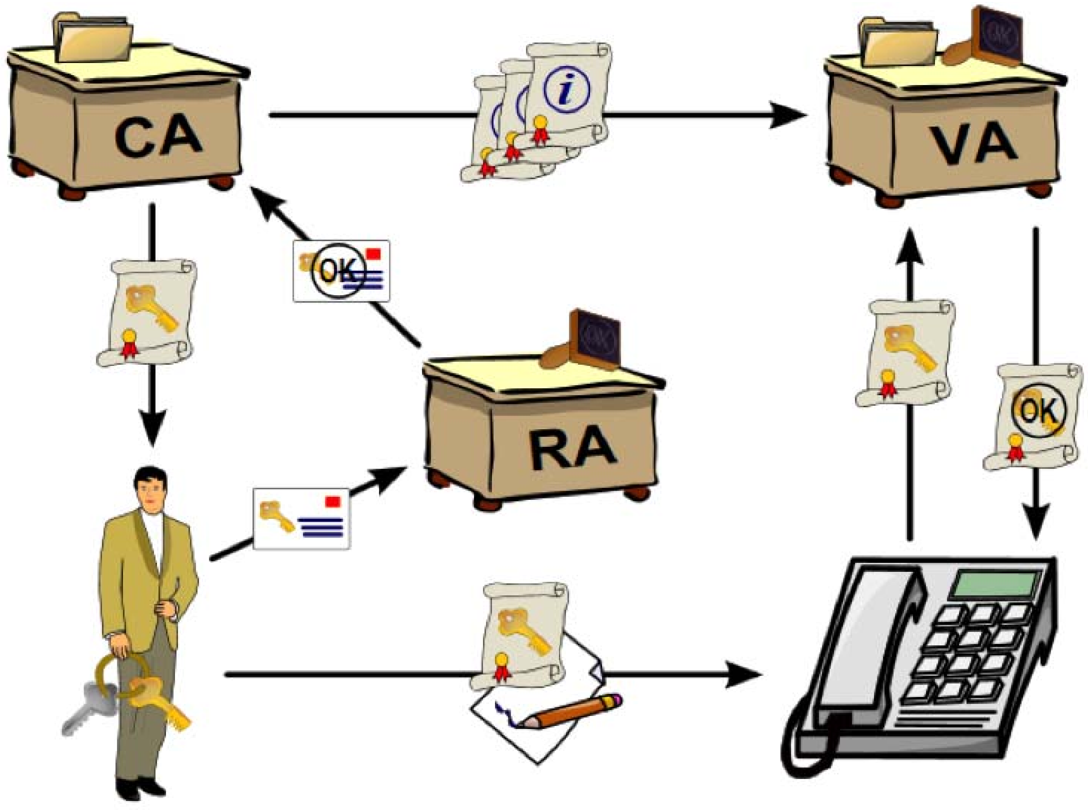


Figure 41. PKI Architecture

Architecture

A PKI consists of the following elements:

- A Certificate Authority (CA) that both issues and verifies the digital certificates. The primary role of the CA is to digitally sign and publish the public key bound to a given user. This is done using the CA's own private key, so that trust in the user's key relies on one's trust in the validity of the CA's key.
- A Registration Authority (RA) which verifies the identity of users requesting information from the CA
- Third-party Validation Authority (VA) which provides information on behalf of CA.

Symmetric encryption and the key-distribution problem

Symmetric encryption may also be referred to as shared key or shared secret encryption. In symmetric encryption, a single key is used both to encrypt and decrypt traffic. Symmetric encryption algorithms can be extremely fast, and their relatively low complexity allows for easy implementation in hardware. However, they require that all hosts participating in the encryption have already exchanged the secret key through some external means. Therefore, the user needs a secure channel to disclose these keys which will lead to asymmetric encryption.

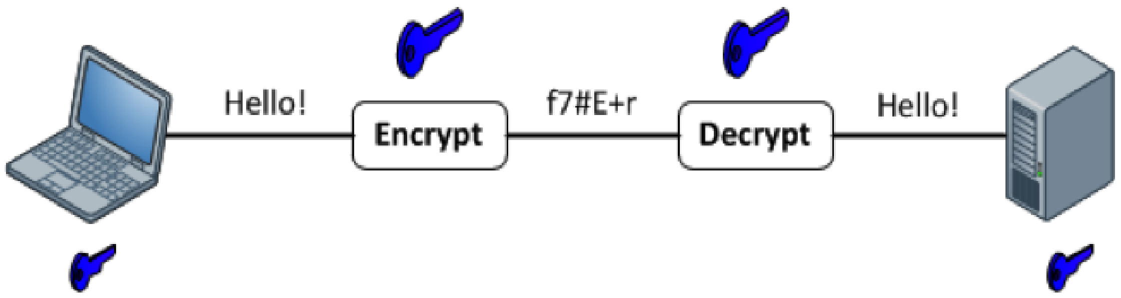


Figure 42. Symmetric Encryption

Asymmetric encryption

Compared to the symmetric way, asymmetric encryption imposes a high computational burden, and tends to be much slower. This property is less efficient when dealing with large amounts of data. However, asymmetric encryption has one advantage which is the ability to establish a secure channel over a non-secure medium (i.e., the Internet). Let's assume that two participants have decided to talk to each other securely. As a first step each of them must generate its private/public key pair.

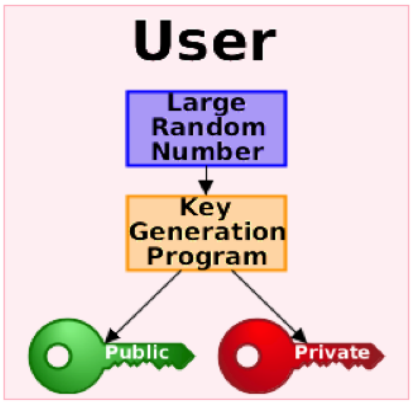


Figure 43. Private/Public Key Generation

The key generation program is fed by a large random number, which is different for each participant; therefore, the public and private keys will also differ.

Refer to figure 44 for an illustration of the usage of public/private key pair in asymmetric cryptography. The two parties will exchange their public keys (contained in the certificate), but will not disclose their private keys. The sending party will use the public key of the receiving party to encrypt message data and forward the encrypted data to the other party. The receiving party will then decrypt it with its private key. Data that is encrypted with the public key can be decrypted with the corresponding private key, and vice versa. However, data encrypted with the public key cannot be decrypted with the public key. This prevents someone from compromising the encrypted data after acquiring both public keys by sniffing on the certificate exchange.

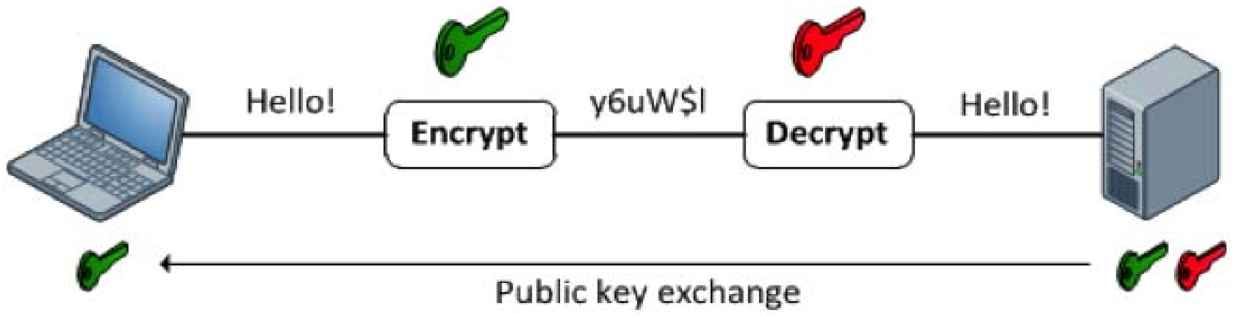


Figure 44. Asymmetric Encryption

If the secure channel is established it is possible to exchange a symmetric session key, which is used to actually encrypt and decrypt data. This allows for a more efficient communication.

CA-signed certificate enrollment

The mentioned public-key exchange happens via certificates. Enrollment is the process of obtaining these certificates.

The enrollment process looks like the following for an end host (see figure 45):

1. The end host generates a private/public key pair.

2. The end host generates a certificate request, which it forwards to the CA or RA.
3. Manual, human intervention is required to approve the enrollment request, which is received by CA or RA.
4. After the CA or RA operator approves the request, the CA or RA signs the certificate request with its own private key and returns the completed certificate to the end host.
5. The end host writes the certificate into a non-volatile storage area.

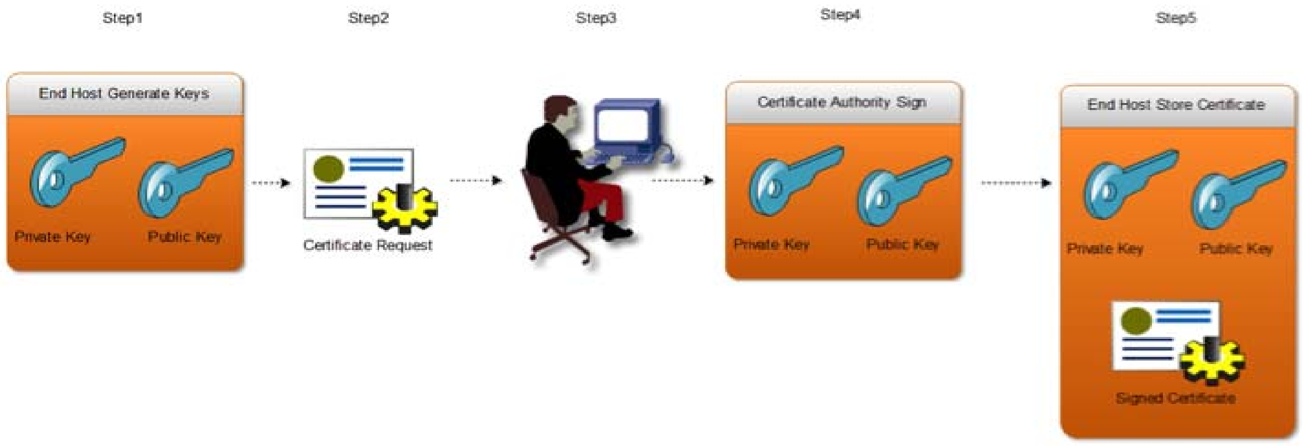


Figure 45. Certificate Enrollment Process

Example 1: Generate a private key and certificate request

This example illustrates how the above process is executed on a Patton device.

1. node(cfg)#generate pki:private-key/key1 key-length 1024
2. node(cfg)#generate pki:certificate-request/request1 private-key pki:private-key/key1 country CH state Bern locality Bern organization Patton-Inalp organization-unit RND common-name 172.16.55.41
3. node(cfg)#export pki:certificate-request/request1

Example 2: Generate all files on the CA server and import them

As an alternative, you may import the private key from a TFTP server and generate the certificate request offline.

1. node(cfg)#copy tftp://server-ip-address/key1 pki:private-key/key1
2. Generate the…
3. …certificate request offline.
4. This certificate request has to be manually signed by the Certificate Authority.
5. *node*(cfg)#copy tftp://server-ip-address/cert1 pki:own-certificate/cert1

Self-signed certificate enrollment

The mentioned public key exchange happens via certificates. Enrollment is the process of obtaining these certificates.

In some cases it is enough to generate a specific self-signed certificate. In a web-of-trust certificate scheme there is no central CA, and so identity certificates for each user can be self-signed. As the name tells it is an identity certificate that is signed by the same entity whose identity it certifies. The generation process looks like the following for an end host (see figure 46):

1. The end host generates a private/public key pair.
2. The end host generates a certificate request.
3. The end host signs the certificate request with its own private key and writes the certificate into a non-volatile storage area.

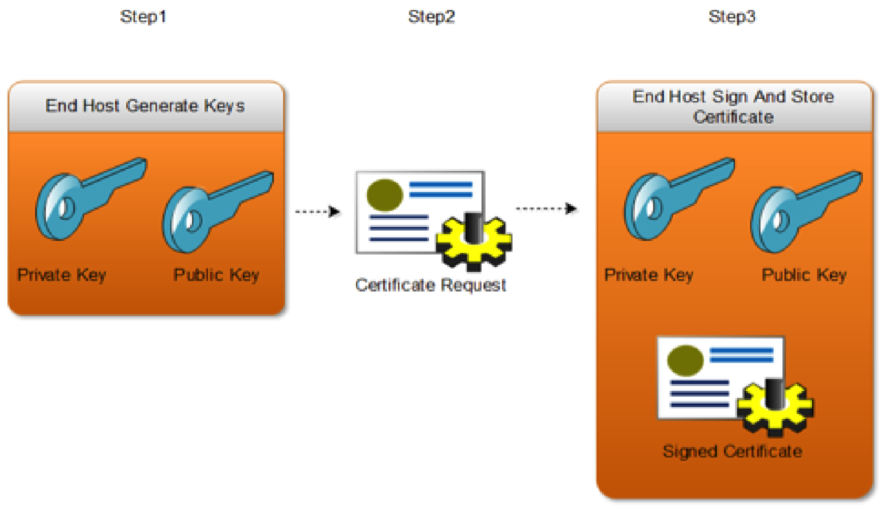


Figure 46. Self-signed Certificate Process

Example 1: Generate a private key and self-signed certificate

This example illustrates how the above process is executed on a Patton device.

1. *node*(cfg)#generate pki:private-key/key1 key-length 1024
2. node(cfg)#generate pki:certificate-request/request1 private-key pki:private-key/key1 country CH state Bern locality Bern organization Patton-Inalp organization-unit RND common-name 172.16.55.41
3. *node*(cfg)#generate pki:own-certificate/cert1 private-key pki:private-key/key1 validity-period 3650

Example 2: Import a private key and a self-signed certificate

As an alternative, you may import the private key and self-signed certificate from a TFTP server.

1. *node*(cfg)#copy tftp://server-ip-address/key1 pki:private-key/key1
2. create and sign the certificate request offline
3. *node*(cfg)#copy tftp://server-ip-address/cert1 pki:own-certificate/cert1

Configuration task list

This section introduces and discusses all PKI-related commands in Trinity.

Private-key handling

Asymmetric encryption requires a private and a public key for encryption and decryption. They can only be generated and deleted together. With the **generate** command it is possible to generate an RSA key with a specified length. A longer modulus length corresponds to a higher level of security but requires more computational resources for key generation and connection handshaking. Keys are mainly used to generate or sign certificate requests.

In order to make the Private Key Infrastructure easier to use a couple of default files will be generated at the first startup. One of them is a private key named *DEFAULT* with a modulus length of 512 bits. The file is generated only once, it is immutable and can be used for basic scenarios (see "Applications" on page 609 for example).

Mode: Administrator exec

Step	Command	Purpose
1	***node*(cfg)#generate pki:private-key/name [key-length <512|768|1024|2048>]**	Generates a private key. Note that it implicitly generates a public key as well.
2	***node*(cfg)#copy tftp://server-ip-address/filename pki:private-key/name**	Imports a private key. Note that it implicitly imports a public key as well. A content check is applied: If the file is not a valid private key it will not be imported.
3	***node*(cfg)#erase pki:private-key/name**	Erases a private key.

Example: Create a private key locally

```
node>enable
node#configure
node(cfg)#generate pki:private-key/key1 key-length 1024
Generating RSA private key, 1024 bit long modulus
................................++++++
.++++++
e is 65537 (0x10001)
writing RSA key
```

Example: Import a private key from a TFTP server

```
node>enable
node#configure
node(cfg)#copy tftp://172.16.55.1/key1 pki:private-key/key2
```

Example: Erase the two private keys

```
node>enablenode#configure
node(cfg)#erase pki:private-key/key2
node(cfg)#erase pki:private-key/key1
```

Public-key handling

Asymmetric encryption requires a private and a public key for encryption and decryption. They can only be generated and deleted together with the **generate** command introduced above.

Mode: Administrator exec

Step	Command	Purpose
1	***node*(cfg)#export pki:public-key/ name**	Displays a public key in Base64 format, ready to be copied from the terminal for export.
2	***node*(cfg)#copy pki: public -key/ name** tftp://*server-ip-address/filename*	Exports a public key to a TFTP server.

Example: Display a public key in Base64 format

```
node>enable
node#configure
node(cfg)#export pki:public-key/key1
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCf7r3uLPuPbsVPX0mSOvV3Og+c
WZhf99M/OsZcstpZqxAlRq3rERiueC3SSDAq2lfEC+3Xgcs/uP/nfTL0IRPzvDpu
By7e2uHUnyNJD4HtirH1OO6+CAa0wTFdvlCtv/Mya1oq7tLulDx+KNVjI61YLKGw
jkfoc69A8bIs4Sh6+wIDAQAB
-----END PUBLIC KEY-----
```

Example: Export a public key to a TFTP server

```
node>enable
node#configure
node(cfg)#copy pki:public-key/key1 tftp://172.16.55.1/key1
```

Certificate-request handling

The key exchange happens via certificates. In order to create a certificate we have to generate a certificate request. All the parameters of the **generate** command are mandatory and have the following meaning:

- **Private-key:** The request has to store a key which will be sent to the peer, which is the public key that corresponds to the specified private key. This public key will be the decryption key of the peer. If no private key is specified, the *DEFAULT* private key will be used.
- **country:** The certificate issuer's country of residence. The country must be a two-letter code. (Example: CH) If nothing is set then empty string will be used.
- **state:** The certificate owner's state of residence. If nothing is set then empty string will be used.
- **locality:** The certificate issuer's locality. If nothing is set then empty string will be used.
- **organization:** The organization to which the certificate issuer belongs. If nothing is set then empty string will be used.
- **organization-unit:** The name of the organizational unit to which the certificate issuer belongs. If nothing is set then empty string will be used
- **Common-name:** The certificate owner's common name. This has to be the IP address or the fully qualified DNS domain name ("im.example.org", "mail.example.net", and "www.example.com", respectively) of the local SIP gateway. The peer side can check whether these parameters match. If nothing was is then empty string will be used.

Mode: Administrator exec

Step	Command	Purpose
1	***node*(cfg)#generate pki:certificate-request/ name private-key <pki:private-key/name> country <country> state <state> locality <locality> organization <organization> organization-unit <organization-unit> common-name <common-name>**	Generates a certificate request.
2	***node*(cfg)#export pki: certificate-request/ name**	Displays a certificate request in Base64 format, ready to be copied from the terminal for export.
3	***node*(cfg)#show pki: certificate-request/name**	Shows a certificate request logical content.
4	***node*(cfg)#copy pki: certificate-request/name tftp://server-ip-address/filename**	Exports a certificate request to a TFTP server.
5	***node*(cfg)#copy tftp://server-ip-address/filename pki: certificate-request/name**	Imports a certificate request from a TFTP server. A content check will be applied: If the file is not a valid certificate request then it will not be imported.
6	***node*(cfg)#erase pki: certificate-request/name**	Erases a certificate request.

Example: Generate a certificate request locally

```
node>enable
node#configure
node(cfg)#generate pki:certificate-request/request1 private-key pki:private-key/
key1 country CH state Bern locality Bern organization Patton-Inalp organization-
unit RND common-name 172.16.55.41
```

Example: Display a certificate request in Base64 format

```
node>enable
node#configure
node(cfg)#export pki:certificate-request/request1
-----BEGIN CERTIFICATE REQUEST-----
IIBoDCCAQkCAQAwYDELMAkGA1UEBhMCQ0gxDTALBgNVBAgMBEJlcm4xDTALBgNV
AcMBEJlcm4xDjAMBgNVBAoMBUluYWxwMQwwCgYDVQQLDANSTkQxFTATBgNVBAMM
DE3Mi4xNi41NS40MTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAn+697iz7
27FT19Jkjr1dzoPnFmYX/fTPzrGXLLaWasQJUat6xEYrngt0kgwKtpXxAvt14HL
7j/530y9CET87w6bgcu3trh1J8jSQ+B7Yqx9TjuvggGtMExXb5Qrb/zMmtaKu7S
Q8fijVYyOtWCyhsI5H6HOvQPGyLOEoevsCAwEAAaAAMA0GCSqGSIb3DQEBBQUA
4GBAGh3UJjIdJWyo+YpPmjIZffPBfav4JeqFNrHs2tWAep7lbsD2dYZ+pzFByVc
u0U5Ioaptk7VmRvTRlDQpoYED/KntyXDv3Sggb3Mf7cGq3xTZloqhXZNzN3PbJl
kf16+4zB7H1gfd//tdEhEUCJVMAWsYUWD85ur6yH8uolItv
-----END CERTIFICATE REQUEST-----
```

Example: Display the logical content of a certificate request

```
node>enable
node#configure
```

```
node(cfg)#show pki:certificate-request/request1
Certificate Request:
   Data:
       Version: 0 (0x0)
       Subject: C=CH, ST=Bern, L=Bern, O=Patton-Inalp, OU=RND, CN=172.16.55.41
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
               Public-Key: (1024 bit)
               Modulus:
                   00:9f:ee:bd:ee:2c:fb:8f:6e:c5:4f:5f:49:92:3a:
                   f5:77:3a:0f:9c:59:98:5f:f7:d3:3f:3a:c6:5c:b2:
                   da:59:ab:10:25:46:ad:eb:11:18:ae:78:2d:d2:48:
                   30:2a:da:57:c4:0b:ed:d7:81:cb:3f:b8:ff:e7:7d:
                   32:f4:21:13:f3:bc:3a:6e:07:2e:de:da:e1:d4:9f:
                   23:49:0f:81:ed:8a:b1:f5:38:ee:be:08:06:b4:c1:
                   31:5d:be:50:ad:bf:f3:32:6b:5a:2a:ee:d2:ee:94:
                   3c:7e:28:d5:63:23:ad:58:2c:a1:b0:8e:47:e8:73:
                   af:40:f1:b2:2c:e1:28:7a:fb
               Exponent: 65537 (0x10001)
       Attributes:
           a0:00
   Signature Algorithm: sha1WithRSAEncryption
       68:77:50:98:c8:74:95:b2:a3:e6:29:3e:68:c8:65:f7:cf:05:
       f6:af:e0:97:aa:14:da:c7:b3:6b:56:01:ea:7b:95:bb:03:d9:
       d6:19:fa:9c:c5:07:25:5c:a6:ed:14:e4:8a:1a:a6:d9:3b:56:
       64:6f:4d:19:43:42:9a:18:10:3f:ca:9e:dc:97:0e:fd:d2:82:
       06:f7:31:fe:dc:1a:ad:f1:4d:99:68:aa:15:d9:37:33:77:3d:
       b2:65:fa:47:f5:eb:ee:33:07:b1:f5:81:f7:7f:fe:d7:44:84:
       45:02:25:53:00:5a:c6:14:58:3f:39:ba:be:b2:1f:cb:a8:94:
       8b:6f
```

Example: Export a certificate request to a TFTP server

```
node>enable
node#configure
node(cfg)#copy pki:certificate-request/request1 tftp://172.16.55.1/request1
```

Example: Import a certificate request from a TFTP server

```
node>enable
node#configure
node(cfg)#copy tftp://172.16.55.1/request1 pki:certificate-request/request2
```

Example: Erase the two certificate requests

```
node>enable
node#configure
node(cfg)#erase pki:certificate-request/request2
node(cfg)#erase pki:certificate-request/request1
```

Own-certificate handling

A certificate chain is transmitted to a remote peer when this peer wants to authenticate the local user/device. A certificate chain is formed from the following certificates:

- A personal certificate, which identifies the local user or device
- Zero or more intermediate certificates, which are certificates that identify intermediary Certificate Authorities between the root certificate and the personal certificate.

- Zero or one root certificate, provided by a trusted third-party Certificate Authority (CA)

The ordered list of certificates, starting from the personal certificate and ending at the root CA, usually denotes a delegation of trust where the root CA trusts intermediate CA1, which in turns trusts intermediate CA2, which in turns trusts intermediate CA3 and so on, up to the intermediate CA that has issued the personal certificate to the local user or device, which trusts the user's/device's identity. With the following commands only single certificates can be handled; the chain of certificates must be formed later in the TLS profile.

Mode: Administrator exec

Step	Command	Purpose
1	***node*(cfg)#generate pki:own-certificate/<name> pki: certificate-request/<name> [private-key pki:private-key/<name>] [validity-period <days>]**	Generates an own certificate.
2	***node*(cfg)#export pki:own-certificate/name**	Displays an own certificate in Base64 format, ready to be copied from the terminal for export.
3	***node*(cfg)#show pki:own-certificate/name**	Shows an own certificate logical content.
4	***node*(cfg)#copy pki:own-certificate/name tftp://server-ip-address/filename**	Exports an own certificate to a TFTP server.
5	***node*(cfg)#copy tftp://server-ip-address/filename pki:own-certificate/name**	Imports an own certificate from a TFTP server. A content check will be applied: If the file is not a valid certificate then it will not be imported.
6	***node*(cfg)#erase pki:own-certificate/name**	Erases an own certificate.

Example: Generate a self-signed certificate locally

```
node>enable
node#configure
node(cfg)#generate pki:own-certificate/cert1 pki:certificate-request/request1 pri-
vate-key pki:private-key/key1 validity-period 365
Signature ok
subject=/C=CH/ST=Bern/L=Bern/O=Patton-Inalp/OU=RND/CN=172.16.55.41
Getting Private key
```

Example: Display a certificate in Base64 format

```
node>enablenode#configurenode(cfg)#export pki:own-certificate/cert1
-----BEGIN CERTIFICATE-----
MIICNzCCAaACCQCUWsjV9Z+l7zANBgkqhkiG9w0BAQUFADBgMQswCQYDVQQGEwJDSDENMAsGA1UECAwEQm
VybjENMAsGA1UEBwwEQmVybjEOMAwGA1UECgwFSW5hbHAxDDAKBgNVBAsMA1JORDEVMBMGA1UEAwwMMTcy
LjE2LjU1LjQxMB4XDTEzMDcwMjA4MjI1OFoXDTE0MDcwMjA4MjI1OFowYDELMAkGA1UEBhMCQ0gxDTALBg
NVBAgMBEJlcm4xDTALBgNVBAcMBEJlcm4xDjAMBgNVBAoMBUluYWxwMQwwCgYDVQQLDANSTkQxFTATBgNV
BAMMDDE3Mi4xNi41NS40MTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAn+697iz7j27FT19Jk-
jr1dzoPnFmYX/fTPzrGXLLaWasQJUat6xEYrngt0kgw
KtpXxAvt14HLP7j/530y9CET87w6bgcu3trh1J8jSQ+B7Yqx9TjuvggGtMExXb5Q
rb/zMmtaKu7S7pQ8fijVYyOtWCyhsI5H6HOvQPGyLOEoevsCAwEAATANBgkqhkiG
9w0BAQUFAAOBgQAY2vYzZWXvkPspAzpCmxwC/YQkvfIhzAW8cbXpLX7I+pbQIYhr
N5BbyLGVTk7WWlw42jmsP+zZNwHqpJdCsd4kxOFmNWIBZSY2b0oiiX4rmgMCxohf
uHk0tzKflHVzMVSPgiaCfeAZ6hFQU8MMe8YxJ8hPc/pQYu5aneKV4zAo3w==
-----END CERTIFICATE-----
```

Example: Display the logical content of a certificate

```
node>enable
node#configure
node(cfg)#show pki:own-certificate/cert1
Certificate:
   Data:
       Version: 1 (0x0)
       Serial Number:
           94:5a:c8:d5:f5:9f:a5:ef
       Signature Algorithm: sha1WithRSAEncryption
       Issuer: C=CH, ST=Bern, L=Bern, O=Patton-Inalp, OU=RND, CN=172.16.55.41
       Validity
           Not Before: Jul  2 08:22:58 2013 GMT
           Not After : Jul  2 08:22:58 2014 GMT
       Subject: C=CH, ST=Bern, L=Bern, O=Patton-Inalp, OU=RND, CN=172.16.55.41
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
               Public-Key: (1024 bit)
               Modulus:
                   00:9f:ee:bd:ee:2c:fb:8f:6e:c5:4f:5f:49:92:3a:
                   f5:77:3a:0f:9c:59:98:5f:f7:d3:3f:3a:c6:5c:b2:
                   da:59:ab:10:25:46:ad:eb:11:18:ae:78:2d:d2:48:
                   30:2a:da:57:c4:0b:ed:d7:81:cb:3f:b8:ff:e7:7d:
                   32:f4:21:13:f3:bc:3a:6e:07:2e:de:da:e1:d4:9f:
                   23:49:0f:81:ed:8a:b1:f5:38:ee:be:08:06:b4:c1:
                   31:5d:be:50:ad:bf:f3:32:6b:5a:2a:ee:d2:ee:94:
                   3c:7e:28:d5:63:23:ad:58:2c:a1:b0:8e:47:e8:73:
                   af:40:f1:b2:2c:e1:28:7a:fb
               Exponent: 65537 (0x10001)
   Signature Algorithm: sha1WithRSAEncryption
       18:da:f6:33:65:65:ef:90:fb:29:03:3a:42:9b:1c:02:fd:84:
       24:bd:f2:21:cc:05:bc:71:b5:e9:2d:7e:c8:fa:96:d0:21:88:
       6b:37:90:5b:c8:b1:95:4e:4e:d6:5a:5c:38:da:39:ac:3f:ec:
       d9:37:01:ea:a4:97:42:b1:de:24:c4:e1:66:35:62:01:65:26:
       36:6f:4a:22:89:7e:2b:9a:03:02:c6:88:5f:b8:79:34:b7:32:
       9f:94:75:73:31:54:8f:82:26:82:7d:e0:19:ea:11:50:53:c3:
       0c:7b:c6:31:27:c8:4f:73:fa:50:62:ee:5a:9d:e2:95:e3:30:
       28:df
```

Example: Export a certificate to a TFTP server

```
node>enable
node#configure
node(cfg)#copy pki:own-certificate/cert1 tftp://172.16.55.1/cert1
```

Example: Import a certificate from a TFTP server

```
node>enable
node#configure
node(cfg)#copy tftp://172.16.55.1/cert1 pki:own-certificate/cert2
```

Example: Erase the two certificates

```
node>enable
node#configure
node(cfg)#erase pki:own-certificate/cert2
ode(cfg)#erase pki:own-certificate/cert1
```

Trusted-certificate handling

Root certificates are certificates that are trusted by the device. If the peer presents a certificate that does inherit from one of them, the connection-handshaking is accepted.

Mode: Administrator exec

Steps	Command	Purpose
1	***node*(cfg)#export pki:trusted-certificate/ name**	Displays a trusted certificate in Base64 format, ready to be copied from the terminal for export.
2	***node*(cfg)#show pki:trusted-certificate/ name**	Shows a trusted certificate logical content.
3	***node*(cfg)#copy pki:trusted-certificate/ name tftp://server-ip-address/filename**	Exports a trusted certificate to a TFTP server.
4	***node*(cfg)#copy tftp://server-ip-address/ filename pki:trusted-certificate/name**	Imports a trusted certificate from a TFTP server. A content check will be applied: If the file is not a valid certificate then it will not be imported.
5	***node*(cfg)#erase pki:trusted-certificate/ name**	Erases a trusted certificate.

Example: Imports a trusted certificate from a TFTP server

```
node>enable
node#configure
node(cfg)#copy tftp://172.16.55.1/cert1 pki:trusted-certificate/cert1
```

Example: Exports a trusted certificate to a TFTP server

```
node>enablenode#configure
node(cfg)#copy pki:trusted-certificate/cert1 tftp://172.16.55.1/cert2
```

Example: Display a trusted certificate in Base64 format

```
node>enablenode#configure
node(cfg)#export pki:trustedertificate/cert1
-----BEGIN CERTIFICATE-----
MIICNzCCAaACCQDSauxdaSaSLTANBgkqhkiG9w0BAQUFADBgMQswCQYDVQQGEwJDSDENMAsGA1UECAwEQm
VybjENMAsGA1UEBwwEQmVybjEOMAwGA1UECgwFSW5hbHAxDDAKBgNVBAsMA1JORDEVMBMGA1UEAwwMMTcy
LjE2LjU1LjQxMB4XDTEzMDcwMjA4MzQxOFoXDTE0MDcwMjA4MzQxOFowYDELMAkGA1UEBhMCQ0gxDTALBg
NVBAgMBEJlcm4xDTALBgNVBAcMBEJlcm4xDjAMBgNVBAoMBUluYWxwMQwwCgYDVQQLDANSTkQxFTATBgNV
BAMMDDE3Mi4xNi41NS40MTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAn+697iz7j27FT19Jk-
jr1dzoPnFmYX/fTPzrGXLLaWasQJUat6xEYrngt0kgw
KtpXxAvt14HLP7j/530y9CET87w6bgcu3trh1J8jSQ+B7Yqx9TjuvggGtMExXb5Q
rb/zMmtaKu7S7pQ8fijVYyOtWCyhsI5H6HOvQPGyLOEoevsCAwEAATANBgkqhkiG
9w0BAQUFAAOBgQBP7KvsIbiJmzoEQvZaZDlefbCdPIZdsAshcLwP7jE9bGC1knbv
aAcyB6n1Bt1lUQXDqbF0GhcauEEKBA3O7I1njWyeebg58j5iKq89FGCaH1sQhXKO
z61A8YPl2gHYQcxCrZX7g9aQ9hwCc2OG+Mg+h4wpxbiOof2qM/3muk1FGQ==
-----END CERTIFICATE-----
```

Example: Shows the logical content of a trusted certificate

```
node>enablenode#configure
node(cfg)#show pki:trustedertificate/cert1
certificate:
   Data:
       Version: 1 (0x0)
       Serial Number:
```

```
                d2:6a:ec:5d:69:26:92:2d
            Signature Algorithm: sha1WithRSAEncryption
            Issuer: C=CH, ST=Bern, L=Bern, O=Patton-Inalp, OU=RND, CN=172.16.55.41
            Validity
                Not Before: Jul  2 08:34:18 2013 GMT
                Not After : Jul  2 08:34:18 2014 GMT
            Subject: C=CH, ST=Bern, L=Bern, O=Patton-Inalp, OU=RND, CN=172.16.55.41
            Subject Public Key Info:
                Public Key Algorithm: rsaEncryption
                    Public-Key: (1024 bit)
                    Modulus:
                        00:9f:ee:bd:ee:2c:fb:8f:6e:c5:4f:5f:49:92:3a:
                        f5:77:3a:0f:9c:59:98:5f:f7:d3:3f:3a:c6:5c:b2:
                        da:59:ab:10:25:46:ad:eb:11:18:ae:78:2d:d2:48:
                        30:2a:da:57:c4:0b:ed:d7:81:cb:3f:b8:ff:e7:7d:
                        32:f4:21:13:f3:bc:3a:6e:07:2e:de:da:e1:d4:9f:
                        23:49:0f:81:ed:8a:b1:f5:38:ee:be:08:06:b4:c1:
                        31:5d:be:50:ad:bf:f3:32:6b:5a:2a:ee:d2:ee:94:
                        3c:7e:28:d5:63:23:ad:58:2c:a1:b0:8e:47:e8:73:
                        af:40:f1:b2:2c:e1:28:7a:fb
                    Exponent: 65537 (0x10001)
        Signature Algorithm: sha1WithRSAEncryption
            4f:ec:ab:ec:21:b8:89:9b:3a:04:42:f6:5a:64:39:5e:7d:b0:
            9d:3c:86:5d:b0:0b:21:70:bc:0f:ee:31:3d:6c:60:b5:92:76:
            ef:68:07:32:07:a9:f5:06:dd:65:51:05:c3:a9:b1:74:1a:17:
            1a:b8:41:0a:04:0d:ce:ec:8d:67:8d:6c:9e:79:b8:39:f2:3e:
            62:2a:af:3d:14:60:9a:1f:5b:10:85:72:8e:cf:ad:40:f1:83:
            e5:da:01:d8:41:cc:42:ad:95:fb:83:d6:90:f6:1c:02:73:63:
            86:f8:c8:3e:87:8c:29:c5:b8:8e:a1:fd:aa:33:fd:e6:ba:4d:
            45:19

node>enable
node#configure
node(cfg)#erase pki:trustedertificate/cert1
```

Generated default files

In order to make the Private Key Infrastructure easier to use a couple of default files will be generated at the first startup. The files are generated only once, are immutable, and can be used in basic scenarios (see "Applications" on page 609 for example). The following default files are being generated:

- Private and public key
  - Name: DEFAULT
  - Key length: 512
- Own certificate
  - Name: DEFAULT
  - Self-signed with private key DEFAULT
  - Validity: 1000 years

Chapter 31 Quality of Service (QoS) Overview

Chapter contents

Introduction

This chapter provides an overview of the core principles of Patton's Quality-of-Service architecture. This chapter includes the following sections:

- "Packet Classification" on page 316
- "Type-of-Service (TOS)/Class-of-Service (CoS) Mapping" on page 317

QoS in networking refers to the capability of the network to provide a better service to selected network traffic. In the context of VoIP, the primary issue is to control the coexistence of voice and data packets such that voice packets are delayed as little as possible.

Currently, Patton devices do not provide a means of scheduling voice packets differently within Trinity. However, Trinity is able to convert external QoS-tags (IP TOS header or 802.1pq CoS header fields) between networks of different QoS realms. (Those tags may have a different meaning in different QoS realms.)

Packet Classification

Several Trinity services, such as access control lists and policy routing, need to distinguish between different types of packets. We refer to those types as "traffic classes". You can think of the traffic-class as if every packet in the Trinity device has a tag attached to it on which the classification can be noted. Trinity's classifier can be used to apply such a traffic-class tag to some types of packets based on their protocol headers (e.g. source/destination address and port, packet length, IP TOS field etc.). Thus conceptually, as depicted in figure 47, the classifier groups packet flows from different originating hosts but of the same service (e.g. data, voice, etc.) into virtual traffic class flows.

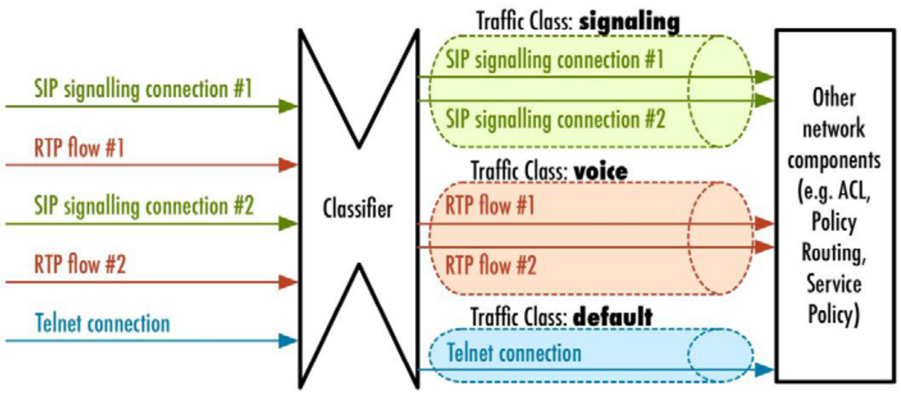

SIP signalling connection #1
RTP flow #1
SIP signalling connection #2
RTP flow #2
Telnet connection
Classifier
Traffic Class: signaling
SIP signalling connection #1
SIP signalling connection #2
Traffic Class: voice
RTP flow #1
RTP flow #2
Traffic Class: default
Telnet connection
Other network components (e.g. ACL, Policy Routing, Service Policy)

Figure 47. Conceptual view on the classifier; it groups packet flows of the same service

If not configured otherwise, the traffic-class tag is set to *DEFAULT* for all received packets. In contrast, locally-generated voice and data packets are tagged with the traffic class *LOCAL-VOICE* and *LOCAL-DEFAULT*, respectively.

Type-of-Service (TOS)/Class-of-Service (CoS) Mapping

The traffic-class tags exist only inside a Trinity device, but layer 2 priority bits (802.1pq class-of-service) and IP header type-of-service bits (TOS field) in received packets can be used to mark specific packet types for the other network nodes in a QoS-enabled network. The figure below shows that several Trinity services are involved in mapping TOS and CoS header fields to traffic classes and vice versa while a packet is routed through the device.

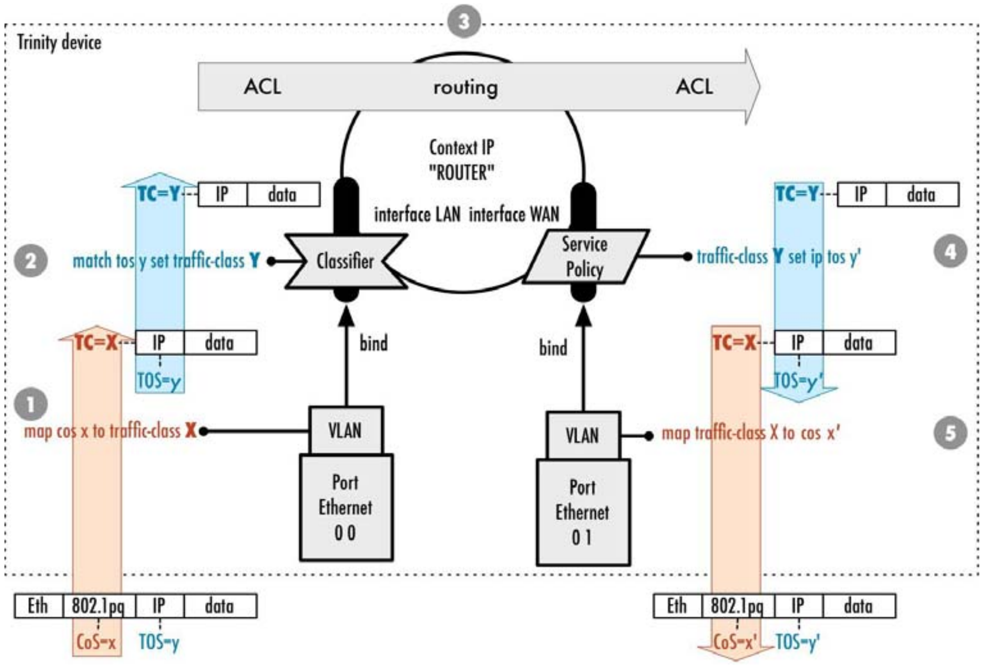


Figure 48. Mapping TOS/CoS to traffic-class and vice-versa

1. The **map** command on a VLAN port can be used to map an 802.1pq class-of-service value (CoS field) to an internal traffic class. This command is explained in more detail in Section "Configuring a VLAN" on page 160 in Chapter 14, "Ethernet Port Configuration" on page 156.
2. The **bind** command on a VLAN port determines over which IP interface a packet reaches Trinity's IP router. The classifier that is attached to an IP interface can be used to map an IP type-of-service value (TOS field) to an internal traffic class. Note that this traffic class overwrites the traffic-class set on the VLAN port (if any). Consult Chapter 34, "Classifier Configuration" on page 341 for more information about how to set up a classifier.
3. Several network services such as Access Control Lists (ACL) or policy routing may use the internal traffic-class tag to treat packets of different services differently. The ACL, for example, may drop all packets belonging to a certain traffic class; the **route** command on an IP interface can be used to select a different routing table for other traffic classes. See Chapter 33, "Access Control List Configuration" on page 331 or Chapter 23, "IP Routing" on page 248 to learn how to configure these services.
4. After the router has found the IP interface over which the packet has to be sent, the packet traverses multiple services that may again inspect and manipulate the packet (see Chapter 21, "IP Context Overview" on page 229 for an overview). The service-policy profile is one of those services. It can be used to map an

internal traffic-class back to an IP type-of-service value (TOS field) (see 35, "Service Policy Configuration" on page 349 for more details).

5. Finally, if the packet leaves the device over a VLAN port, the **map** command may be used to convert the internal traffic class to a 802.1pg class-of-service value (CoS field) (see Chapter 14, "Ethernet Port Configuration" on page 156).

Chapter 32 Profile Service-Policy Configuration

Chapter contents

Introduction

This chapter describes how to use and configure service-policy profiles. Service-policy profiles can be bound to individual IP interfaces to map internal traffic classes to QoS-related IP header fields. For an overview of Trinity's Quality-of-Service architecture and the meaning of traffic classes, please refer to Chapter 31, "Quality of Service (QoS) Overview" on page 315.
This chapter includes the following sections:

- Quick references
- Assigning bandwidth to traffic classes
- Service Policy configuration task list

QoS in networking refers to the capability of the network to provide a better service to selected network traffic. In the context of VoIP, the primary issue is to control the coexistence of voice and data packets such that voice packets are delayed as little as possible. This chapter shows you how to configure Trinity to best use the access link.
In many applications you can gain a lot by applying the minimal configuration found in the quick reference section, but read sections "Applying Scheduling at the Bottleneck" and "Using Traffic Classes" first to understand the paradox of why we apply a rate-limit to reduce delay and what a "traffic-class" means.

Applying Scheduling at the Bottleneck

When a Patton Device acts as an access router and voice gateway, sending voice and data packets to the Internet, the access link is the point where intelligent use of scarce resources really makes a difference. Frequently, the access link modem is outside of the Patton Device and the queueing would happen in the modem, which does not distinguish between voice and data packets. To improve QoS, you can configure the Patton Device to send no more data to the Internet than the modem can carry. This keeps the modem's queue empty and gives the Patton Device control over which packet is sent over the access link at what time.

Using Traffic Classes

The Service Policy needs to distinguish between different types of packets. We refer to those types as "traffic-classes". You can think of the traffic-class as if every packet in the Patton Device has a tag attached to it on which the classification can be noted. The classifier can be used to apply such a traffic-class name to some type of packet based on its IP-header filtering capabilities. The traffic-class tags exist only inside the Patton Device, but layer 2 priority bits (802.1pq class-of-service) and IP header type-of-service bits (TOS field) can be used to mark a specific packet type for the other network nodes. By default the traffic-class tag is empty. Only two types of packets are automatically marked by the Software: voice packets and data packets originated from the Patton Device itself are marked as "local-voice" and "local-default" respectively.

Patton DownStreamQoS™

For traffic flowing downstream from the Internet, traditional access routers cannot control the volume nor the sending users. Although downstream traffic is usually requested and initiated by users on the local network (LAN), controlling these upstream requests is not sufficient to limit the downstream traffic effectively because one request may trigger a server to send an entire file over the downstream path. The ISP's edge router commonly responds to a packet rate that exceeds the link bandwidth by discarding VoIP packets with the same probability as any other packet type. These factors, or a combination of them, may degrade voice quality to a degree that users find objectionable. Unlike data traffic -which can be retransmitted- real-time voice traffic is vulnerable to packet loss since it leads to audible interruption.

To resolve the problem of degraded voice quality for incoming traffic, Patton has devised a leadingedge technology called DownStreamQoS™. Within the Patton Device deployed at the customer premise, DownStreamQoS dynamically creates a virtual bottleneck against the incoming packet stream. This bottleneck can throttle non-real-time traffic, preventing the edge router from blocking or impeding voice traffic, to ensure voice or other real-time packets are transmitted freely downstream. DownStreamQoS exploits flow-control mechanisms within the TCP standard to create the bottleneck. Because 80% of Internet traffic is transported via TCP, DownStreamQoS is especially effective. See section Configure DownStreamQoS™ below.

Introduction to Scheduling

Scheduling essentially means to determine the order in which packets of the different traffic-classes are served. The following sections describe the ways this arbitration can be done.

Priority

One way of ordering packets is to give priority to one traffic-class and to serve the other trafficclasses when the first has nothing to send. Trinity uses the priority scheme to make sure that voice packets generated by the Patton Device will experience as little delay as possible. Voice packets can receive this treatment because they will not use up the entire bandwidth.

Weighted fair queuing (WFQ)

This arbitration method assures a given minimal bandwidth for each source. An example: you specify that traffic-class A gets three times the bandwidth of traffic-class B. So A will get a minimum of 75% and B will get a minimum of 25% of the bandwidth. But if no class A packets are waiting B will get 100% of the bandwidth. Each traffic-class is in fact assigned a relative weight, which is used to share the bandwidth among the currently active classes. Patton recommends that you specify the weight as percent which is best readable.

Shaping

There is another commonly used way to assign bandwidth. It is called shaping and it makes sure that each traffic-class will get just as much bandwidth as configured and not more. This is useful if you have subscribed to aservice that is only available for a limited bandwidth e.g. low delay. When connecting the Patton Device to a Diff-Serv network shaping might be a required operation.

Handling of bursts

For shaping there is a variation of the scheduler that allows to specify if a traffic class may temporarily receive a higher rate as long as the average stays below the limit. This burstiness measure allows the network to explicitly assign buffers to bursty sources. When you use shaping on the access link the shaper sometimes has the problem that multiple sources are scheduled for the same time - and therefore some of them will be served too late. If the rate of every source had to strictly obey its limit, all following packets would also have to be delayed by the same amount, and further collisions would reduce the achieved rate even further. To avoid this effect, the Trinity shaper assumes that the burstiness needed for sources to catch up after collisions is implicitly allowed. Future versions of Trinity might allow setting the burst rate and bursting size if more control over its behavior is considered necessary.

Hierarchy

An arbiter can either use wfq or shaping to determine which source to serve next. If you want the scheduler to follow a combination of decision criteria you can combine different schedulers in hierarchy to do a multi-level arbitration. Hierarchical scheduling is supported in Trinity with service policy profiles used inside service policy profiles. In Figure 23 an example of hierarchical scheduling is illustrated. The 1st level arbiter Level_1 uses weighted fair queuing to share the bandwidth among source classes VPN, Web and incorporates the traffic from the 2nd level arbiter Low_Priority, which itself uses shaping to share the bandwidth among source classes Mail and Default.

Quick References

The following sections provide a minimal "standard" service-policy configuration for the case where voice and data share a (DSL/cable) modem link.

Setting the Modem Rate

To match the voice and data multiplexing to the capacity of the access link is the most common application of the Trinity service-policy.

1. Create a minimal profile.

```
profile service-policy MODEM-512
rate-limit 512 overhead 30 atm
source traffic-class local-voice
priority
```

2. Apply the profile just created to the interface connected to the modem.

```
context ip
interface WAN
use profile service-policy MODEM-512 out
```

Some explanations:

- "MODEM-512" is the title of the profile which is referred to when installing the scheduler
- "rate-limit 512" allows no more than 512 kbit/sec to pass which avoids queueing in the modem.
- "overhead 30" specifies how many framing bytes are added by the modem to "pack" the IP packet on the link. The framing is taken into account by the rate limiter.
- "atm" tells the rate limiter that the access link is ATM based. This option includes the ATM overhead into the rate limit calculation. Please add 8 bytes to the overhead for AAL5 in this case.
- "source traffic-class" enters a sub-mode where the specific handling for a traffic-class is described. The list of sources in the service-policy profile tells the arbiter which "traffic sources" to serve.
- "local-voice" is the predefined traffic-class for locally terminated voice packet streams.
- "priority" means that packet of the source being described are always passed on immediately, packets of other classes follow later if the rate limit permits.

Configure DownStreamQoS™

The Patton Device can throttle downstream data traffic on the access link, minimizing the risk of lost or delayed voice or other real-time packets coming from the Internet or WAN. The example below defines a service-policy profile applied to incoming traffic on the WAN interface:

Mode: profile service-policy

Step	Command	Purpose
1	**device(pf-svcpol)[SVCPOL_0]#rate-limit <kilobytes> [voice-margin <kilobytes>]**	Specifies the margin by which the rate-limit is lowered for DownstreamQoS.
2	**device(pf-svcpol)[$(profile-Name)]#source traffic-class <traffic-class>**	Enter traffic class.

Mode: profile service-policy

Step	Command	Purpose
1	**device(src-tc)[TC_0]#real-time**	Configures the traffic as real-time, optimizing it for minimum-delay.

The command parameters in the example above function as described below:

voice-margin:
For DownStreamQoS (only for incoming traffic) Trinity reduces the maximum bandwidth by the specified bit rate in kbit/sec (200 kbit/sec in this example). The throttling effectively reduces the average rate of incoming data packets, which frees bandwidth for priority traffic thus improving voice quality. For best results, use a generous value for voice-margin to accommodate fluctuations in traffic flow. 33% of the total bandwidth is recommended, up to a maximum of 200 kbit/sec (this default behavior can be easily configured with the voice-margin auto option).

real-time:
If at least one of the listed traffic classes contains the "real-time" command the service policer filters-out bursts in the data traffic to keep the delay of voice streams small and without fluctuations.

Service-Policy configuration task list

To configure service-policy profiles, perform the tasks in the following sections.

- Creating a service-policy profile
- Configure link arbiter
- Set QoS-related IP header field
- Binding a classifier profile to the outbound traffic of an IP interface

Creating a service-policy profile

The service-policy profile defines whether and how to set IP header fields such as TOS, precedence, DSCP, and ECN based on the traffic-class tag of an outgoing packet.

This procedure describes how to create a service-policy profile and enter the profile's configuration mode.

Mode: administrator exec

Step	Command	Purpose
1	**node(cfg)#profile service-policy** *<profile-name>*	Creates the service-policy profile <*profile-name*> and enters its configuration mode.

The parameter <*profile-name*> is the identifier by which the profile will be known. Entering this command puts you into service-policy profile configuration mode where you can enter traffic-class-specific modes as shown in the next section.

Use the no form of this command to delete a service-policy profile. You cannot delete a service policy profile if it is currently bound to an interface. When you modify a service-policy profile, the new settings immediately become active.

Example: Create a service-policy profile

In the following example the service-policy profile named SP_WAN is created.

```
node>enable
node#configure
node(cfg)#profile service-policy SP_WAN
node(pf-svcpol)[SP_WAN]#
```

Configure Link arbiter

For the link arbitration the profile service-policy holds all the necessary parameters.

Rate limit

For QoS it is important to be at the bottleneck and control the queue of the bottleneck in order to control which streams get which share and which packets needs to be dropped in the case of an overload of the link. Therefore the rate-limit needs to be configured to a value slightly smaller than the actual available link.

Mode: Profile service-policy

Step	Command	Purpose
1	**node(svcpol-cls)**[<profile-name>]**#[no] ratelimit**<kbits> **[overhead <Bytes>] [ethernet | atm][voice-margin**<kbits>**|auto]**	Limits global interface rate to value in kbits. The configured value should be slightly smaller than the real available uplink. Overhead specifies if there are additional header bytes added from the modem, which are more than the normal Ethernet header. With link-layer atm the padding for atm cells can be taken into account. Voicemargin is substracted from total limit for DownstreamQoS (virtual bottleneck). Default: disabled. Default link-layer: Ethernet. Default overhead: 0 Bytes. Default voice-margin: 0.

Arbiter mode

The arbiter mode defines if the distribution of bandwidth to traffic-classes or hierarchical policies happens with limiting to maximum values (shaper) or guaranteeing minimum values (wfq). For achieving a mixture of both arbitration modes one needs to define a hierarchical service-policy profile and include that as source in another service policy profile.

Mode: Profile service-policy

Step	Command	Purpose
1	**node(svcpol-cls)[**<profile-name>**]#mode (shaper | wfq)**	Set the arbiter mode to wait-fair-queuing (wfq) or shaper. Default: wfq

Source traffic-class

The main sources of packets for the arbiter are packets belonging to traffic-classes. The packets get the traffic-class form the classifier. In the arbiter the share or bandwidth is assigned to traffic classes.

Packets belonging to a traffic-class not explicitly handled in the profile service-policy get only scheduled if there is enough left bandwidth after handling all the other traffic-classes.

Mode: Profile service-policy

Step	Command	Purpose
1	**node(svcpol-cls)[**<profile-name>**]#[no] source traffic-class** <traffic-class>	Enters the configuration mode for a traffic-class.

Hierarchical profile service-policy

It is possible to have hierarchical service policy profiles as a source of other service policy profiles. This can be used for achieving a mixture of the arbiter modes shaper and wfq. The other use-case is setting up the borrowing between traffic-classes in the wfq mode. Borrowing means that bandwidth of a traffic-class which is not used is distributed even for all the sources in the same hierarchical profile. An only after that it is given to other traffic-classes in other profiles.

Mode: Profile service-policy

Step	Command	Purpose
1	**node(svcpol-cls)[**<profile-name>**]#[no] sourcepolicy** <service-policy profile>	Enters the configuration mode of a hierarchical lower level service-policy profile.

Share for weight –fair-queuing

The command share is used with wfq link arbitration to assign the weight to the selected traffic-class. When defining a number of source classes, the values are relative to each other. It is recommended to split 100—which can be read as 100%—among all available source classes, e.g. with 20, 30 and 50 as value for the respective share commands, which represent 20%, 30% and 50%.

This command is only used when the arbiter is in wfq mode. For the shaper mode it is ignored. Traffic-classes without share get no guaranteed bandwidth and therefore are sent only if there is some bandwidth left from the others.

Mode: Source traffic-class and policy

Step	Command	Purpose
1	**node(src)[**name**]#[no] share** <percentage>	Defines fair queuing weight for minimal guaranteed bandwidth (relative to other sources) to percentage for the selected class or policy name. Default: disabled

Bit-rate for shaper

The command rate is used with shaper link arbitration to assign the (average) bit-rate to the selected source. When enough bandwidth is available each source will exactly receive this bandwidth (but no more), when overloaded the shaper will behave like a wfq arbiter. Bit-rate specification for shaper (kilobits).achieving a mixture of both arbitration modes one needs to define a hierarchical service policy profile and include that as source in another service policy profile.

This command is only used when the arbiter is in shaper mode. For the wfq mode it is ignored. Traffic-classes without rate are degraded to a lower priority and get served only if all other traffic classes are served before.

Mode: Source traffic-class and policy

Step	Command	Purpose
1	**node(src)[**name**]#[no] rate** <kbits>	Defines the (average) bit-rate to the selected in kbps kilobits or as remaining if a second priority source is getting the unused bandwidth for the selected class or policy name

Assigning absolute priority

This command priority can only be applied to classes, but not to lower level polices. The class is given absolute priority effectively bypassing the link arbiter. Care should be taken, as traffic of this class may block all other traffic. The packets given “priority” are taken into account by the “rate-limit”. Use the command police to control the amount of “priority” traffic.

Mode: Source traffic-class

Step	Command	Purpose
1	**node(src)[**name**]#[no] priority**	Defines absolute priority effectively bypassing the link arbiter for the selected class or policy name

Real time traffic

The command real-time specifies that the traffic in this class requires a completely fixed rate with minimum delay for user interactivity (i.e. voice). This command minimizes packet-bursts processing so no real-time packets are being delayed after them, keeping the rate as fix as possible.

Mode: Source traffic-class

Step	Command	Purpose
1	**node(src)[**name**]#[no] real-time**	Configures the traffic as real-time, optimizing it for minimum-delay

Queue length

The command queue-limit specifies the maximum number of packets queued for the class name. Excess packets are dropped. Used in “class” mode—queuing only happens at the leaf of the arbitration hierarchy tree. The no form of this command reverts the queue-limit to the internal default value, which depends on your configuration.

Mode: Source traffic-class

Step	Command	Purpose
1	**node(src)[**name**]#[no] queue-limit** <packets>	Defines the maximum number of packets queued for the selected class

Queue type

The type of queue defines the drop-behavior and/or has an influence to the fairness between streams of the same traffic-class.

Mode: Source traffic-class

Step	Command	Purpose
1	**node(src)[**name**]#queue-mode fifo**	The packets first come are first served. When the queue is full, excess packets are dropped (taildrop). This is the default setting.
1	**node(src)[**name**]#queue-mode sfq [flows** <number of flows> **]**	The streams are divided into sub-queues and these are served all with equal weight. Flows control the number of sub-queues. Default flows: 1024.
1	**node(src)[**name**]#queue-mode red [burst-tolerance** <1-10> **]**	Defines random early detection (RED) for queues of for the selected traffic-class or policy name. The range for the optional value burst-tolerance is from 1 to 10. Default burst-tolerance: 5.

Discarding excess load

The command police controls traffic arriving in a queue for class name. The value of the first argument average -kilobits defines the average permitted rate in kbps, the value of the second argument kilobits-ahead defines the tolerated burst size in kbps ahead of schedule. Excess packets are dropped.

Mode: Source traffic-class

Step	Command	Purpose
1	**node(src)[**name**]#[no] police** <aver-agerate> **burst-size** <kbits-ahead>	Defines how traffic arriving in a queue for the selected class or policy name has to be controlled. The value average-kilobits for average rate permitted is in the range from 0 to 10000 kbps. The value kilobits-ahead for burst size tolerated ahead of schedule is in the range from 0 to 10000.

Set QoS-related IP header field

When a service-policy profile is bound to an IP interface, the profile may set IP header fields such as TOS, precedence, DSCP, and ECN of all packets sent over that IP interface. For each internal traffic class, a separate set of header field values can be configured. This procedure describes how to map an internal traffic class to a set of IP header fields.

Mode: Profile service-policy

Step	Command	Purpose
1	**node(svcpol-cls)[**<profile-name>**]#source traffic-class** traffic-class	Enters a sub-mode that combines configuration commands for packets tagged with the specified traffic-class.
2a	**node(src-tc)[**traffic-class**]#set tos** tos	Sets the IP type-of-service header field to tos.
3a	**node(src-tc)[**traffic-class**]#set precedence** precedence	Sets the IP precedence header field to precedence.
2b	**node(src-tc)[**traffic-class**]#set dscp** dscp	Sets the IP differentiated-services-code-point header field to dscp.
3b	**node(src-tc)[**traffic-class**]#set ecn** ecn	Sets the IP explicit-congestion-notification field to ecn.

Note The TOS/precedence header fields use the same bits as the DSCP/ECN fields. Thus you can either configure TOS and/or precedence values or DSCP and/or ECN values. If you for example try to enter a DSCP value after having configured a TOS value, the DSCP value is not accepted and you will be presented with an error message.

Example: Set IP TOS and precedence fields

The following example prepares a service-policy profile to set the IP TOS field to 5 for all packets tagged with the MGMT traffic class. Note that the profile must be bound to an IP interface in order to get activated (see section below).

```
node(cfg)#profile service-policy SP_WAN
node(pf-svcpol)[SP_WAN]#source traffic-class MGMT
node(src-tc)[MGMT]#set tos 5
```

Binding a classifier profile to the outbound traffic of an IP interface

To apply a service-policy profile to the outbound traffic it has to be bound to an IP interface. This procedure describes how to do so with the use command.

Mode: IP interface

Step	Command	Purpose
1	**node(if-ip)[**ifname**]#use profile service-policy [in|out]** <profile-name>	Binds the service-policy profile <profile-name> to outgoing packets on the IP interface ifname.

The command offers the following options:

Parameter	Explanation
ifname	Name of the IP interface to which a service-policy profile gets bound.
<profile-name>	The name of a service-policy profile that has already been created using the profile service-policy command. This argument must be omitted in the no form.
in	Specifies that the service-policy profile applies to packets received over this interface. This currently is of limited use as packet tagging should always happen at an egress point of the device to match the QoS-field semantics of the target network.
out	Specifies that the service-policy profile applies to packets sent over this interface.

The no form of the use commandis used to unbind a service-policy profile from an IP interface. When using this form, you don't have to specify the profile name.

Mode: IP Interface

Step	Command	Purpose
1	**node(if-ip)[**ifname**]#no use profile service-policy [in|out]**	Unbinds the service-policy profile for incoming or incoming or outgoing packets from the IP interface ifname.

Parameter	Explanation
ifname	Name of the IP interface on which a service-policy profile gets unbound.
in	Specifies that the service-policy profile applies to packets received over this interface.
out	Specifies that the service-policy profile applies to packets sent over this interface.

Example: Bind and unbind a service-policy profile.

Bind the service-policy profile created in the previous example to tag outgoing packets on interface WAN in the default IP context (ROUTER).

```
node(cfg)#context ip ROUTER
node(ctx-ip)[ROUTER]#interface WAN
node(ip-if)[ROUTER.WAN]#use profile service-policy out SP_WAN
Unbind the SP_WAN service-policy profile from outgoing traffic on interface WAN:
node(ip-if)[ROUTER.WAN]#no use profile service-policy out
```

Troubleshooting

In the case of an Issue with the service-policy profile the first thing to check is if the profile could be applied to the system successfully. Execute on a newly booted device:

- Show log boot
- Show log error

If there are errors logged for service-policy then you should open an incident at your patton support representative with the log files and your startup-configuration. For the meantime a nearing approach can be tried:

- Make sure the underlying transport (in most cases Ethernet) is up and running.
- Make sure the underlying transport (in most cases Ethernet) is bound to the ip interface.
- Unbind the failing profile service-policy from the ip interface.
- Create a new and empty profile service-policy.
- Bind the new profile service-policy to the ip interface.
- Enter the new profile service-policy and execute one command at once until it is rejected with an error or the final desired configuration is reached.
- When there is a command rejected due to failure, you may try to change parameters, use alternative commands or omit the command for going forward.

Chapter 33 Access Control List Configuration

Chapter contents

Introduction

This chapter provides an overview of IP Access Control Lists (ACLs) and describes the tasks involved in configuring them.

This chapter includes the following sections:

- About ACLs
- ACL configuration task list (see page 334)
- Examples (see page 339)

About Access Control Lists (ACLs)

This section briefly describes what access lists do, why and when you should configure access lists, and basic versus advanced access lists.

What Access Lists Do

ACLs implement a firewall by filtering network traffic, forwarding or dropping each packet based on the ACL rules and bindings. ACL rules may match packets by any of the criteria listed in chapter (see Chapter 36, "Packet Matching" on page 353). ACL bindings determine which ingress and/or egress interface will be matched.

The firewall is a stateful firewall, meaning that packets may be matched not only based on information that can be determined from that packet alone, e.g. ingress/egress interface or header information such as source/destination IP addresses, but also based on the connection state which is determined by examining previous packets. This is useful because many networking application protocols have an initial connection that can be matched with well-known criteria, e.g. the server's port number, but then use the initial connection to negotiate a second, related connection using dynamically assigned port numbers. For these applications to work through a stateless firewall, the entire port range possible for that related connection must be permitted, usually all ports 1024 and greater, which greatly decreases the security of the firewall. A stateful firewall, on the other hand, understands how the related connection is negotiated by the initial connection and dynamically permits that related connection once it is negotiated.

The router tracks the connection state for the following protocols:

- ICMP
- TCP
- UDP

and the following applications:

- SIP
- FTP
- TFTP
- PPTP
- SANE
- IRQ

Note Sophisticated users can sometimes successfully evade or fool basic access lists because no authentication is required.

Why You Should Configure Access Lists

You should use ACLs to provide a basic level of security for accessing your network. If you do not configure ACLs on your router, all packets passing through the router could be allowed onto all parts of your network.

For example, ACLs can allow one host to access a part of your network, and prevent another host from accessing the same area. In figure 49, host A is allowed to access the Human Resources network and host B is prevented from accessing the Human Resources network.

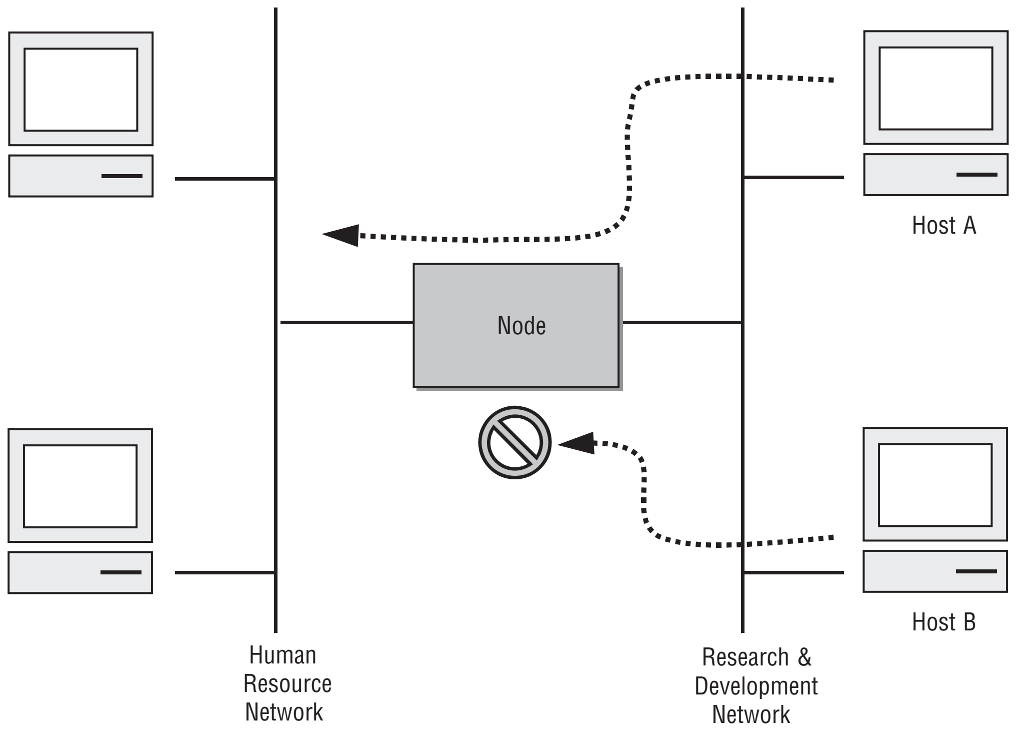


Figure 49. Using traffic filters to prevent traffic from being routed to a network

You can also use access lists to decide which types of traffic are forwarded or blocked at the device interfaces. For example, you can permit e-mail traffic to be forwarded but at the same time block all Telnet traffic.

When to Configure Access Lists

ACLs should be used in firewall routers, which are often positioned between your internal network and an external network such as the Internet. You can also use ACLs on a router positioned between two parts of your network, to control traffic entering or exiting a specific part of your internal network.

To provide the security benefits of ACLs, you should configure ACLs at least on border routers, i.e. those routers situated at the edges of your networks. This provides a basic buffer from the outside network or from a less controlled area of your own network into a more sensitive area of your network.

On these routers, you should configure ACLs for each network protocol configured on the router interfaces. You can configure ACLs so that inbound traffic or outbound traffic or both are filtered on an interface.

Features of Access Control Lists

The following features apply to all IP ACLs:

- An ACL may contain multiple rules. The order of rules is significant. Each rule is processed in the order it appears in the configuration file. As soon as a rule matches, the corresponding action is taken and no further processing takes place.
- An IP interface may be bound to multiple ACLs. The order in which the ACLs are bound is significant. Each ACL is scanned in the order in which it was bound. If no matching rule in found in the first ACL, the second ACL is scanned, and so on. If no matching rule is found in any of the ACLs, the packet is dropped. That is, there is an implicit **deny** all rule at the end of the last ACL.

Note Two or more administrators should not simultaneously edit the configuration file. This is especially the case with access lists. Doing this can have unpredictable results.

Once in access control list configuration mode, each command creates a statement in the access control list. When the access control list is applied, the action performed by each statement is one of the following:

- **permit** statement causes any packet matching the criteria to be accepted.
- **deny** statement causes any packet matching the criteria to be dropped.

Access Control List Configuration Task List

To configure an IP access control list, perform the tasks in the following sections.

- Mapping out the goals of the ACL
- Creating an ACL profile and enter configuration mode (see page 335)
- Adding a filter rule to the current ACL profile (see page 335)
- Binding and unbinding an ACL profile to an IP interface (see page 336)
- Displaying an ACL profile (see page 338)

Mapping Out the Goals of the Access Control List

To create an ACL, you must:

- Assign a unique name to the access list
- Define packet-filtering criteria

A single access control list can have multiple rules.

Before you begin to enter the commands that create and configure the ACL, be sure that you are clear about what you want to achieve with the firewall. Consider whether it is better to deny specific accesses and permit all others or to permit specific accesses and deny all others.

Note A single ACL can have multiple rules, but editing those rules online can be tedious. Therefore, we recommend editing complex ACLs offline within a configuration file and downloading the configuration file later via TFTP to your device.

Creating an Access Control List Profile and Enter Configuration Mode

This procedure describes how to create an ACL and enter ACL configuration mode.

Mode: Administrator execution

Step	Command	Purpose
1	***device*(cfg)#profile acl** *name*	Creates the ACL profile *name* and enters the configuration mode for this ACL.

The ACL profile will be known by *name*. Entering this command puts you into *ACL configuration mode* where you can enter the individual statements that will make up the ACL rules.

Use the **no** form of this command to delete an ACL profile. You cannot delete an ACL profile if it is currently bound to an interface.

Example: Create an ACL profile

In the following example, the ACL profile named *WAN_RX* is created and the shell of the ACL configuration mode is activated.

```
device>enable
device#configure
device(cfg)#profile acl WAN_RX
device(pf-acl)[WAN_RX]#
```

Adding and Deleting a Filter Rule to the Current Access Control List Profile

The commands **permit** or **deny** are used to define an ACL rule. This procedure describes how to create an ACL rule.

Mode: ACL configuration

Step	Command	Purpose
1	***device*(pf-acl)device#{permit | deny}** *match*	Creates an ACL rule that either permits or denies access according to the *match*.

The table below explains the syntax:

Keyword	Meaning
permit	Forward any packet matching the *match* criteria.
deny	Drop any packet matching the *match* criteria.
match	See Chapter 36, "Packet Matching" on page 353.

If no *match* is specified, the rule will match all packets, so if you place the rule deny at the top of an access control list profile, no packets will pass regardless of the other rules you defined.

Example: Create ACL rules

Select the ACL profile named WAN_RX and create some rules to:

- drop all ICMP echo requests (as used by the ping command),
- but forward all other ICMP traffic,
- forward any TCP traffic to host 193.14.2.10 via port 80,
- forward UDP traffic from host 62.1.2.3 to host 193.14.2.11 via any port in the range from 1024 to 2048
- forward all traffic from the 97.123.111.0/24 subnet to host 193.14.2.11.

```
device(cfg)#profile acl WAN_RX
device(pf-acl)[WAN_RX]#deny protocol icmp icmp-type echo-request
device(pf-acl)[WAN_RX]#permit protocol icmp
device(pf-acl)[WAN_RX]#permit protocol tcp dest-ip 193.14.2.10 dest-port 80
device(pf-acl)[WAN_RX]#permit protocol udp src-ip 62.1.2.3 dest-ip 193.14.2.11
dest-port 1024..2048
device(pf-acl)[WAN_RX]#permit src-ip 97.123.111.0/24 dest-ip 193.14.2.11
device(pf-acl)[WAN_RX]#exit
device(cfg)#
```

The no form of the permit or deny command is used to delete an ACL rule. This procedure describes how to delete an ACL rule.

Mode: ACL Configuration

Step	Command	Purpose
1	***device*(pf-acl)device#show running-config current-mode**	Show the ACL rules in the ACL *name*. Use this command to get the index of the rule you want to delete. You will use it in step 2.
2	***device*(pf-acl)device#no permit** *index*	Removes the ACL rule at the specified *index*.

Example: Delete an ACL rule

Select the ACL profile named WAN_RX and delete the rule to permit any TCP traffic to host 193.14.2.10 via port 80.

```
device(cfg)#profile acl WAN_RX
device(pf-acl)[WAN_RX]#show running-config current-mode
deny 1 protocol icmp icmp-type echo-request
permit 2 protocol icmp
permit 3 protocol tcp dest-ip 193.14.2.10 dest-port 80
permit 4 protocol udp src-ip 62.1.2.3 dest-ip 193.14.2.11 dest-port 1024..2048
permit 5 src-ip 97.123.111.0/24 dest-ip 193.14.2.11
device(pf-acl)[WAN_RX]#no permit 3
device(pf-acl)[WAN_RX]#exit
device(cfg)#
```

Binding and Unbinding an Access Control List Profile to an IP Interface

The command **use** is used to bind an ACL profile to an IP interface. This procedure describes how to bind an ACL profile to incoming packets on an IP interface.

Mode: Interface

Step	Command	Purpose
1	***device*(if-ip)[*if-name*]#use profile acl in** *name* **[to { local | interface** *dest-if-name*} **]**	Binds ACL profile *name* to incoming packets on IP interface *if-name*. If **local** is specified, the ACL only applies to packets destined to the router itself. If *dest-if-name* is specified, the ACL only applies to packets destined to be routed out of *dest-if-name*. Otherwise, the ACL applies to all incoming packets on IP interface *if-name* regardless of the destination.

This procedure describes how to bind an ACL profile to all outgoing packets on an IP interface.

Mode: Interface

Step	Command	Purpose
1	***device*(if-ip)[*if-name*]#use profile acl out** *name*	Binds ACL profile *name* to all outgoing packets on IP interface *if-name*.

The table below explains the syntax:

Keyword	Meaning
if-name	The name of the IP interface to which an ACL profile is bound.
name	The name of an ACL profile that has already been created using the profile acl command.
local	*(Optional)* If specified, the binding only applies to packets destined to the router itself.
dest-if-name	*(Optional)* If specified, the binding only applies to packets destined to be routed out of the IP interface by this name.

The no form of the use command is used to unbind an ACL profile from an IP interface. This procedure describes how to unbind an ACL profile from an IP interface.

Mode: Interface

Step	Command	Purpose
1	***device*(if-ip)[*if-name*]#show running-config current-mode**	Shows the index to use to unbind the desired ACL.
2	***device*(if-ip)[*if-name*]#no use profile acl {in | out}** *index*	Unbinds ACL profile at *index* for either incoming or outgoing packets on IP interface *if-name*.

The table below explains the syntax:

Keyword	Meaning
if-name	The name of the IP interface from which the ACL profile is unbound.

Keyword	Meaning
in	Unbind the profile from incoming packets.
out	Unbind the profile from outgoing packets.
index	The index of the access control list which is unbound.

Example: Bind and unbind an ACL profile to/from an IP interface

Bind ACL profile WAN_RX to incoming packets on the interface *WAN* in the IP router context.

```
device(cfg)#context ip ROUTER
device(cfg-ip)[ROUTER]#interface WAN
device(cfg-if)[WAN]#use profile acl WAN_RX in
```

Bind ACL profile WAN_RX from incoming packets on the interface *WAN* in the IP router context.

```
device(cfg)#context ip ROUTER
device(cfg-ip)[ROUTER]#interface WAN
device(cfg-if)[WAN]#no use profile acl in 1
```

Displaying an Access Control List Profile

The **show profile switch acl** command displays the indicated ACL profile. If no specific profile is selected all created ACL profiles are shown. If an ACL is linked to an IP interface the number of matches for each rule is displayed.

This procedure describes how to display a certain ACL profile.

Mode: Administrator execution or any other mode, except the operator execution mode

Step	Command	Purpose
1	***device*#show profile acl** *name*	Displays the ACL profile *name.*

Example: Displaying an ACL entry

The following example shows how to display the ACL profile named WAN_RX.

```
device#show profile acl WAN_RX
```

Examples

Denying a Specific Subnet

Figure 50 shows an example in which a server attached to network 172.16.1.0 shall not be accessible from outside networks connected to IP interface *wan*. To prevent access, an incoming filter rule named *JAMMING* is defined, which blocks any IP traffic from network 172.16.2.0 and has to be bound to IP interface *wan*.

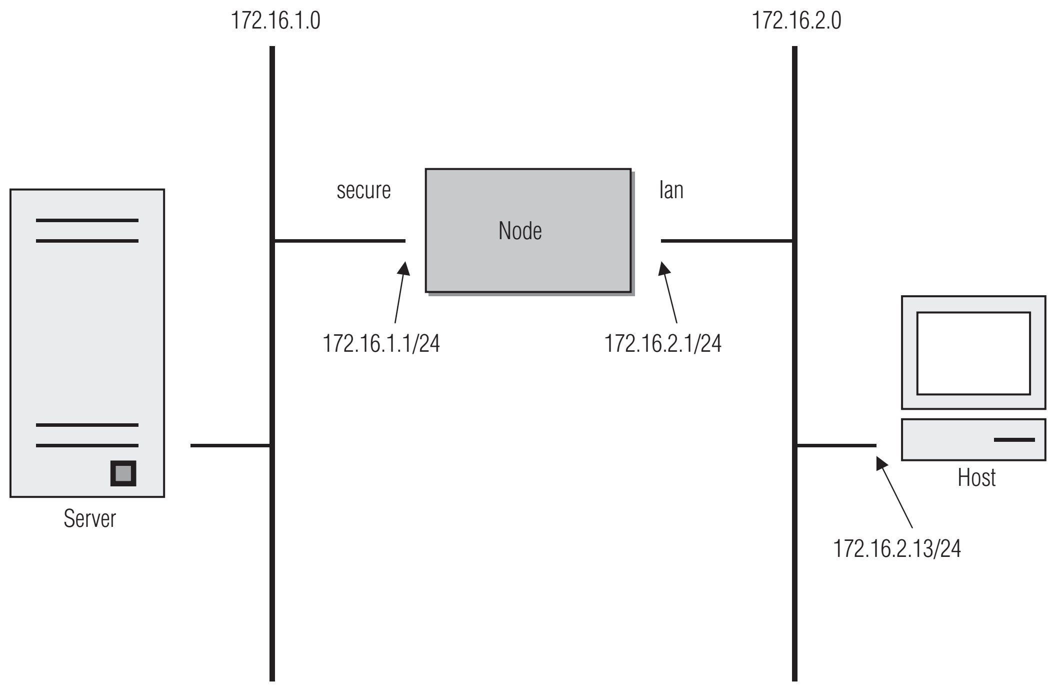


Figure 50. Deny a specific subnet on an interface

The commands that have to be entered are listed below.

```
172.16.2.1>enable
172.16.2.1#configure
172.16.2.1(cfg)#profile acl JAMMING
172.16.2.1(pf-acl)[JAMMING]#deny ip src-ip 172.16.2.0/24 dest-ip 172.16.1.0/24
172.16.2.1(pf-acl)[JAMMING]#permit
172.16.2.1(pf-acl)[JAMMING]#exit
172.16.2.1(cfg)#context ip ROUTER
172.16.2.1(cfg-ip)[ROUTER]#interface wan
172.16.2.1(if-ip)[lan]#use profile acl in JAMMING
172.16.2.1(if-ip)[lan]#exit
172.16.2.1(cfg-ip)#copy running-config startup-config
```

Denying Traffic Between Two Interfaces

Figure 51 shows an example in which there are two LANs, neither of which should be accessible from the other, but both of which should have access to the WAN.

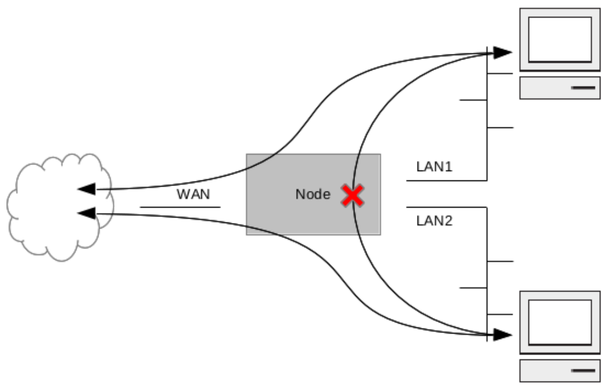


Figure 51. Deny traffic between two interfaces

The commands that have to be entered are listed below.

```
device>enable
device#configure
device(cfg)#profile acl DENY
device(pf-acl)[DENY]#deny
device(pf-acl)[DENY]#exit
device(cfg)#context ip ROUTER
device(cfg-ip)[ROUTER]#interface LAN1
device(cfg-ip)[LAN1]#use profile acl in DENY to interface LAN2
device(cfg-ip)[LAN1]#exit
device(cfg-ip)[ROUTER]#interface LAN2
device(cfg-ip)[LAN2]#use profile acl in DENY to interface LAN1
device(cfg-ip)[LAN2]#exit
device(cfg-ip)[ROUTER]#copy running-config startup-config
```

Permit Only Traffic Generated From LAN

In this example, clients on the LAN are able to connect to FTP (port 21), SSH (port 22), and HTTP (port 80) servers on the WAN as well as to ping them, but any traffic received from the WAN that was not initiated by a client on the LAN is denied.

```
device>enable
device#configure
device(cfg)#profile acl PERMITTED_APPLICATIONS
device(pf-acl)[PERMITTED_APPLICATIONS]#permit protocol tcp dest-port 20,22,80
device(pf-acl)[PERMITTED_APPLICATIONS]#permit protocol icmp
device(pf-acl)[PERMITTED_APPLICATIONS]#exit
device(cfg)#profile acl STATEFUL_FIREWALL
device(pf-acl)[STATEFUL_FIREWALL]#permit connection-state established related
device(pf-acl)[STATEFUL_FIREWALL]#exit
device(cfg)#context ip ROUTER
device(cfg-ip)[ROUTER]#interface WAN
device(if-ip)[WAN]#use profile acl in STATEFUL_FIREWALL
device(if-ip)[WAN]#use profile acl out STATEFUL_FIREWALL
device(if-ip)[WAN]#use profile acl out PERMITTED_APPLICATIONS
```

Chapter 34 Classifier Configuration

Chapter contents

Introduction

This chapter provides an overview of Trinity's packet classifier and describes the tasks involved in its configuration.

About the Classifier

This section briefly describes what the classifier does, as well as why and when to configure the packet classification.

What the Classifier Does

The classifier is part of the overall Quality-of-Service architecture of Trinity. As depicted in figure 47 on page 316 the classifier groups packet flows into virtual traffic-classes. Other network components in Trinity are able to make routing and quality-of-service decisions based on this traffic-class. For example, the traffic-class may be used for the following:

- To perform **policy routing** by consulting special routing tables for certain traffic classes
- To restrict access to the device by rejecting all packets belonging to a traffic-class in an access control list (**ACL**)
- To tag packet headers with class-of-service information when sending the packet to a remote host (see **service-policy profile**)

That is, the classifier makes Trinity devices aware of different services whereas the ACL, policy routing and the service-policy profile decide how to actually treat them.

How the Classifier Works

The classifier consists of an ordered set of rules. Each rule sets the traffic-class of certain packets dependent on their header fields. A rule consists of the following two parts:

(a) A condition part, which inspects each packet based on over 20 criteria, such as the source/destination address and port, the packet length, the IP TOS field, etc. (see #<Packet Matcher#> for the criteria available for matching packets).

(b) An action part, which sets a traffic-class tag to all packets that are matching the condition **(a)**.

The order of the rules in a classifier profile is significant. Each entry is processed in the order it appears in the configuration file. As soon as the condition part **(a)** matches, the traffic class is set **(b)** and no further rules are processed.

Classifier rules are grouped together in profiles. This allows you to bind the same profile (the same set of rules) to several IP interfaces, for example. The order in which the profiles are bound is significant. Each profile is processed in the order in which it is bound in the configuration file. If an entry in one of the profiles matches, the remaining bound profiles are not processed any further.

Figure 52 shows that there are three locations in the routing path where the classifier may examine packets and assign them an internal traffic-class. In each of those locations, a list of classifier profiles can be bound. Quite early after receiving a packet over a port or circuit, the input classifier of the bound IP interface has a chance to classify the packet. In addition, after routing a packet to an egress IP interface, the output classifier of that interface may modify the traffic-class of the packet. Finally, each locally-generated packet may be tagged on the local pseudo-interface before it is routed to an egress interface. Figure 25 on page 232 shows the exact order in which each packet traverses the classifier and other packet-processing services.

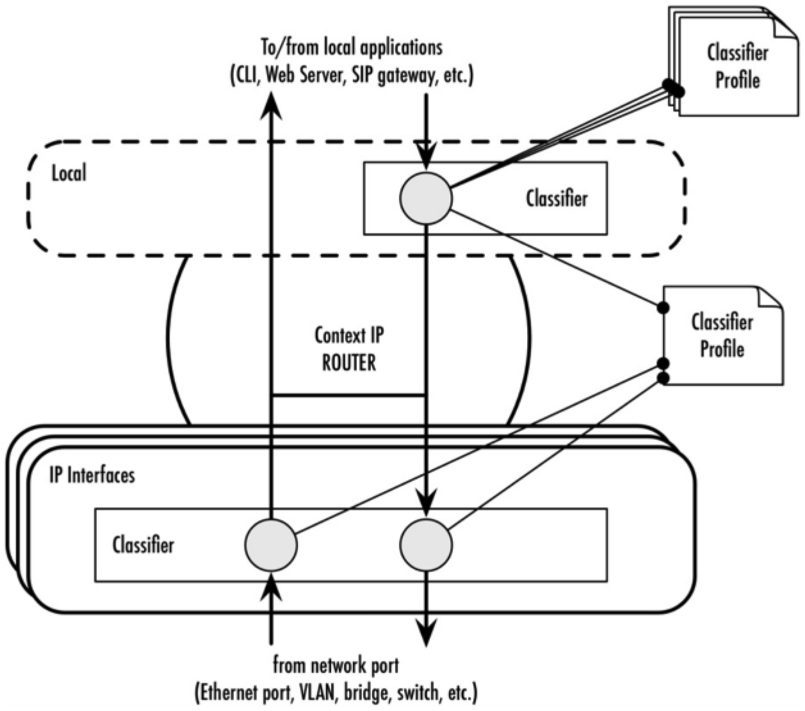


Figure 52. Locations in the routing path where packets may be classified

Classifier Configuration Task List

To configure the classifier, perform the tasks in the following sections.

Mapping the Goals of the Classifier

To create a classifier profile you must:

- Assign a unique name to the classifier profile
- Define packet-filtering criteria
- Define the traffic-class to be set

Before you begin to enter the commands that tag packets with a traffic-class, be sure that you have planned how many traffic-classes are needed, what they mean and what classifier rules are needed to achieve your goals.

Note The access control list (ACL) may rely on traffic-class tags to accept or reject packets. Thus editing a (bound) classifier profile is dangerous as you could potentially lock yourself out forever. Therefore, we recommend editing complex QoS scenarios offline within a configuration file and downloading the configuration file later via TFTP to our Trinity device.

Creating a Classifier Profile and Enter Configuration Mode

This procedure describes how to create a classifier profile and enter the classifier profile configuration mode.

Mode: Administrator exec

Step	Command	Purpose
1	***device*(cfg)#profile classifier** *<profile-name>*	Creates the classifier profile *<profile-name>* and enters its configuration mode.

The parameter *<profile-name>* is the identifier by which the profile will be known. Submitting this command enters into the classifier profile configuration mode, where you can enter the individual classifier rules to set the traffic-class of packets based on filtering conditions.

Use the **no** form of this command to delete a classifier profile. You cannot delete a classifier profile if it is currently bound to an interface or to the *local* pseudo-interface. When you modify a classifier profile, the new settings immediately become active.

Example: Create a classifier profile

In the following example, the classifier profile name *WAN_RX* is created.

```
device>enable
device#configure
device(cfg)#profile classifier WAN_RX
device (pf-cls)[WAN_RX]#
```

Adding a Rule to the Current Classifier Profile

The **match** command is used to define a classifier rule, which matches for some criteria in the packet and sets a traffic-class tag. This procedure describes how to create a classifier rule.

Mode: Profile classifier

Step	Command	Purpose
1	***device*(pf-cls)**[*<profile-name>*]**#match set traffic-class** *traffic-class*	Creates a classifier rule that sets the traffic-class tag to *traffic-class* if the packet matches all criteria specified in *match*. See #<Packet Matcer># for the syntax of the *match* specifier.

Example: Set the traffic class to *MGMT* for packets (UDP or TCP) sent to destination port 23 (Telnet) or 80 (HTTP).

```
device(cfg)#profile classifier WAN_RX
device(pf-cls)[WAN_RX]#match protocol udp dest-port 23,80 set traffic-class MGMT
device(pf-cls)[WAN_RX]#match protocol tcp dest-port 23,80 set traffic-class MGMT
```

Binding and Unbinding a Classifier Profile To/From an IP Interface to Tag Incoming/Outgoing Packets

To filter incoming or outgoing packets, one of more classifier profiles can be bound to an IP interface. This allows configuring a different classifier for ingress/egress traffic over each IP interface.

The **use** command is used to bind a classifier profile to an IP interface. This procedure describes how to bind a classifier profile to filter incoming/outgoing packets on an IP interface.

Mode: IP interface

Step	Command	Purpose
1	***device*(if-ip)**[*ifname*]**#use profile classifier** [in | out] <*profile-name*>	Binds the classifier profile <*profile-name*> to incoming or outgoing packets on the IP interface *ifname*.

The command offers the following options:

Parameter	Explanation
ifname	Name of the IP interface to which a classifier profile gets bound.
<*profile-name*>	The name of a classifier profile that has already been created using the profile classifier command. This argument must be omitted in the no form.
in	Specifies that the classifier applies to packets received over this interface.
out	Specifies that the classifier applies to packets sent over this interface.

The **no** form of the **use** command is used to unbind a classifier profile from an IP interface. When using this form, instead of specifying the name of the profile to unbind, you have to give the numeric index of the respective **use** command as it can be found in the current running-config.

Mode: IP interface

Step	Command	Purpose
1	***device*(if-ip)**[*ifname*]**#no use profile classifier** [in | out]*index*	Unbinds the classifier profile at *index* for incoming or outgoing packets from the IP interface *ifname*.

Parameter	Explanation
ifname	Name of the IP interface to which a classifier profile gets bound.
index	Numeric index of the bind command (type show running-config current-mode to retrieve the index).
in	Specifies that the classifier applies to packets received over this interface.
out	Specifies that the classifier applies to packets sent over this interface.

Example: Bind and unbind several classifier profiles

Bind two classifier profiles in the specified order to incoming packets on interface *WAN* in the default IP context (*ROUTER*)

```
device(cfg)#context ip ROUTER
device(ctx-ip)[ROUTER]#interface WAN
device(ip-if)[ROUTER.WAN]#use profile classifier in WAN_RX
device(ip-if)[ROUTER.WAN]#use profile classifier in GENERIC
```

Show the current configuration of the IP interface *WAN*:

```
device(ip-if)[ROUTER.WAN]#show running-config current-mode
```

```
ipaddress WAN 20.1.1.1/24
ipaddress DHCP
use profile classifier in 1 WAN_RX
use profile classifier in 2 GENERIC
```

Unbind the *WAN_RX* classifier profile from incoming traffic on interface *WAN*:

```
device(ip-if)[ROUTER.WAN]#no use profile classifier in 1
```

Verify that we have unbound the right profile:

```
device(ip-if)[ROUTER.WAN]#show running-config current-mode
ipaddress WAN 20.1.1.1/24
ipaddress DHCP
use profile classifier in 1 GENERIC
```

Binding and Unbinding a Classifier Profile to Tag Locally-generated Packets

Instead of binding a classifier profile to an IP interface, you are able to bind one or more classifier profiles to the *local* pseudo-interface of the IP context. As shown in Figure 52, these profiles are applied to all packets that are generated locally, independent of the IP interface over which they are being sent.

The **use** command is used to bind a classifier profile to the *local* pseudo-interface. This procedure describes how to bind a classifier profile to filter locally-generated packets

Mode: Context IP

Step	Command	Purpose
1	***device*(ctx-ip)**[ROUTER]**#local**	Enters the configuration mode of the pseudo-interface for locally-generated traffic.
2	***device*(local)**[ROUTER]**#use profile classifier out** <*profile-name*>	Binds the classifier profile <*profile-name*> to locally-generated packets.

The command offers the following options:

Parameter	Explanation
<*profile-name*>	The name of the classifier profile that has already been created using the profile classifier command. This argument must be omitted in the no form.
out	Specifies that the classifier applies to locally-generated packets.

The **no** form of the **use** command is used to unbind a classifier profile from the local pseudo-interface. When using this form, instead of specifying the name of the profile to unbind, you have to give the numeric index of the respective **use** command as it can be found in the current running-config.

Mode: Context IP

Step	Command	Purpose
1	***device*(ctx-ip)**[ROUTER]**#local**	Enters the configuration mode of the pseudo-interface for locally-generated traffic.
2	***device*(local)**[ROUTER]**#use profile classifier out** *index*	Unbinds the classifier profile at *index* for locally-generated packets.

Parameter	Explanation
index	Numeric index of the bind command (type show running-config current-mode to retrieve the index).
out	Specifies that the classifier applies to packets sent over this interface.

Example: Bind and unbind a classifier profile for locally-generated packets.

Bind a classifier profile to locally-generated packets in the default IP context (*ROUTER*)

```
device(cfg)#context ip ROUTER
device(ctx-ip)[ROUTER]#local
device(local)[ROUTER]#use profile classifier out GENERIC
```

Show the current configuration of the *local* pseudo-interface:

```
device(local)[ROUTER]#show running-config current-mode
use profile classifier out 1 GENERIC
```

Unbind the *GENERIC* classifier profile from the *local* pseudo-interface:

```
device(local)[ROUTER]#no profile classifier out 1
```

Displaying a Classifier Profile

The **show profile classifie**r command displays the indicated classifier profile. If no specific profile is specified, all installed classifier profiles are shown. The **show** command displays a list of all classifier rules, as well as a list of IP interfaces to which it is bound.

This procedure describes how to display one specific or all classifier profiles

Mode: any

Step	Command	Purpose
1	***device*>show profile classifier** [<*profile-name*>]	Displays the rules and bindings of classifier profile <*profile-name*>, or, if the <*profile-name*> parameter is absent, displays all classifier profiles.

Example: Display a classifier profile

The following example shows how to display the classifier profile named WAN_RX.

```
device>show profile classifier WAN_RX
Classifier Profile: WAN_RX
--------------------------

Rules
-----
match protocol udp dest-port 23,80 set traffic-class MGMT
match protocol tcp dest-port 23,80 set traffic-class MGMT

References: 1
-------------
IP Interface(s) (inbound): WAN (ROUTER)
```

Chapter 35 Service Policy Configuration

Chapter contents

Introduction

This chapter describes how to use and configure service-policy profiles. Service-policy profiles can be bound to individual IP interfaces to map internal traffic classes to QoS-related IP header fields. For an overview of Trinity's Quality-of-Service architecture and the meaning of traffic classes, please refer to Chapter 31, "Quality of Service (QoS) Overview" on page 315.

Service Policy Configuration Task List

To configure service policy profiles, perform the tasks in the following sections:

- "Creating a service policy profile" on page 350
- "Set QoS-related IP header field" on page 351
- "Binding a classifier profile to the outbound traffic of an IP interface" on page 351

Creating a service policy profile

The service-policy profile defines whether and how to set IP header fields such as TOS, precedence, DSCP, and ECN based on the traffic-class tag of an outgoing packet.

This procedure describes how to create a service-policy profile and enter the profile's configuration mode.

Mode: Administrator exec

Step	Command	Purpose
1	**device(cfg)#profile service-policy** *<profile-name>*	Creates the service-policy profile *<profile-name>* and enters its configuration mode.

The parameter *<profile-name>* is the identifier by which the profile will be known. Entering this command puts you into service-policy profile configuration mode where you can enter traffic-class-specific modes as shown in the next section.

Use the **no** form of this command to delete a service-policy profile. You cannot delete a service-policy profile if it is currently bound to an interface. When you modify a service-policy profile, the new settings immediately become active.

Example: Create a service-policy profile

In the following example the service-policy profile named *SP_WAN* is created.

```
device>enable
device#configure
device(cfg)#profile service-policy SP_WAN
device(pf-svcpol)[SP_WAN]#
```

Set QoS-related IP header field

When a service-policy profile is bound to an IP interface, the profile may set IP header fields such as TOS, precedence, DSCP, and ECN of all packets sent over that IP interface. For each internal traffic class, a separate set of header field values can be configured.

Mode: Profile service-policy

Step	Command	Purpose
1	**device(svcpol-cls)**[*pfname*]**#source traffic-class** *traffic-class*	Enters a sub-mode that combines configuration commands for packets tagged with the specified *traffic-class.*
2a	**device(src-tc)**[*traffic-class*]**#set tos** *tos*	Sets the IP type-of-service header field to *tos*.
3a	**device(src-tc)**[*traffic-class*]**#set precedence** *precedence*	Sets the IP precedence header field to *precedence.*
2b	**device(src-tc)**[*traffic-class*]**#set dscp** *dscp*	Sets the IP differentiated-services-code-point header field to *dscp.*
3b	**device(src-tc)**[*traffic-class*]**#set ecn** *ecn*	Sets the IP explicit-congestion-notification field to *ecn.*

Note The TOS/precedence header fields use the same bits as the DSCP/ECN fields. Thus you can either configure TOS and/or precedence values or DSCP and/or ECN values. If you for example try to enter a DSCP value after having configured a TOS value, the DSCP value is not accepted and you will be presented with an error message.

Example: Set IP TOS and precedence fields

The following example prepares a service-policy profile to set the IP TOS field to 5 for all packets tagged with the *MGMT* traffic class. Note that the profile must be bound to an IP interface in order to get activated (see section below).

```
device(cfg)#profile service-policy SP_WAN
device(pf-svcpol)[SP_WAN]#source traffic-class MGMT
device(src-tc)[MGMT]#set tos 5
```

Binding a classifier profile to the outbound traffic of an IP interface

To apply a service-policy profile to the outbound traffic it has to be bound to an IP interface. This procedure describes how to do so with the **use** command.

Mode: IP Interface

Step	Command	Purpose
1	**device(if-ip)**[*ifname*]**#use profile service-policy [in|out]** *pfname*	Binds the service-policy profile *pfname* to outgoing packets on the IP interface *ifname.*

The command offers the following options:

Parameter	Explanation
ifname	Name of the IP interface to which a service-policy profile gets bound.
pfname	The name of a service-policy profile that has already been created using the **profile service-policy** command. This argument must be omitted in the no form.
in	Specifies that the service-policy profile applies to packets received over this interface. This currently is of limited use as packet tagging should always happen at an egress point of the device to match the QoS-field semantics of the target network.
out	Specifies that the service-policy profile applies to packets sent over this interface.

The **no** form of the **use** command is used to unbind a service-policy profile from an IP interface. When using this form, you don't have to specify the profile name.

Mode: IP Interface

Step	Command	Purpose
1	**device(if-ip)**[*ifname*]**#no use profile service-policy [in|out]**	Uninds the service-policy profile for incoming or outgoing packets on the IP interface *ifname.*

Parameter	Explanation
ifname	Name of the IP interface to which a service-policy profile gets bound.
in	Specifies that the service-policy profile applies to packets received over this interface. This currently is of limited use as packet tagging should always happen at an egress point of the device to match the QoS-field semantics of the target network.
out	Specifies that the service-policy profile applies to packets sent over this interface.

Example: Bind and unbind a service-policy profile.

Bind the service-policy profile created in the previous example to tag outgoing packets on interface *WAN* in the default IP context (*ROUTER*).

```
device(cfg)#context ip ROUTER
device(ctx-ip)[ROUTER]#interface WAN
device(ip-if)[ROUTER.WAN]#use profile service-policy out SP_WAN
```

Unbind the *SP_WAN* service-policy profile from outgoing traffic on interface *WAN*:

```
device(ip-if)[ROUTER.WAN]#no use profile service-policy out
```

Chapter 36 Packet Matching

Chapter contents

Introduction

Several features perform operations on packets based upon certain criteria, such as, their headers. For example, the Classifier feature assigns packets to different traffic classes, the Policy Routing feature routes packets according to different tables and the ACL feature permits or denies packets, depending on the same criteria. All of these features use the same command line syntax to specify which packets match. This chapter lists the criteria that may be used to match packets and specifies the command line syntax.

Criteria

This section lists the criteria that may be used to match packets.

Connection State

This matches packets based on the connection state, which include the following:

- **New**—This matches the first packet the router has received for a given connection. For example, the first packet in an FTP connection is a TCP SYN packet sent from the client to the server. This packet is **new**.
- **Established**—This matches replies to **new** packets, and all subsequent packets in the given connection. For example, when an FTP server responds to a client with a TCP SYN/ACK packet, this packet is **established**.
- **Related**—This matches packets from a connection initiated by another **established** connection. For example, the packets belonging to an FTP-data connection are **related** to the FTP-control connection.
- **Invalid**—This usually indicates a system error, such as being out of memory.

An ACL rule may be added to match packets in **established** and **related** connections in order to implement a stateful firewall.

Traffic Class

This matches packets that have been assigned to a given traffic class by a Classifier rule.

Source MAC Address

This matches packets with a given source MAC address in the Ethernet header.

Note This is only valid for packets received by the router and not for packets generated by the router itself.

Ethernet Packet Type

This matches packets based on whether they are Ethernet unicast packets, broadcast packets (i.e. the destination MAC address is FF:FF:FF:FF:FF:FF), or multicast packets (i.e. the destination MAC address has bit 01:00:00:00:00:00 set).

ToS

This matches packets based on the ToS bits in the IPv4 ToS field.

Note This only applies to IPv4 packets and not IPv6 packets.

Precedence

This matches packets based on the precedence bits in the IPv4 ToS field.

Note This only applies to IPv4 packets and not IPv6 packets.

DSCP

This matches packets based on the DSCP bits int he IP DiffServ field.

ECN

This matches packets based on the ECN bits int he IP DiffServ field.

Length

This matches packets based on the IP payload length.

TTL

This matches packets based on the IPv4 Tim-To-Live or IPv6 Hop Count field.

Protocol

This matches packets based on the IP protocol, for example, ICMP, ICMPv6, TCP, or UDP.

Source IP Address

This matches packets based on the IP source address.

Destination IP Address

This matches packets based on the IP destination address.

ICMP Type/Code

This matches ICMP or ICMPv6 packets based on the ICMP type, and optionally the ICMP code.

Source Port

This matches TCP, UDP, and SCTP packets based on the source port.

Destination Port

This matches TCP, UDP, and SCTP packets based on the destination port.

TCP Flags

This matches TCP packets based on the TCP flags: SYN, ACK, FIN, RST, URG, and PSH.

TCP Option

This matches TCP packets based on whether or not a given TCP option is set.

TCP MSS

This matches TCP packets based on the maximum segment size (MSS).

Note This only applies to TCP SYN and SYN/ACK packets because the MSS is only negotiated during the TCP handshake.

Command Line Syntax

This section explains how to specify the criteria listed above using the CLI.

The following syntax may be used at the CLI to specify the packet match criteria.

Note Multiple criteria may be specified in the same command, in which case, all of the specified criteria must be met for a packet to be considered a match.

Table 26. Command Line Syntax

Criterion	Syntax	Notes
Connection State	connection-state [new] [established] [related] [invalid]	If the connection-state keyword appears, at least one state must be specified.
Traffic Class	traffic-class <name>	The traffic class must have been previously assigned to the packet by the Classifier.
Source MAC Address	src-mac <aa:bb:cc:dd:ee:ff>	
Ethernet Packet Type	packet-type {unicast | broadcast | multicast}	
ToS	tos {<0..15>|maximize-cost | maximize-reliability | maximize-throughput | minimize-cost | minimize-delay | normal} [mask <0..15>]	An optional mask may be specified to limit which bits are examined.
Precedence	precedence <0..7>	
DSCP	dscp {<0..63> | af11 | ... | ef} [mask <0..63>]	An optional mask may be specified to limit which bits are examined.
ECN	ecn <0..3> [mask <0..3>]	An optional mask may be specified to limit which bits are examined.
Length	length <0..65535>[..<0..65535>]	A single length, e.g. 500, or a range, e.g. 28..1480, may be specified. Note that either the minimum or maximum may be omitted in which case it is implied to be 0 or 65535, respectively.
TTL	ttl {eq | lt | gt} <0..255>	Match packets having a TTL (IPv4) or Hop Count (IPv6) equal to, less than, or greater than the given value.
Protocol	protocol {<0..255> | icmp | icmpv6 | tcp | udp}	
Source IP Address	src-ip *address*	The address may be an IPv4 or IPv6 host, e.g. 192.168.1.1, subnet, e.g. 192.168.1.0/24, or range, e.g. 192.168.1.1-192.168.1.10.
Destination IP Address	dest-ip *address*	The address may be an IPv4 or IPv6 host, e.g. 192.168.1.1, subnet, e.g. 192.168.1.0/24, or range, e.g. 192.168.1.1-192.168.1.10.
ICMP Type/ Code	icmp-type {0..255> | echo-reply | destination-unreachable | ...} [<0.255>]	The first value is the type and the second, optional value is the code. You must also specify `protocol icmp` to use this criterion.

Table 26. Command Line Syntax

Criterion	Syntax	Notes
ICMPv6 Type/ Code	icmpv6-type {<0..255> | echo-reply | destination-unreachable | ...} [<0.255>]	The first value is the type and the second, optional value is the code. You must also specify `protocol icmpv6` to use this criterion. **Note** ICMPv6 types may have the same name as an ICMP type, but a different numeric value.
Source Port	src-port <0..65535	The port may be a single value, a list, or a range. For example, `22`,`22,23,80`, `1024..65535`, and `22,23,80,1024..` are all valid. You must also specify `protocol tcp, protocol udp` or `protocol sctp` to use this criterion.
Destination Port	dest-port <0..65535>	The port may be a single value, a list, or a range. For example, `22`,`22,23,80`, `1024..65535`, and `22,23,80,1024..` are all valid. You must also specify `protocol tcp, protocol udp` or `protocol sctp` to use this criterion.
TCP Flags	tcp-flags [syn {set | clear}] [ack {set | clear}] [fin {set | clear}] [rst {set | clear}] [urg {set | clear}] [psh {set | clear}]	Specify which TCP flags to examine and their value. If the flag is not specified, then it may have either value, set (1) or clear (0). You must also specify `protocol tcp` to use this criterion.
TCP Option	tcp-option <0..255>	You must also specify `protocol tcp` to use this criterion.
TCP MSS	tcp-mss <0..65535>[..<0..65535>]	A single MSS, e.g. 500, or a range, e.g. 28..1480, may be specified. **Note** Either the minimum or maximum may be omitted in which case it is implied to be 0 or 65535, respectively. You must also specify `protocol tcp` to use this criterion.

Examples

```
profile classifier CLASSIFIER_LAN
 # Assign packets destined for an SSH, Telnet, or HTTP server to traffic class
 # MGMT.
 match protocol tcp dest-port 22,23,80 set traffic-class MGMT
 # Assign packets destined for an FTP or TFTP server to traffic-class DATA.
 match protocol tcp dest-port 20,21 set traffic-class DATA
 match protocol udp dest-port 69 set traffic-class DATA

profile acl ACL_WAN
 # Allow established and related connections.
 permit connection-state established related

context ip

 interface LAN
   ipaddress 192.168.1.1/24
   use profile classifier in 1 CLASSIFIER_LAN
   # Route packets from 192.168.1.254 to the 172.16.2.0/24 subnet according to
   # TABLE1.
   route src-ip 192.168.1.254 dest-ip 172.16.2.0/24 dest-table TABLE1
   # Route all other packets according to the DEFAULT routing table.
   route dest-table DEFAULT

 interface WAN
   ipaddress 172.16.1.10/24
   use profile acl in 1 ACL_WAN

 route DEFAULT
   route default gateway 172.16.1.1

 route TABLE1
   route default gateway 172.16.1.2
```

Chapter 37 SIP Profile Configuration

Chapter contents

Introduction

The SIP profile specifies disconnect cause mappings from SIP codes to Q.931 causes, and vice versa. As for all profiles, there is a *default* profile at system startup that can be modified. Only those causes that differ from the default mapping have to be configured. If a new profile is created, all mappings are set to their default and are only overwritten if configured. The default mapping in both directions is according to RFC3398 - ISUP to SIP Mapping.

A SIP profile can either be attached to SIP interfaces or to identities. To see how to configure a SIP profile for an interface, see chapter 48, "SIP Interface Configuration" on page 538. For information about SIP profile configuration for identities, see chapter 44, "Location Service" on page 434.

SIP Profile Configuration Task List

This section describes the configuration tasks for SIP profile listed below.

- Enter the configuration mode for a SIP profile (see page 360)
- Map *from* a SIP disconnect cause *to* a Q.931 cause (see page 360)
- Map *to* a SIP cause *from* a Q.931 disconnect cause (see page 361)
- Map *from* a SIP redirection code *to* a Q.931 redirect reason (see page 361)
- Map *to* a SIP redirection code *from* a Q.931 redirect reason (see page 361)

Entering the Configuration Mode for a SIP Profile

The **profile sip** command enters the configuration mode of an existing profile or creates a new one with a specified name. It also destroys an existing profile except the default, which always exists.

Mode: Configure

Step	Command	Purpose
1	**[name](cfg)#[no] profile sip**<*name*>	Creates/Destroys a SIP profile or enter the configuration mode of an existing one.

Mapping from a SIP Disconnect Cause

The **map cause from-sip** command maps a specific SIP disconnect cause to a Q.931 cause used by the call control. All causes are pre-defined in the system and are provided by the command.

Mode: Profile SIP

Step	Command	Purpose
1	**[name](pf-sip)[name]#map cause from-sip** *sip-cause* **to** *q931-cause*	Maps a specific SIP disconnect cause to a Q.931 cause.

Mapping to a SIP Cause

The **map cause to-sip** command can be used to map a call control Q.931 cause to a SIP cause. All causes are pre-defined in the system and are provided by the command.

Mode: Profile SIP

Step	Command	Purpose
1	**[name](pf-sip)[name]#map cause to-sip** *q931-cause* **to** *sip-cause*	Maps a specific Q.931 disconnect cause to a SIP cause code.

Mapping from a SIP Redirection Reason

The **map redir-reason from-sip** command can be used to map a specific SIP redirect code to a Q.931 redirect reason used by the call control. All redirect codes and reasons are pre-defined in the system and are provided by the command.

Mode: Profile SIP

Step	Command	Purpose
1	**[name](pf-sip)[name]#map redir-reason from-sip** *code* **to** *reason*	Maps a SIP redirection code to a Q.931 redirection reason.

Mapping to a SIP Redirection Code

The **map redir-reason to-sip** command can be used to map a Q.931 redirect reason to a specific SIP redirect code. All redirect codes and reasons are pre-defined in the system and are provided by the command.

Mode: Profile SIP

Step	Command	Purpose
1	**[name](pf-sip)[name]#map redir-reason to-sip** *reason* **to** *code*	Maps a Q.931 redirect reason to a SIP redirect code.

SIP-Tunneling

The newly introduced profile sip-tunneling allows tunneling of SIP headers from SIP to SIP calls. Each profile defines a set of SIP headers and a set of SIP messages where the tunneling should be active. A profile bound in incoming direction to a sip interface parses the according messages for the specified headers and forwards the headers to the peer session. A profile bound in outgoing direction inserts all the headers which it get passed from its peer into all specified messages.

Overwriting Headers incoming from new messages overwrite the previously stored headers of the same type. When there are no headers of a certain type in a message then the previously stored headers form older messages still remain active.

Messages triggering Changes of headers which are tunneled do not trigger the sending of new messages. Messages containing the new headers are only sent when the SIP call-flow is in the according state and a signaling message has to be sent anyway.

Excluded headers There is a list of headers which cannot be tunneled because it would destroy the call-flow of the sip calls. These headers are:

- Authorization
- Call-ID
- Contact
- Content-Description
- Content-Disposition
- Content-Encoding
- Content-ID
- Content-Language
- Content-Length
- Content-Transfer-Encoding
- Content-Type
- CSeq
- From
- Max-Forwards
- MIME-Version
- Proxy-Authenticate
- Proxy-Authorization
- RAck
- Record-Route
- Require
- Retry-After
- Route
- RSeq
- To
- Via
- WWW-Authenticate

Caution for these headers These headers may or may not inserted into sip messages from the normal SIP application. This highly depends on the configured features and the call-flows which happens. When tunneling these headers this may lead to overwriting or duplication of headers. But the device should still be able to perform normal SIP calls, even when having conflict in these headers.

- Accept

- Allow
- Diversion
- Event
- Expires
- History-Info
- Min-Expires
- Min-SE
- P-Asserted-Identity
- P-Preferred-Identity
- Privacy
- Proxy-Require
- Refer-To
- Referred-By
- Replaces
- Server
- Session-Expires
- SIP-If-Match
- Subscription-State
- Supported
- Timestamp
- Unsupported
- User-Agent

Mode: configure

Step	Command	Purpose
1	**node(cfg)#profile sip-tunneling <profile name>**	Create a new or enter an existing profile sip tunneling.

Mode: sip-tunneling

Step	Command	Purpose
2	**node(pf-tun)[<name>]#[no] header <header name>**	Adds or removes a sip header type to be tunneled to or from the profile. The header names are case insensitive.

Mode: sip-tunneling

Step	Command	Purpose
3	**node(pf-tun)[<name>]#message { INVITE | INFO | CANCEL | BYE | 180 | 183 | 200 | 3xx | 4xx | 5xx }**	Sets the exact list of messages where the profile should be active parsing headers inbound or sending headers outbound.
3	**node(pf-tun)[<name>]#message add { INVITE | INFO | CANCEL | BYE | 180 | 183 | 200 | 3xx | 4xx | 5xx }**	Adds messages to the list of already configured ones.
3	**node(pf-tun)[<name>]#message default**	Sets the list of active messages back to the defaults. Which is INVITE, CANCEL, BYE, 180, 183, 200, 3xx, 4xx and 5xx.
3	**node(pf-tun)[<name>]#message all**	Activates the profile on all possible messages. Which is INVITE, INFO, CANCEL, BYE, 180, 183, 200, 3xx, 4xx and 5xx.
3	**node(pf-tun)[<name>]#no message { INVITE | INFO | CANCEL | BYE | 180 | 183 | 200 | 3xx | 4xx | 5xx }**	Removes the specified messages from the profile, to not be active on these anymore.

Mode: context cs/interface sip

Step	Command	Purpose
4	**node(if-sip)[<name>]#[no] use profile sip-tunneling <profile name> { in | out | both }**	Activates a profile to be used on the sip interface in the specified direction.

Autonomous Transitioning for SIP

The autonomous transitioning command provides simultaneous media for audio and media for image in an INVITE request. The opening of a port for audio and a second port for image allows to switch from a normal call to fax relay T38. This is accomplished by sending packets to the other port, without the need of sending a re-INVITE when fax is detected.

Note This feature should not be enabled when the call is passing through NAT or Firewall.

Mode: Profile SIP

Step	Command	Purpose
1	**[name](pf-sip)[name]# [no] autonomous-transitioning**	Enables or disables the autonomous transitioning.

Chapter 38 VoIP Profile Configuration

Chapter contents

Introduction

This chapter gives an overview of VoIP profiles, and describes how they are used and the tasks involved in VoIP profile configuration.

A VoIP profile is a container for all datapath-related settings on VoIP connections. The profile settings apply to all calls going through the interface. A VoIP profile can be assigned to VoIP gateways and VoIP interfaces in context CS. If no profile is specified for a particular interface, a profile from the gateway the interface binds to is used instead. figure 53 illustrates the relations between VoIP profiles, gateways and CS interfaces. The following components are configurable:

- Codecs and codec parameters (such as silence suppression, RTP payload type, and audio filters)
- DTMF relay
- Dejitter buffer
- Fax transmission
- Modem transmission

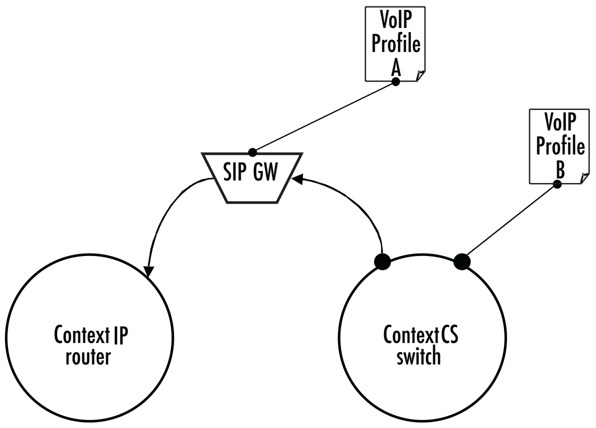


Figure 53. VoIP profile association

IMPORTANT

Configuring voice datapath options can improve ***or degrade*** the quality of the transmitted voice data. Many of the default values of these components have configured defaults that should only be changed if required. Misconfiguration can strongly affect the voice quality perceived by the user and the bandwidth requirements of VoIP connections. Be sure you understand the meaning and impact of all commands prior to changing any settings.

VoIP Profile Configuration Task List

The following tasks describe components that can be configured through the VoIP profile:

- Creating a VoIP profile
- Configuring codecs
- SDP Ptime attribute
- Enabling DTMF relay (see page 368)
- Configuring RTP payload types (see page 372)
- Configuring the dejitter buffer (advanced) (see page 372)
- Enabling/disabling filters (advanced) (see page 376)
- Configuring fax transmission (see page 377)
- Configuring modem transmission (see page 381)

If a VoIP profile is modified, the saved modification is applied to all open calls and is valid for all future calls on the gateway or interface using this VoIP profile.

Creating a VoIP Profile

Before configuring voice parameters, a VoIP profile must be created. Each VoIP profile has a name that can be any arbitrary string of not more than 25 characters. When you create the VoIP profile, the VoIP profile configuration mode appears so you can configure VoIP components.

Note The VoIP profile named *default* always exists in the system. It is used by all interface components if there is no other VoIP profile available. If VoIP parameters are the same throughout all interfaces, you can simply change the profile *default* instead of creating a new profile.

Procedure: Create a VoIP profile and enter the VoIP profile configuration mode

Mode: Configure

Step	Command	Purpose
1	**device(cfg)#profile voip** *name*	Creates a VoIP profile with name and goes into VoIP profile configuration mode. The newly created profile contains default values for all parameters. If a profile with the same name already exists, only that VoIP profile configuration mode is entered.
2	**device(pf-voip)device#...**	Configuration steps are described in the following sections.

Example: Creating a VoIP profile

This example shows how to create a VoIP profile named g729_FaxRelay and enter into VoIP profile configuration mode.

```
device>enable
device#configure
```

```
device(cfg)#profile voip g729_FaxRelay
device(pf-voip)[g729_fa~]#...
```

Configure Codecs

The VoIP profile contains a *list* of codecs the forms the set of allowed codecs that can be used to set up a VoIP connection. The list is assembled in order of priority (i.e. the first entered codec is the most preferred one). For each codec in the list, a set of parameters can be configured.

IMPORTANT

Signaling protocols have a codec negotiation mechanism, it is not guaranteed that the first codec in the list is used to set up the connection. Each codec in the list may be used. To make sure that only one codec is possible, configure this codec alone. See how to display the currently configured codecs in a VoIP profile on page 381.

IMPORTANT

The default VoIP profile contains the codecs G.711uLaw and G.711aLaw. If you don't want to use these, you must explicitly remove them from the list.

Procedure: Add a codec to the list (this procedure is valid for all other codecs as well).

Note If you press the **<tab>** key after entering a few letters of a configuration command, the full command name will display or a listing of commands that begin with those letters will display. Press the **<enter>** key to select the desired command.

Mode: Profile VoIP

Step	Command	Purpose
1	**device(pf-voip)device#codec g729 tx-length 30 rx-length 30 silence-suppression**	Appends codec g729 to the list of codecs. Specifies the payload duration for transmitted RTP packets of this codec, and the maximum supported payload duration for received RTP packets of this codec. Allows silence suppression to be used with this codec. If the codec g729 already existed in the list, its parameters are updated with the entered values. The following codecs are available: • g711alaw64k • g711ulaw64k • g723-5k3 • g723-6k3 • g726-16k • g726-16k-cisco[a] • g726-24k • g726-24k-cisco • g726-32k • g726-32k-cisco • g726-40k • g727-16k • g727-24k • g727-32k • g729 • netcoder-6k4 • netcoder-9k6 • transparent • transparent-cisco

a. Cisco does not use the standard ITU G.726 version of G.726, but the ATM AAL2 version. This build series now supports both versions of these codecs. The Cisco G.726 codecs are available in *profile voip* as separate codecs with their name ending in *-cisco*.

Procedure: Remove a codec from the list

Mode: Profile VoIP

Step	Command	Purpose
1	**device(pf-voip)device#no codec g729**	Remove codec g729 from the list of codecs.

Procedure: Insert a codec at a specific position in the list

Mode: Profile VoIP

Step	Command	Purpose
1	**device(pf-voip)device#codec 1 g729 tx-length 30 rx-length 30 silence-suppression**	Inserts codec g729 at the first position of the list (most preferred codec). The parameters are the same previously described. If the codec g729 had yet existed in the list, it is moved to the first position of the list, adopting the entered parameter values.

Configuring the Transparent-clearmode codec

To be compatible with RFC4040, transparent-clearmode was made available as a codec in the voip profile. The codec can be used if exclusively packetization and no coding/decoding is needed.

Mode: configure/profile voip

Step	Command	Purpose
1	**device(pf-voip)**[*profile*]**#[no] codec transparent-clearmode**	Allows to use the codec transparent-clearmode.

Configuring the Cisco Versions of the G.726 Codecs

The Cisco versions of codecs are listed in the previous section as separate codecs with their name ending in –Cisco. Trinity supports four Cisco codec versions: *g726-16k-cisco*, *g726-24k-cisco*, *g726-32k-cisco*, and *transparent-cisco*. Three of the codecs are variations of G.726, the fourth is *transparent-cisco*.

The *transparent-cisco* codec provides full compatibility with Cisco's *clear-channel* codec used for transmission of Unrestricted Digital Information over a VoIP (SIP) network.

Cisco does not use the standard ITU G.726 version of G.726, instead it uses the ATM AAL2 version.

All supported Cisco codecs are available in profile voip.

Mode: VoIP *name*

Step	Command	Purpose
1	**device(pf-voip)device#codec** { g726-16k-cisco | g726-24k-cisco | g726-32k-cisco | … }	To operate with Cisco's G.726 codecs.

The next table indicates the method of configuring a Cisco-variant codec as the most preferred codec. This example sets the 'transparent-cisco' as number 1, the most preferred.

Mode: VoIP *name*

Step	Command	Purpose
1	**device(pf-voip)device#codec 1 transparent-cisco**	Configures transparent-cisco as the most preferred codec.

Configuring the AAL2-G.726-32k Codec

It is possible to configure AAL2-G726-32k codec to be available as an option in SDP negotiation.

Mode: profile voip

Step	Command	Purpose
1	**device(pf-voip)device#[no] codec** [before | after] **[index] g726-32k-aal2**	Enables AAL2-G726-32k codec for negotiation. Please note that specifying this codec automatically disables any other g726-32k codec (g726-32k and g726-32k-cisco). Default: disabled.

It is recommended to specify a custom payload type for the newly introduced AAL2-G726-32k codec (current value is 2).

Mode: profile voip

Step	Command	Purpose
1	**device(pf-voip)device#[no] rtp payload-type g726-32k-aal2** *<value>*	Sets the new payload type value for aal2-g726-32k codec (in range 96..127)

SDP ptime Attribute

The SDP ptime attribute announces the maximum receive duration for the offered coders. Because ptime can only be offered on media level and not on a per coder basis, Trinity selects the rx-length of the first configured coder as value for the ptime attribute. By default the new attribute is not included in SIP's SDP content. It can be configured to be included with the voip profile's sdp-ptime-announcement command.

Mode: profile voip

Step	Command	Purpose
1	**node(pf-voip)[name]#[no] sdp-ptime-announce-ment**	Enables/Disables announcement of the ptime attribute in SIP's SDP content. Default: disabled

Configuring DTMF Relay

Dual tone multi-frequency (DTMF) tones are usually transported accurately in band when using high bit-rate voice codecs such as G.711. Low bit-rate codecs such as G.729 and G.723.1 are highly optimized for voice patterns and tend to distort DTMF tones. The **dtmf relay** command solves the problem of DTMF distortion by transporting DTMF tones out-of-band or separate from the encoded voice stream as shown in figure 54. SIP uses a mechanism of RTP to reliably transport tones (according to RFC2833).

If "dtmf-relay rtp" (RFC2833) is enabled, the DTMF tones are sent as RTP EVENTS and the DTMF tones are filtered from the RTP media stream. With this enabled, you will only hear a very short tone inband before DTMF detection and filtering kicks in. The default configuration in the voip profile is "dtmf-relay default", which defaults to "dtmf-relay rtp" when SIP is used as the signaling protocol. So, RFC2833 is enabled by default. If you need to send DTMF tones inband you have to disable dtmf-relay, as see in the table below.

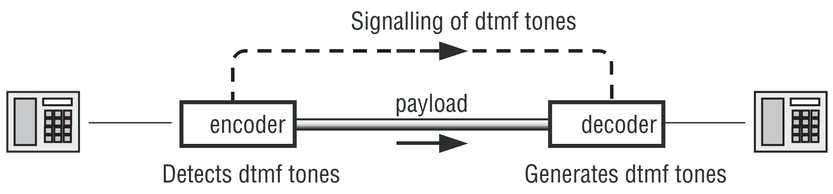


Figure 54. DTMF Relay

This procedure describes how to configure DTMF relay.

Mode: Profile VoIP

Step	Command	Purpose
1	**device(pf-voip)device#dtmf-relay signaling** [default | broadsoft]	The **dtmf-relay signaling default** command configures the SIP INFO message to be sent in Patton proprietary format. The **dtmf-relay signaling broadsoft** command configures the SIP INFO message to be sent in Broadsoft proprietary format.
2	**device(pf-voip)device#[no] dtmf-relay**	Using the **no dtmf-relay** command, the dtmf are passed INBAND together with the voice in the RTP flow

Step	Command	Purpose
1	**device(pf-voip)device#flash-hook-relay** [dtmf | rtp | signaling [broadsoft]]	With the “flash-hook-relay” command the user can chose a different relay method for flash-hook than for the other DTMF keys. The default setting is dtmf.

- **flash-hook-relay dtmf:** flash-hook signals are relayed with the same method as all other DTMF keys. This is the default behavior
- **rtp:** flash-hook signals are sent as rfc2833 RTP events
- **signaling**: flash-hook signals are sent as standard SIP Info messages
- **signaling broadsoft:** flash-hook signals are sent as Broadsoft SIP Info messages

Restrictions:

Since inband-transmission and rfc2833 do not work concurrently, flash-hook-signaling method rtp is not supported when dtmf-relay is disabled or dtmf-relay method is inband. All other combinations work.

Configuring RTP Payload Types

The RTP payload type is one of RTP’s header fields. It identifies the format (e.g. encoding) of the RTP payload and determines the interpretation of the application.

Procedure: Configure RTP NTE payload type

Mode: Profile VoIP

Step	Command	Purpose
1	**device(pf-voip)device#rtp payload-type nte** *payload-type*	Specifies the RTP payload-type for named tone events NTE (RFC2833). Default: 101.

Configuring RTP Payload Type for Transparent

The following command configures the RTP payload type used for the transparent codec.

Mode: profile voip

Step	Command	Purpose
1	**device(pf-voip)**[*profile*]**#[no] rtp payload-type transparent** <*value*>	Specifies the RTP payload type used for the transparent codec. Value must be between 0 and 127. Default value is 91.

Configuring RTP Payload Type for Transparent-cisco

The following command configures the RTP payload type used for the transparent-cisco codec.

Mode: profile voip

Step	Command	Purpose
1	**device(pf-voip)**[*profile*]**#[no] rtp payload-type transparent-cisco** <*value*>	Specifies the RTP payload type used for the transparent-cisco codec. Value must be between 96 and 127. Default value is 116.

Configuring RTP Payload Type for Transparent-clearmode

The following command configures the RTP payload type used for the transparent-clearmode codec.

Mode: profile voip

Step	Command	Purpose
1	**device(pf-voip)**[*profile*]**#[no] rtp payload-type transparent-clearmode** <*value*>	Specifies the rtp payload type used for transparent clearmode. Value must be between 96 and 127. Default value is 97.

Configuring RTP Payload Types for the g726-32k and g726-32k-cisco Coders

The following command specifies the RTP payload types for the g726-32k and the g726-32k-cisco coders to be used. It allows changing the payload types to a value in the range of 96 to 127 whereas the default value is 2 for both. Once a payload type has been changed, the ‘no’ form of the command must be used to go back to the default value.

Mode: profile voip

Step	Command	Purpose
1	**device(pv-voip)device#[no] rtp payload-type** [g726-32k | g726-32k-cisco] <value 96..127>	Defines the RTP payload types for the g726-32k and the g726-32k-cisco coders. Default: 2

Configuring RTP Payload Type for Cisco NSE

Configure the RTP payload-type when transmitting the NSE events. This payload-type is negotiated during call-setup when using SIP.

Named Service Events (NSE) are the Cisco-proprietary version of Named Telephony Events (NTEs). NTEs are defined in RFC 2833. Various telephony signaling events use tones, for example, DTMF. NSEs and NTEs communicate these tones (for representing signaling events), not by the presence of tones, but by sending a binary code representing the tone that is recreated at the destination. Cisco's proprietary NSEs use different values to represent tones and events than the NTEs use.

NSEs are normally sent with RTP payload type 100. The RTP packets have the same source and destination IP addresses and UDP ports as the other packets in the media stream, but differ in the RTP payload types so they can be distinguished from the stream's audio packets.

Mode: Profile VoIP

Step	Command	Purpose
1	device(pf-voip)[*name*]#**rtp payload-type nse payload-type**	Specifies the RTP payload-type for Named Signaling Events (NSE). Default: *100*

Configuring Cisco NSE for Fax

This command specifies the method to be used for signaling the remote device the RTP Stream has switched to a voice-band Modem transmission. This feature is only available on the SIP protocol. If the command option 'v150-vbd' is selected, a re-invite will be sent even if the current voice coder is configured the same as the modem bypass coder. Furthermore the re-invite contains a gpmd-attribute line with the value 'vbd=yes;ecan=off ' in the media description part. This attribute signals the remote device of the new media transmission. If the command option 'default' is selected, the system behavior is the same as before.

Trinity also supports the Cisco NSE standard which uses RFC2833 events for modem transmission over a VoIP (SIP) network. Upon detecting a modem transmission, the called peer issues NSE Event 192. NSE Event 192 indicates a Voice Band Data stream that forces the calling peer to deactivate Voice Activity Detection and to reconfigure the de-jitter buffer for data reception. Afterwards it issues the NSE Event 193 to trigger the calling peer to switch off Echo-Cancellation.

Configuring the Dejitter Buffer (advanced)

Packet networks always introduce a certain amount of jitter in the arrival of voice packets. To compensate for the fluctuating network conditions, a dejitter buffer is integrated in the RTP processing engine. Typical voice sources generate voice packets at a constant rate, the matching voice decompression algorithm also expects incoming voice packets to arrive at a constant rate. However, the packet-by-packet delay inflicted by the network may be different for each packet. As shown in figure 55, the result of the delays is that packets which are sent equally spaced from the left-hand gateway arrive irregularly spaced at the right-hand gateway.

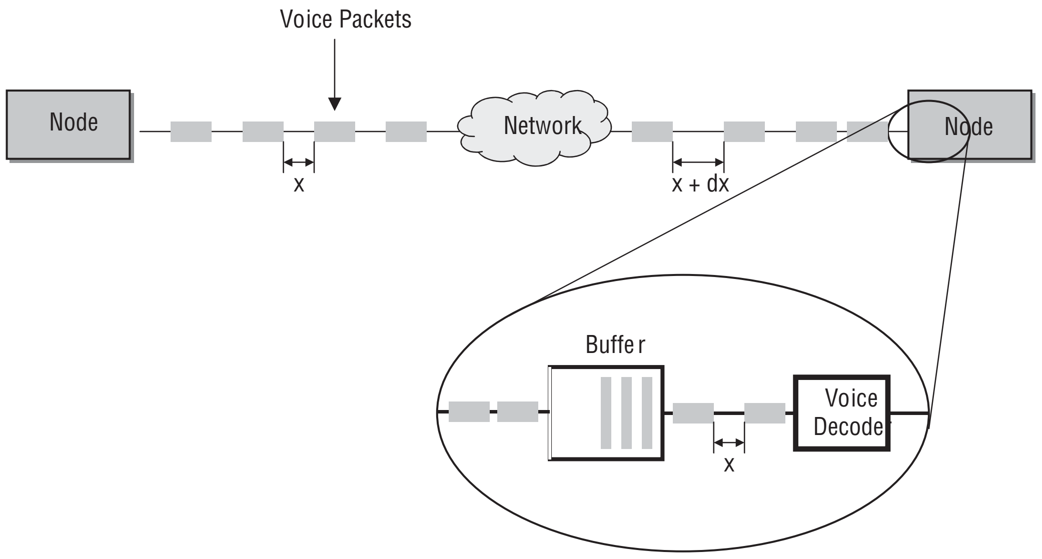


Figure 55. Jitter and dejitter buffer

The dejitter buffer delays incoming packets so it can present them to the decompression algorithm at fixed intervals. It will also fix any out-of-order packets by looking at the sequence number in the RTP packets. Such buffering has the effect of smoothing packet flow, and increasing the resiliency of the codec to packet loss, delayed packets, and other transmission effects. The negative side of dejitter buffering is that it can add significant delay. The dejitter buffer size is configurable and can be optimized for given network conditions.

The operating modes for the dejitter buffer are illustrated in figure 56:

- Adaptive—The adaptive buffer automatically adapts to variations in the network's delay characteristics and in general yields the best results for voice conversations.

IMPORTANT

In the adaptive dejitter buffer there are parameters that can be configured (such as shrink-speed, grow-step, etc.) that should not be changed unless it is necessary to do so. An incorrect configuration can lead to interoperability problems and loss of service. ***Therefore, it is strongly recommended that only experienced users change these parameters.***

- Static—The static buffer is useful for voice conversations if you have specific information about your network's delay characteristics (such as jitter period, etc.), so it should only be used by experienced users.
- Static-data—The static-data mode if you want to create a profile for fax or modem transmission without using the T.38 or fax bypass features described later in this chapter

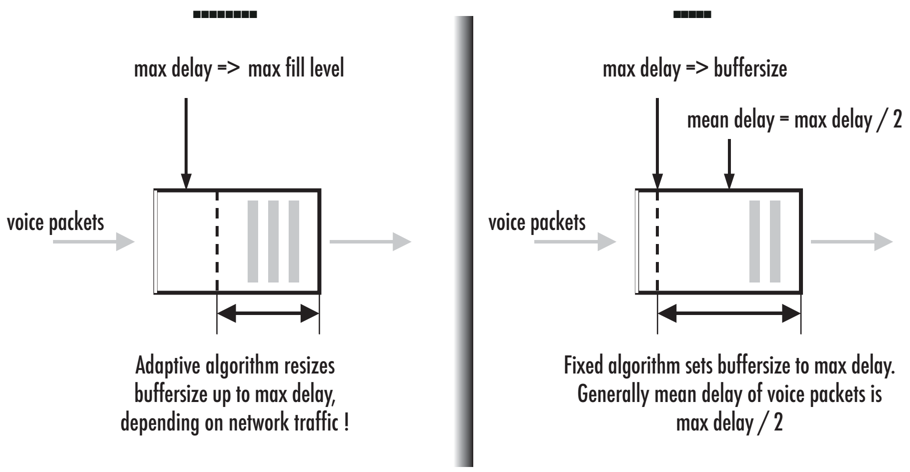

max delay => max fill level
voice packets
Adaptive algorithm resizes buffersize up to max delay, depending on network traffic !
max delay => buffersize
mean delay = max delay / 2
voice packets
Fixed algorithm sets buffersize to max delay. Generally mean delay of voice packets is max delay / 2

Figure 56. Adaptive versus static dejitter buffer

Procedure: Configure the dejitter buffer.
Mode: Profile VoIP

Step	Command	Purpose
1	**device(pf-voip)device#dejitter-mode** *mode*	Specify the dejitter buffer as adaptive, static or static-data.
2	**device(pf-voip)device#dejitter-max-delay** *max-delay*	Specify the maximum delay in milliseconds that the dejitter buffer is allowed to introduce. This setting is valid for all modes.

Enabling/Disabling Filters (advanced)

The voice decoder output is normally filtered through a perceptual post-filter to improve voice quality. Likewise a high pass filter is normally used to cancel noises at the coder input. When the communication channels include several Patton devices in tandem as shown in figure 57, sequential post filtering or high pass filtering can cause degrade signal quality. In this case, the user can choose to disable the post-filter and the high-pass-filter.

Note Filtering only occurs with G.723 and G.729 codecs.

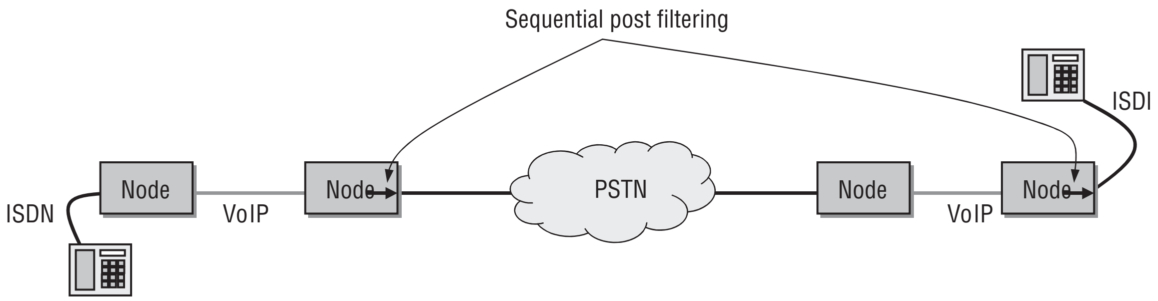


Figure 57. Multiple tandem and sequential post filtering

This procedure describes how to disable post-filtering and high-pass-filtering.

Mode: Profile VoIP

Step	Command	Purpose
1	**device(pf-voip)device#no post-filter**	Disable decoder output filter
2	**device(pf-voip)device#no high-pass-filter**	Disable decoder input high pass filter

Example: Disable filters

The following example shows how to disable the decoder output post-filter and the input high-pass filter.

```
device>enable
device#configure
device(cfg)#profile voip myProfile
device(pf-voip)[myProfi~]#no post-filter
device(pf-voip)[myProfi~]#no high-pass-filter
```

Configuring Fax Transmission

Fax is a protocol for electronically transmitting written material in-band over a voice channel. In public switched telephone networks (PSTN), a fax is handled the same way as a voice conversation. A G3 Fax device transforms (modulates) a scanned page into audible tones that are transmitted in-band. The receiving device converts the tones (demodulates) and reconstructs the page. In IP networks, problems can make it difficult to handle a faxed call in the same way as a voice call:

- If one or more RTP packets that transport the voice (tones) are lost, the receiver can't reconstruct what the sender sent.
- Codecs other than G.711 compress the voice streams. They are optimized for compressing voice and not modulated data. Compressing and decompressing always incurs a loss of data.

Trinity provides two solutions for fax transmission problem, fax bypass, and fax relays:

- Fax bypass—When a fax transmission is detected by the Patton device, it automatically switches to a configured fallback codec that does no or little compression. The dejitter buffer is configured with settings optimized for fax transmission.
- Fax relay—Terminates the fax protocol on the Patton device and sends the reference data over a fax protocol (T.38) to the receiver. Fax relay has a smaller bit-error-rate than bypass.
- Fax failover—When using fax transmission in SIP, you can configure the SIP gateway first to try T.38, but if the remote gateway does not support T.38, it will automatically fall back to a high-rate codec.

Both solutions require changing codecs during an established call, which imposes several requirements on the signaling protocol and the remote gateway. Make sure these requirements are met when configuring a fax transmission mode.

Figure 58 illustrates the difference between Fax relay and Fax bypass.

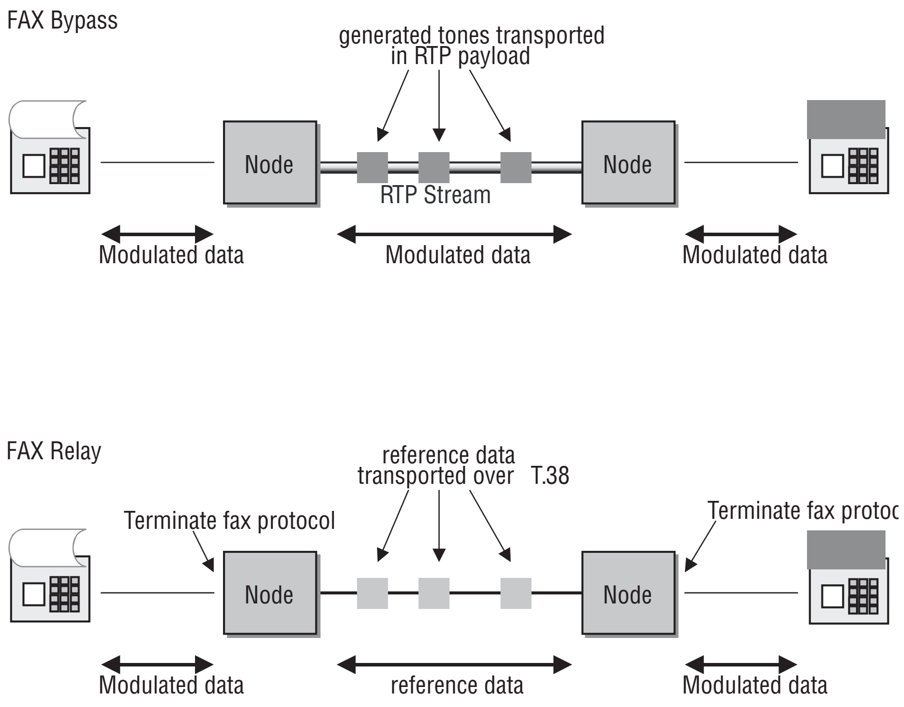


Figure 58. Fax relay and Fax bypass

Fax transmission modes are organized the same way codecs are: there is an ordered list of fax transmission modes; the most preferred fax transmission mode is the first one in the list.

Procedure: Configure fax bypass

Mode: Profile VoIP

Step	Command	Purpose
1	**device(pf-voip)device#fax transmission bypass** (g711alaw64k | g711ulaw64k) [**tx-length** <*ms*>] [**rx-length** <*ms*>]	Configures the fax bypass transmission mode and sets the packetization time (optional). Default packetization time: *10ms*
2 (optional)	**device(pf-voip)device#fax dejitter-max-delay** *buffer-size*	Sets the size of the dejitter buffer during fax transmissions. The operating mode of the dejitter buffer is automatically set to fax optimized static-data mode. Patton recommends that you keep the size for fax transmissions higher than that used for voice, since fax is less sensitive to delay than packet loss. The default value is 200ms which should be nominal for almost any transmission network. In exceptional cases it may be necessary to increase this value (maximum 400ms).

Procedure: Configure fax relay (T.38)

Mode: Profile VoIP

Step	Command	Purpose
1	**device(pf-voip)device# fax transmission relay t38-udp**	Adds fax relay transmission with T.38 protocol over UDP to the list of fax transmission modes.
2 (optional)	**device(pf-voip)device#fax redundancy ls** *low-speed-redundancy* **hs** *high-speed-redundancy*	Packet loss can be avoided by transmitting the fax data packets several times. This can be configured separately for low speed and the high speed traffic. The default for both parameters is 0 (no redundant transmission). Note that values greater than 0 provide more reliable transmissions, but consume additional bandwidth.
3 (optional)	**device(pf-voip)device# fax dejitter-max-delay** *buffer-size*	For proper operation, a dejitter buffer is used on the receiver. The dejitter period can be set to compensate for the jitter imposed by the network. The default value is 200ms which should be nominal for almost any transmission network. Only in exceptional cases it may be necessary to increase this value (maximum 400ms). The dejitter buffer, by default, applies the operation mode 'static-data', i.e. minimizes the packet loss.
4 (optional)	**device(pf-voip)device#fax volume** *volume*	Adjusts the volume of the fax signals re-generated on the receiver side. The volume is in dB, in the range -18.5 ... -3.5 (Default: -9.5dB).
5 (optional)	**device(pf-voip)device# fax max-bit-rate** { 2400 | 4800 | 7200 | 9600 | 12000 | 14400 }	Sets maximum allowed bit-rate for fax relay (Default 14400 Bit/sec).
6 (optional)	**device(pf-voip)device# fax detection** { ced-tone | fax-frames }	Selects the method when fax transmissions are detected: By CED tone or by fax frames (Default: ced-tone). It takes longer to detect Fax frames than CED tones, but the risk of misdetection is minimized.
7 (optional)	**device(pf-voip)device#no fax error-correction**	Disables error correction mode (Default: enabled). If the error correction mode is disabled, the connected fax devices cannot negotiate error correction mode. Connections with error correction mode enabled are more sensitive to packet loss. Disable error correction mode when packet loss is more than 2–3%. **Note** Error correction mode does not cancel IP packet loss.

Step	Command	Purpose
8 (optional)	**device(pf-voip)device#no fax hdlc**	Disables HDLC image transfer (Default: enabled). If HDLC mode is enabled, the Patton device removes bit-stuffing, checks CRCs of fax frames arriving from the PSTN and regenerates the CRCs before sending fax frames towards the PSTN. HDLC can only be enabled together with error correction. Disable HDLC when the fax peer does not support this mode.

T.38 CED Retransmission

Even if the user has configured redundant transmission for low-speed and high-speed packets, the T.30 Indicator messages are not included in this process. If the CED message gets lost, the remote device only receives the CED Tone that is sent in-band. But the transmitted in-band tone may be short due to T.38 switchover or too much distortion, such as by using a low bit-rate voice coder like G.723. In this case it is possible that the remote device never starts the initial T.30 procedure, because it has never received the CED tone. For that reason Trinity sends the CED three times in an interval of 100ms with the same sequence number. With this command, you can disable this feature or set the number of retransmissions to a user defined value.

Mode: profile voip *profile-name*

Step	Command	Purpose
1	**device (pf-voip)device#[no] fax ced-retransmission** *number*	Specifies the number of CED retransmissions. Default: *2*

T.38 No-Signal Retransmission

Some SIP gateways change their port number when switching from audio to T.38. This behavior causes problems if the Patton device is located on the A-Side behind a NAT. Due to T.30 being a unidirectional protocol and the B-Side is normally the initiator of the T.30 handshaking, the Patton device never receives the initial packets of the B-Side because the NAT ports are not yet opened.

To open the NAT ports, Trinity sends T.38 'no-signal' packets when a codec change is detected. By default Patton device sends three such packets. To adjust the number of 'no-signal' packets, use the following configuration command.

Mode: Configure/profile voip

Step	Command	Purpose
1	**device (pf-voip)device#fax nosignal-retransmission [1...5]**	Sets how many times a T.38 'nosignal' is retransmitted. Default: *3*

Fax Bypass Method

This command specifies the method for notifying the remote device that the RTP Stream has switched to a voice-band FAX transmission. This feature is only available on the SIP protocol. If the command option '**v150-vbd**' is selected, a re-invite is sent even if the current voice coder is configured the same as the fax bypass coder. Furthermore the re-invite contains a gpmd-attribute line with the value '**vbd=yes**' in the media description part. It signals the remote device of the new media transmission. If the command option 'default' is selected, the system behavior is the same as before.

For a fax transmission over a VoIP (SIP) network, the Cisco NSE standard uses events defined by RFC2833. These events are used for the setup of the fax transmission starting between the calling- and called-peer. Upon detecting a fax transmission, the called-peer issues NSE Event 192. NSE Event 192 indicates the data stream is via a voice band, and it forces the calling-peer to do two things—deactivate voice activity detection and reconfigure the de-jitter buffer for data reception. The option 'nse' enables this fax transmission standard.

Mode: profile voip

Step	Command	Purpose
1	**device (pf-voip)device#fax bypass-method { default | v150-vbd | nse | x-fax | signaling }**	Specifies the fax bypass signaling method. Default: *default*

Configuring Fax Failover

When using fax transmission in SIP, it is possible configure the SIP gateway to use T.38 and to fall back to a high-rate codec if the remote gateway does not support T.38. This can be configured as follows:

Mode: profile voip <pf-name>

Step	Command	Purpose
1	**device (pf-voip)**[pf-name]**# fax transmission 1 relay t38-udp**	Define T.38 UDP as the first fax transmission method to try
2	**device (pf-voip)**[pf-name]**# fax transmission 2 bypass g711alaw64k**	Define G.711 A-Law as the second fax transmission method to try, if T.38 is not supported by the remote gateway.

Note The first codec must always be T.38, while the second one must be a high-rate codec such as G.711, which supports fax transmission.

Configuring Modem Transmission

Modem transmission is similar to fax transmission, except that modem data is always transported in bypass mode. This means that an ordered list of bypass codecs can be defined for modem transmission. If no modem transmission codec is configured, no action is taken to change the codec when modem is detected.

Procedure: Configure modem bypass

Mode: Profile VoIP

Step	Command	Purpose
1	**device(pf-voip)device#modem transmission bypass (g711alaw64k | g711ulaw64k)** [**tx-length** *<ms>*] [**rx-length** *<ms>*]	Configures the modem bypass transmission mode and sets the packetization time (optional). Default packetization time: *10ms*

Modem Bypass Method

This command specifies the method to be used for signaling the remote device that the RTP Stream has switched to a voice-band Modem transmission. This feature is only available on the SIP protocol. If the command option *v150-vbd* is selected, a re-invite will be sent even if the current voice coder is configured the same as the modem bypass coder. Furthermore the re-invite contains a gpmd-attribute line with the value *vbd=yes;ecan=off* in the media description part. This attribute signals the remote device of the new media transmission. If the command option *default* is selected, the system behavior is the same as before.

Trinity also supports the Cisco NSE standard which uses RFC2833 events for modem transmission over a VoIP (SIP) network. Upon detecting a modem transmission, the *called* peer issues NSE Event 192. NSE Event 192 indicates a Voice Band Data stream that forces the *calling* peer to deactivate Voice Activity Detection and to reconfigure the Dejitter Buffer for data reception. Afterwards it issues the NSE Event 193 to trigger the calling peer to switch off Echo-Cancellation.

Step	Command	Purpose
1	**device (pf-voip)device#modem bypass- method** { default | v150-vbd | nse | x-modem | signaling **}**	Specifies the modem bypass signaling method. Default: *default*

Configuring Packet Side Modem/Fax Answer Tone Detection

This feature allows the detection of Modem/Fax answer tones on the packet (RTP) side. It allows instantaneous switching from audio mode to voice-band-data mode without executing a re-invitation of the session or sending Named Signaling Events (NSE). This feature allows Trinity to be interoperable with third party gateways or servers that are not able to indicate a media switch by sending V.150 VBD attributes in re-invites nor sending Cisco's NSE events. When enabled, the Patton device starts observing the incoming RTP stream for three seconds upon reaching the connected state. During this observation period, a detected answer tone must be stable for 300 milliseconds. CED-Tone detection is only active when the Patton device is the calling party. If the CED-Tone net side detection is enabled, the user can select from the behaviors described below:

- **default**—Switch from audio mode to voice-band-data mode without executing a re-invitation of the session.
- **re-negotiation**—Issue a re-invite for T.38. This behavior is only valid when fax preferred codec is T.38.
- **fallback**—Switch in to bypass (a bypass coder must be configured). If the bypass coder is different from the audio coder then a re-invite is sent. This behavior is only valid when fax preferred codec is T.38.

Mode: profile voip

Step	Command	Purpose
1	**[*device*](pf-voip)device#[no] ced net-side-detection** [re-negotiation | fallback]	Enables/Disables Modem/Fax answer tone detection on the network side.Choose between *default* (no option) or *re-negotiation* or *fallback* scenario.

Disabling Fax/Modem Detection for Voice Calls

To prevent wrong fax/modem detection during a voice call, the media detection feature can be disabled. To allow fax and voice calls on the same Patton device, and because a fax call during setup is not distinguishable from a voice call, media detection is enabled for the specified time period. If after that time no fax or modem is detected, the detection feature will disable to prevent wrong detections that can disturb the call.

Mode: Profile VoIP

Step	Command	Purpose
1	**device(pf-voip)device# [no] media detection-timeout <seconds>**	Configures the time period that the fax/modem detection is enabled. When disabled or set to 0, the fax/modem detection is enabled during the whole call. Default: Disabled.

Media Processing

In some cases, it may be desired to detect fax or modem calls on the Smartnode. However, for SIP to SIP scenarios, this was not possible when the Smartnode was not actively transcoding. If both call-ends used the same codec, the Smartnode would release the DSP resources to optimize performance but disabling the possibility to detect fax or modem (media) during the call. A new command has been added to make this performance optimization configurable, so that fax and modem can be detected even when not transcoding..

Note TRANSCODING license is required for this feature

Mode: Interface SIP

Step	Command	Purpose
1	**device(pf-voip)[<name>]#media-processing auto**	Remove DSP resources if not needed (default).
2	**device(pf-voip)[<name>]#media-processing forced**	Never remove DSP resources, enabling media detection in any SIP-SIP call.

Configuring IP-IP Codec Negotiation

This command is only available on VoIP devices and applies to calls between two IP endpoints (e.g. SIP-SIP). It is in mode "profile voip".

Step	Command	Purpose
1	**device(pf-voip)**[*default*]**#[no] codec negotiation**	Enables/disables codec negotiation.

Disabled "codec negotiation" honors the codec lists from each call leg independently, formed out of the remote and local capabilities. The DSP is inserted into the RTP path to make sure each side can use its codec. The DSP is transcoding between the codecs of the two RTP streams. Enabled "codec negotiation" will keep the DSP out of the picture for IP-IP calls and tries to negotiate a common codec for both call legs.

Configuring the Preferred Codec for Responses

The normal codec negotiation procedure adopts the preference set through the order that the codecs are listed in the incoming SDP offer. For example, the first codec is the most preferred one followed by the common codecs between peers according to the ordered list of incoming codecs. With the *preferred codec* feature, the Patton device can be forced to respond with the preferred codec only in the SDP answer. If the preferred codec is contained in the suggested codecs of the incoming SDP offer, it is going to be the only codec that is supported in the current call independently of its priority order in the suggested codecs list.

- Only a codec that has been configured can be selected as preferred codec for the response.
- A codec that has been selected as the preferred codec for the response cannot be removed in the configured codecs list.
- The codec preference feature is taken into account only if the peer on the other side of the call control does not support the IP-IP codec negotiation.

Step	Command	Purpose
1	**device(pf-voip)#[no] response-preferred-codec** *<codec>*	Configures the codec to be used in responses to all incoming and supporting SIP calls. Default: disabled.

Example: Choosing a preferred codec for the response

```
profile voip voipName
  response-preferred-codec <codec>
```

Examples

Different applications require different VoIP profiles. This section includes a variety of applications and show how the VoIP profile for these applications would be configured.

Home Office in an Enterprise Network

figure 59 is an example of a home office in an enterprise network. The connection bandwidth is 128 kbps and is of very low quality, so the low bit-rate G.723_6k3 codec is used. Likewise, silence suppression is enabled. Because of the low bit-rate codec, DTMF relay is also enabled. As 80 to 100 ms jitter is anticipated, the dejitter buffer is set to *adaptive* with a maximum delay of 100 ms.

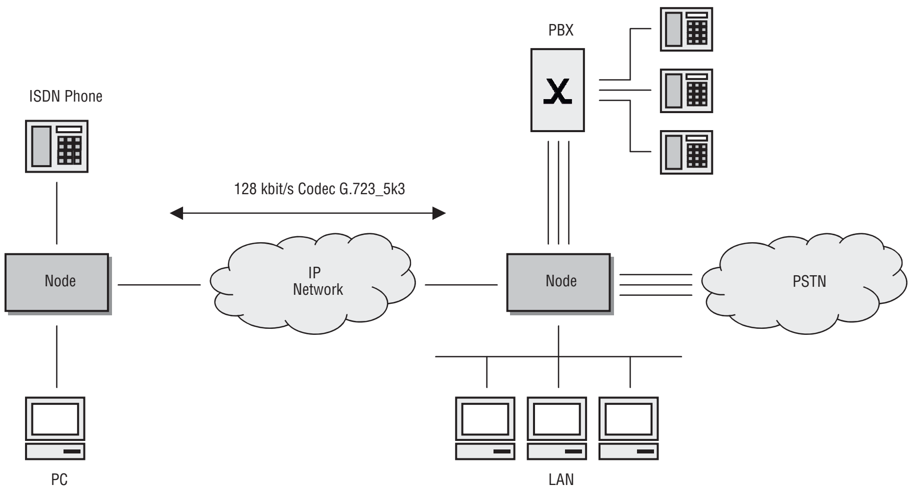


Figure 59. Home office in an enterprise network

First, configure the required CS interfaces (see chapter 41, “CS Interface Configuration” on page 416) and call routing (see chapter 45, “Call Router Configuration” on page 455).

Next, configure the voice over IP settings as needed based on the previous description. First we create the VoIP profile with the needed configurations.

```
device>enable
device#configure
device(cfg)#profile voip Wire128kbit
device(pf-voip)[Wire128~]#no codec g711aLaw64k
device(pf-voip)[Wire128~]#no codec g711uLaw64k
device(pf-voip)[Wire128~]#codec g723-6k3 tx-length 30 rx-length 30 silence-sup-
pression
device(pf-voip)[Wire128~]#dejitter-max-delay 100
device(pf-voip)[Wire128~]]#show profile voip Wire128kbit
VoIP Profile: Wire128kbit
=========================

 Used:                                  by 0 module(s)

 Codecs
 ------

  G.723 6k3:                            rxlen=30;txlen=30;ss

 Fax Transmission
 Modem Transmission

 Dejitter
 --------

  Mode:                                 Adaptive
  Max. Delay:                           100ms
  Max. Packet Loss:                     4/1000
  Shrink Speed:                         1
  Grow Step:                            1
  Grow Attenuation:                     1
  High Pass Filter:                     enabled
  Post Filter:                          enabled

 Fax
 ---

  Detection:                            CED Tone
  T.38 High Speed Redundant Packets:    0
  T.38 Low Speed Redundant Packets:     0
  Max. Bit Rate:                        14400bps
  Volume:                               -9.500dB
  Error Correction:                     enabled
  HDLC:                                 enabled
  Dejitter Max Delay:                   200ms

 Modem
 -----

  Max. Bit Rate:                        14400
```

```
      Volume:                                    -9.500dB
      HDLC:                                      enabled

     DTMF
     ----

      Relay:                                     enabled
      Mute Encoder:                              enabled

     RTP
     ---

      Payload Type NTE:                          101
```

Description:

3. Create VoIP profile and give it a name. All settings have default values

4., 5. Remove the default codecs G.711alaw and G.711uLaw

6. Add codec g723-6k3 with silence-suppression enabled

7. Allow the dejitter buffer to compensate 100 milliseconds of network jitter.

8. Show the configured profile.

Home Office with Fax

Preconditions are those used in section "Home Office in an Enterprise Network" on page 384: low bandwidth and high jitter. In this example, bandwidth is 256 kbps, what enables us to use the G.729 codec. But since the fax protocol must also be supported, the configuration is extended:

```
1 device>enable
2 device#configure
3 device(cfg)#profile voip g729_FaxRelay
4 device(pf-voip)[g729_Fa~]#no codec g711aLaw64k
5 device(pf-voip)[g729_Fa~]#no codec g711uLaw64k
6 device(pf-voip)[g729_Fa~]#codec g729 tx-length 20 rx-length 20 silence-suppres-
sion
7 device(pf-voip)[g729_Fa~]#dejitter-max-delay 100
8 device(pf-voip)[g729_Fa~]#fax transmission relay t38-udp
9 device(pf-voip)[g729_Fa~]#fax max-bit-rate 9600
10 device(pf-voip)[g729_Fa]]#show profile voip g729_FaxRelay
VoIP Profile: g729_FaxRelay
===========================

 Used:                                       by 0 module(s)

 Codecs
 ------

  G.729A:                                    rxlen=20;txlen=20;ss
  T.38 UDP

 Fax Transmission
 ----------------
```

```
         T.38 UDP

        Modem Transmission

        Dejitter
        --------

         Mode:                                 Adaptive
         Max. Delay:                           100ms
         Max. Packet Loss:                     4/1000
         Shrink Speed:                         1
         Grow Step:                            1
         Grow Attenuation:                     1
         High Pass Filter:                     enabled
         Post Filter:                          enabled

        Fax
        ---

         Detection:                            CED Tone
         T.38 High Speed Redundant Packets:    0
         T.38 Low Speed Redundant Packets:     0
         Max. Bit Rate:                        9600bps
         Volume:                               -9.500dB
         Error Correction:                     enabled
         HDLC:                                 enabled
         Dejitter Max Delay:                   200ms

        Modem
        -----

         Max. Bit Rate:                        14400
         Volume:                               -9.500dB
         HDLC:                                 enabled

        DTMF
        ----

         Relay:                                enabled
         Mute Encoder:                         enabled

        RTP
        ---

         Payload Type NTE:                     101
```

Description:

3. Create VoIP profile and give it a name. All settings have default values

4., 5. Remove the default codecs G.711alaw and G.711uLaw

6. Add codec g729 with silence-suppression enabled

7. Allow the dejitter buffer to compensate 100 milliseconds of network jitter.

8. Enable fax relay over T.38 protocol

9. Limit the maximum bit rate the fax devices can communicate with each other to 9600 kbps

10. Show the configured profile.

Soft Phone Client Gateway

A soft phone client can only use G.711uLaw or G.723 codes, neither of which can use silence suppression, DTMF relay, or fax.

```
1 device>enable
2 device#configure
3 device(cfg)#profile voip softPhone
4 device(pf-voip)[softPho~]#no codec g711aLaw64k
5 device(pf-voip)[softPho~]#codec g723-6k3 tx-length 30 rx-length 30 no-silence-
suppression
6 device(pf-voip)[softPho~]#no dtmf-relay
7 device(pf-voip)[softPho]]#show profile voip softPhone
VoIP Profile: softPhone
=======================

 Used:                                   by 0 module(s)

 Codecs
 ------

  G.711 u-law:                           rxlen=20;txlen=20
  G.723 6k3:                             rxlen=30;txlen=30

 Fax Transmission
 Modem Transmission

 Dejitter
 --------

  Mode:                                  Adaptive
  Max. Delay:                            60ms
  Max. Packet Loss:                      4/1000
  Shrink Speed:                          1
  Grow Step:                             1
  Grow Attenuation:                      1
  High Pass Filter:                      enabled
  Post Filter:                           enabled

 Fax
 ---

  Detection:                             CED Tone
  T.38 High Speed Redundant Packets:     0
  T.38 Low Speed Redundant Packets:      0
  Max. Bit Rate:                         14400bps
  Volume:                                -9.500dB
  Error Correction:                      enabled
  HDLC:                                  enabled
  Dejitter Max Delay:                    200ms

 Modem
```

```
-----

 Max. Bit Rate:                        14400
 Volume:                               -9.500dB
 HDLC:                                 enabled

DTMF
----

 Relay:                                disabled
 Mute Encoder:                         disabled

RTP
---

 Payload Type NTE:                     101
```

Chapter 39 PSTN Profile Configuration

Chapter contents

Introduction

This chapter gives an overview of PSTN profiles, and describes how they are used and the tasks involved in PSTN profile configuration.

A PSTN profile is a container for all datapath-related settings on PSTN connections. It can be assigned to PSTN interfaces in context CS. If no profile is specified in a particular interface, the profile *default* is used. The settings apply to all calls crossing the interface. figure 60 illustrates the relationship between PSTN profiles and CS interfaces. The following components are configurable:

- Echo canceller
- Output gain
- Input gain

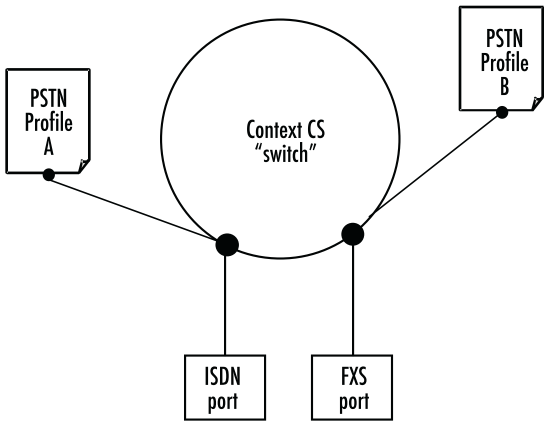


Figure 60. PSTN profile association

PSTN Profile Configuration Task List

The following tasks describe components that can be configured through the PSTN profile.

- Creating a PSTN profile
- Configuring the echo canceler (see page 392)
- Configuring output gain (see page 392)

If a PSTN profile is modified, the saved modification is applied to all open calls and is valid for all future calls on the interface using this PSTN profile.

Creating a PSTN Profile

Each PSTN profile has a name that can be any arbitrary string of not more than 25 characters. When you create the PSTN profile, the PSTN profile configuration mode appears so you can configure PSTN components.

Note The PSTN profile named *default* always exists in the system. It is used by all interface components if there is no other PSTN profile available. If PSTN parameters are the same throughout all interfaces, you can simply change the profile *default* instead of creating a new profile.

Procedure: Create a PSTN Profile and enter the PSTN profile configuration mode.

Mode: Configure

Step	Command	Purpose
1	***device*(cfg)#profile pstn <*name* | *default*>**	Create a PSTN profile with name *name* and enter PSTN profile configuration mode. The newly created profile contains default values for all parameters. If a profile with name *name* already exists, only the PSTN profile configuration mode is entered.
2	***device*(pf-pstn)device#...**	Configuration steps as described in the chapters below

Configuring the Echo Canceller

Echoes are reflections of the transmitted signal that result from impedance mismatches in the hybrid (bi-directional 2-wire to 4-wire conversion) device, causing an echo on the wire. Echo cancellation provides near-end echo compensation for this effect as shown in figure 61.

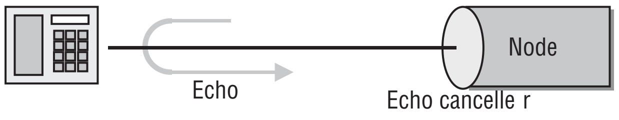


Figure 61. Echo Cancellation

Procedure: Disable echo cancellation.

Mode: Profile PSTN

Step	Command	Purpose
1	***device*(pf-pstn)device#no echo-canceller**	Disable echo canceller (Default: Enabled)

Configuring Output Gain

The output gain determines the voice output volume gain towards PSTN ports as shown in figure 62.

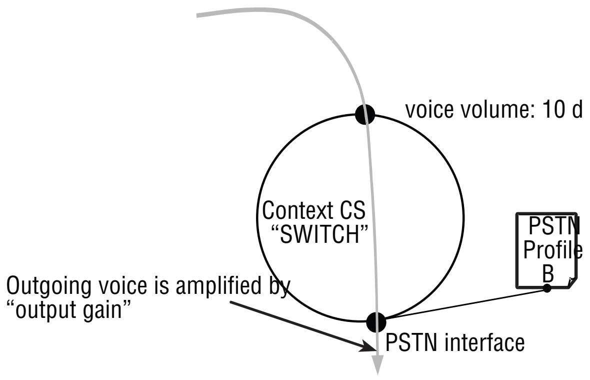


Figure 62. Applying output gain

Procedure: Configure voice output gain.

Mode: Profile PSTN

Step	Command	Purpose
1	***device*(pf-pstn)device#output-gain** *gain*	Set the output gain to value in dB (Def. 0dB)

Configuring Input Gain

The input gain determines the voice output volume gain from PSTN ports as shown in figure 63.

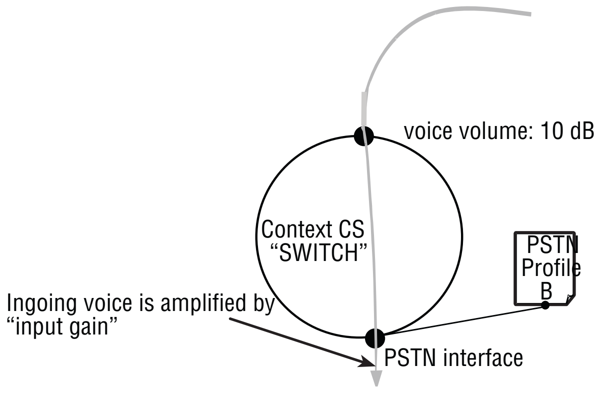


Figure 63. Applying input gain

Procedure: Configure voice input gain.

Mode: Profile PSTN

Step	Command	Purpose
1	***device*(pf-pstn)device#input-gain** *gain*	Set the input gain to value in dB (Def. 0dB)

Chapter 40 CS Context Overview

Chapter contents

Introduction

This chapter gives an overview of the circuit-switching (CS) context and associated components and describes the tasks involved in its configuration. It describes the steps needed to configure voice connectivity and refers to other chapters where a configuration topic is explained in more detail. Before reviewing the content in this chapter, read the configuration concepts as described in Chapter 2, "Configuration Concepts" on page 47.

The CS context is a high level conceptual entity that is responsible for all aspects of circuit signaling, switching, and emulation. Besides the context switch itself, the CS entity consists of the following (indicated by the shaded area enclosed by a dashed line in figure 64):

- The CS interfaces
- ISDN
- Tone-set profiles
- Context SIP gateways
- VoIP profiles

The CS Context is enabled by default.

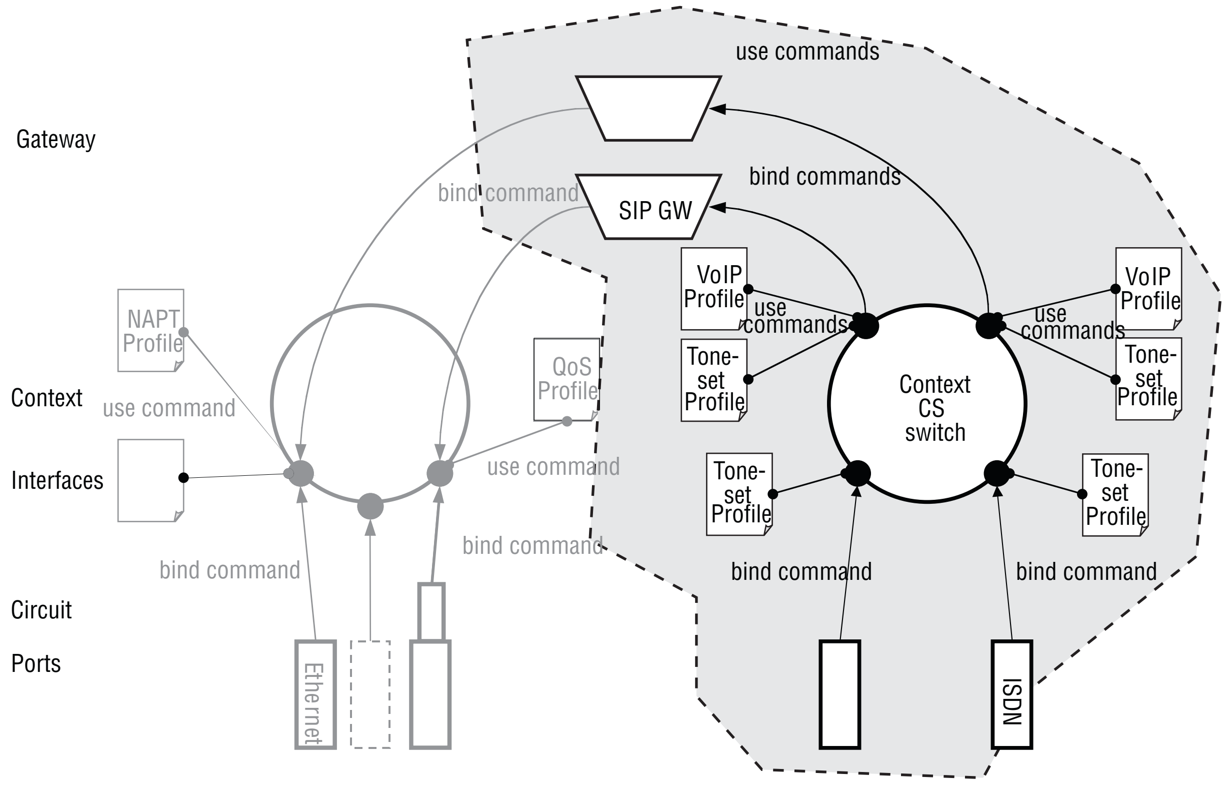


Figure 64. CS context configuration components

The CS context and its associated components route and establish voice calls. For example, the signaling for dial-up circuits is routed and the corresponding voice call circuits are switched between PSTN interfaces and via VoIP interfaces to the Context SIP gateways and the IP context (see section "Configuring Call Routing" on page 400 for more details).

CS Context Configuration Task List

Information needed for CS entity configuration is distributed among several configuration tasks, depending on its logical content. For example, information pertaining to call routing is described in section "Configuring Call Routing" on page 400. These configuration tasks can be described in other chapters; thus, to configure call routing you have to refer to Chapter 41, "CS Interface Configuration" on page 416 and Chapter 45, "Call Router Configuration" on page 455.

This chapter shows you the relationship between the CS configuration components. We recommend that you perform the CS context configuration in the sequence described below. Many of the parameters have default values that do not need to be changed, which means that you do have to modify all of the described configuration tasks. In such cases it is stated in the text that you can skip the optional configuration tasks.

1. Planning the CS configuration
2. Configuring general CS settings
3. Configuring call routing
4. Creating and configuring CS Interfaces
5. Configuring dial tones (advanced)
6. Configuring voice over IP settings (advanced)
7. Configuring ISDN ports
8. Configuring a SIP VoIP connection
9. Activating the CS context configuration
10. CS Context Chapter Overview Example

Planning the CS Configuration

There are many policies and factors that can influence the CS context configuration. It depends on what your application is and how your network is configured. Several factors to consider for planning your CS configuration are listed below:

- Application/network scenario
- Peripheral devices, such as PBX or remote VoIP gateway.
- VoIP protocol
- Number and type of physical telephony ports available
- Call routing

Figure 65 Shows a typical application with a remote office in an enterprise network. This example focuses on the Patton device in the remote office. There is an ISDN phone, a personal computer, a connection to the public ISDN network, and a connection to the IP backbone. The VoIP protocol used is SIP with a codec G.711. A call can be routed to the IP backbone and the public ISDN network depending on its prefix and number length.

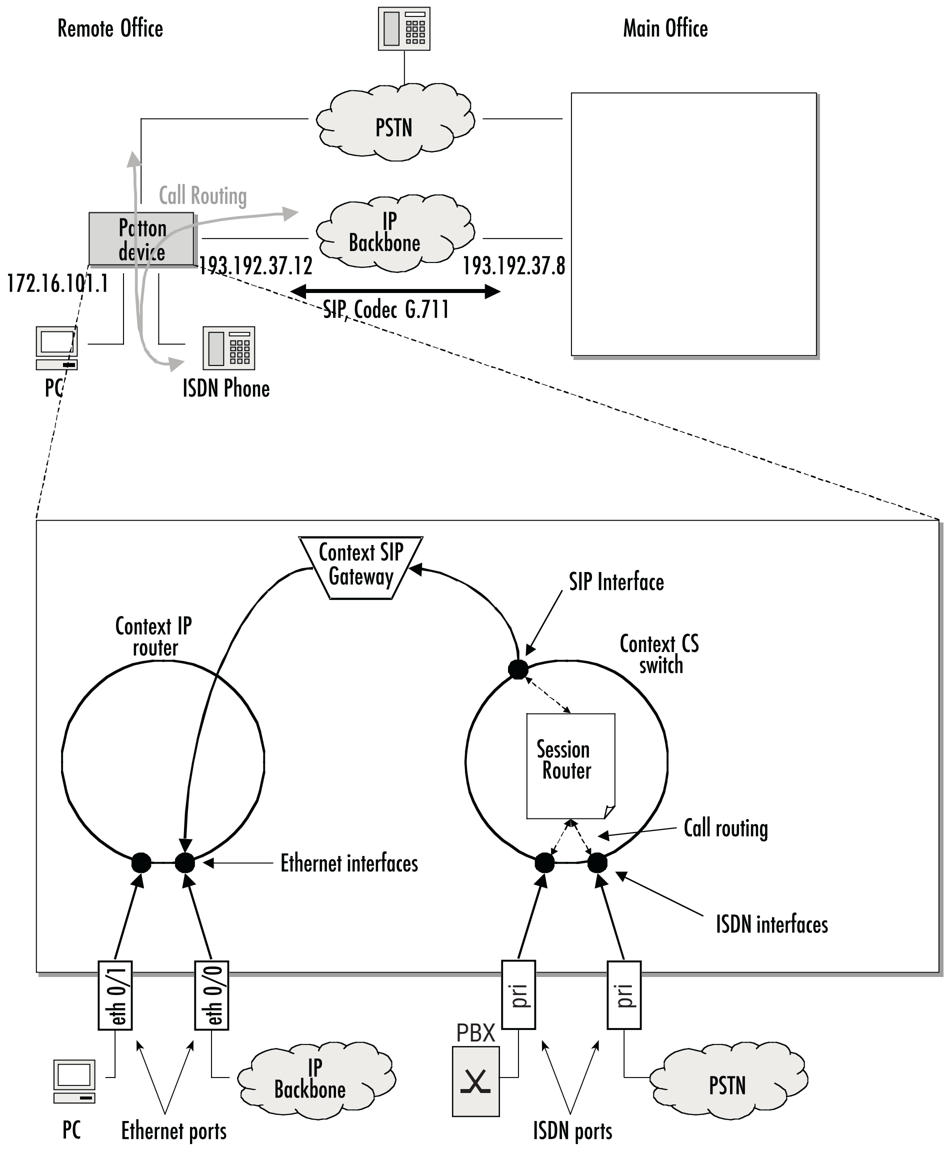


Figure 65. Remote office in an Enterprise network

An application like that shown in figure 65 would require the following CS configuration:

- Since the remote office is connected to the public switched telephone network, the clock-source comes from the corresponding ISDN port. (Described in section "Configuring General CS Settings" on page 398).

Note Be careful when choosing where you get your clock source, if the clock used for packaging the ISDN voice frames is not synchronized with the remote

ISDN clock, bit errors may result (such synchronization problems would probably cause a fax transmission to fail).

- Two PRI ports will be needed, the first port for the ISDN PRI PBX and the second for the public ISDN network (see section "Configuring ISDN Ports" on page 402).
- Two ISDN interfaces will be needed, each bound to a BRI port (see section "Configuring Call Routing" on page 400)
- The call router routing tables, and the SIP and ISDN interfaces will have to be configure to support call routing (see section "Configuring Call Routing" on page 400).

  Calls are routed from an ISDN PBX with a number in the range of 1xx–5xx to the main office with a fall-back to the PSTN. All other calls are routed from the ISDN PBX to the PSTN and from the PSTN or main office to the ISDN phone behind the PBX.
- The Context SIP gateway must be configured to use the G.711 codec (see section, "Configuring Voice Over IP Parameters" on page 401)
- Two Ethernet ports and their corresponding IP interfaces will be needed.

You must not start to configure the CS context and its components until you have finished planning your voice environment. The following chapters explain how to convert the planned voice environment into the Trinity CS configuration. The IP configuration is not a topic in this example. For more information on IP configuration refer to Chapter 21, "IP Context Overview" on page 229.

Configuring General CS Settings

There are several parameters that cannot be collected into one specific configuration task, because they are independent of the rest of the CS context configuration and apply mostly to an interface card or even to the entire Patton device.

Configuring the clock source

A reference clock is needed for packaging the ISDN voice frames. The reference clock can be generated internally or obtained from an external source (e.g. public ISDN). Patton devices have a feature called 'Clock Source Hunting'. This feature allows the user to configure an index-based list of clock sources. The source with the lowest index has the highest priority and vice versa. On Patton devices populated with several PRI or BRI ports where more than one port is working in 'clock slave' mode, all these ports can be entered in the clock source list. The algorithm behind this feature always takes the first synchronized 'slave' port in the list as the current clock source. If the links of all the ports in the list are down or not synchronized, the system is falling back to its internal clock source. It is also possible to enter all PRI or BRI ports of the device into the list, independent of their clock mode. The Clock Source Hunting algorithm ignores all entered ports that are not working in 'slave' mode.

Mode: System

Step	Command	Purpose
1	***device*(sys)#clock-source<index>[1..10]**	Enter position of the clock source
2	***device*(sys)#clock-source after <index>[1..10]**	Insert an entry after position ‘index’ (1-10)
3	***device*(sys)#clock-source before<index>[1..10]**	Adds a BRI port as clock source
4	***device*(sys)#clock-source bri <slot>[0..]**	Adds a E1T1 port as a clock source
5	***device*(sys)#clock-source e1t1 <slot>[0..]**	Move entry at ‘index’ number of ‘positions up
6	***device*(sys)#clock-source index up positions**	Move entry at ‘index’ number of ‘positions’ up
7	***device*(sys)#clock-source index down positions**	Move entry at ‘index’ number of ‘positions’ down

Debugging the clock source

To control the system behavior at runtime, there exists a debug clock-source command with the options ‘detail’ and ‘full-detail’.

Mode: Operator execution

Step	Command	Purpose
1	***device*#[no] debug clock-source (detail 0 | full-detail)**	Troubleshoots the system clock monitor

Mode: Operator execution

Step	Command	Purpose
1	***device*#show clock-source**	Print system clock information

```
device#show clock-source

Current clock source
====================

 internal

Registered clock sources
========================

 Name                        Capable Sync
 e1t1 0 0 0
 internal                    X       X
```

Selecting PCM law compression

The PCM law-select specifies the voice characteristic compression curve. Two values are possible: *a-Law* (used in Europe) and *μ-Law* (used in the USA).

Procedure: To set the general CS parameters

Step	Command	Purpose
1	**[node](cfg)#system pcm law-select { alaw | ulaw }**	Generates the reference clock internally or specifies a specific port to receive the reference clock. Default: alaw

Configuring Call Routing

Calls through a Patton device can be routed according to a set of routing criteria. The entity that manages call routing is called the *call router*. Calls are routed from one CS interface to another. The call router determines the destination interface for every incoming call. It supports complex call routing and call property manipulation (e.g. number manipulation) functions. See Chapter 45, "Call Router Configuration" on page 455.

Call routing occurs in the context CS element between several CS interfaces. Accordingly, a CS context and two or more CS interfaces must be created.

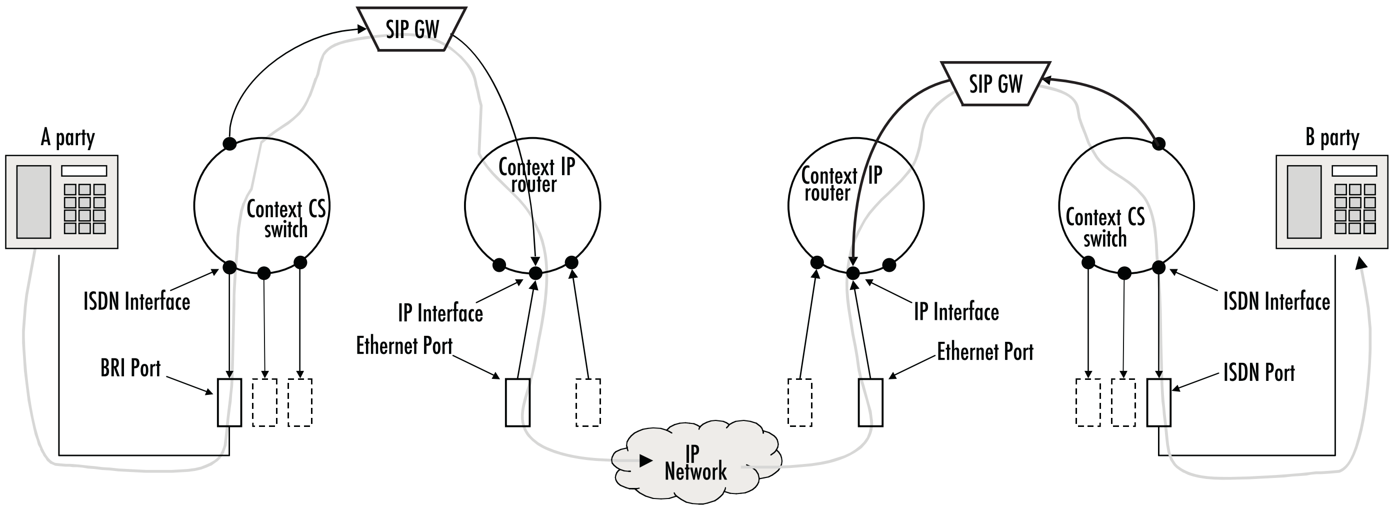


Figure 66. Direct call routing from one Patton device to another

Figure 66 shows a call set up from the A-party on the left to the B-party on the right. The call is routed from the phone on the left-hand side over the ISDN interface directly to an SIP interface. Once it has passed the IP context and the IP network, the other device—from the SIP interface to the ISDN interface and then over the BRI port to the B-party phone—routes the call.

Note Because call routing occurs only in the CS context, in future figures the context IP is omitted. For configuring call routing you have to create the CS interfaces and the call router tables as described in the chapters below. For simple call routing directly from one interface to another you can even omit router tables.

Creating and Configuring CS Interfaces

Multiple instances of CS contexts are supported. The name of the default instance is *switch*. The name and number of CS interfaces depends on your own configuration. The interfaces on the CS context represent logical connections to other equipment or networks. CS interfaces are used as source and destination in the call router. VoIP CS interfaces are bound to a gateway. Telephony ports are bound to respective interfaces.

Interface names can be any arbitrary string with a maximum of 25 characters. For ease of identification, the interface type can be a part of the name. For examples and information on how to create CS interfaces, refer to Chapter 41, "CS Interface Configuration" on page 416.

Specify Call Routing

As mentioned previously, for basic call routing you can omit creating call router tables. Trinity offers two levels of call routing:

- Basic interface routing
- Advanced call routing

Basic interface routing allows you to forward all incoming calls on a CS interface directly to a destination CS interface. The call router allows you to route calls to all available CS interfaces, based on a call property such as calling number, destination number and ISDN bearer capability and many more.

We recommend that you first carefully consider what interfaces and call router tables are required to achieve your goals on a sheet of paper, then start creating and configuring CS interfaces, and setting up call router tables.

To configure basic interface routing refer to Chapter 41, "CS Interface Configuration" on page 416. Other topics that belong to call routing are also explained in this chapter.

To configure advanced call routing in relation to the call router tables refer to Chapter 45, "Call Router Configuration" on page 455. In this chapter, the differences between basic interface routing and advanced call routing are described in more detail.

Configuring Dial Tones

Trinity supports country-specific, configurable, in-band dial tones that are generated for specific events, For example, alerting, and dialing or busy signals. The tones are configured in tone-set profiles that are used from a specific CS interface.

If no tone-set profile is specified, a default tone-set profile is used. In most cases, the default profile can be used, so you do not need to perform this configuration task.

Configuring Voice Over IP Parameters

In Trinity, there are many configurable parameters that can affect a voice over IP connection.

The *voice over IP* (VoIP) parameters are configured in the VoIP profile. A VoIP profile is used by a SIP interface. All calls going through that interface (see figure 66 on page 400) use the settings in the VoIP profile. The following parameters are configured in the VoIP profile:

- Codecs
- Fax transmission
- Filters
- DTMF relay
- Echo canceller
- Silence compression
- Voice volume
- Dejitter buffer

Refer to Chapter 38, "VoIP Profile Configuration" on page 365 to configure general VoIP parameters. Some settings can adversely affect the voice quality perceived by the user and the bandwidth requirements of VoIP connections, so be sure you understand the meaning of the commands before changing any settings. Most of the default values of these parameters are adequate, so you generally do not need to perform these configuration tasks.

If no VoIP profile is specified for use on an interface, a default VoIP profile is used. In most cases, the default profile can be used so you just need to change the default VoIP profile.

Configuring ISDN Ports

BRI and E1/T1 ports represent physical ports on the Patton device. The configuration of the ISDN ports depends on the port type (BRI, E1 or T1), and on the connected voice device. To configure the ISDN ports, refer to Chapter 55, "ISDN Interface Configuration" on page 631.

Configuring a SIP VoIP Connection

To configure a SIP connection, you have to specify the voice codec selection and the call signaling method for the VoIP profile.

When configuring the voice codec for a SIP connection, you must specify the VoIP profile that shall be used on a SIP interface. The VoIP profile contains an ordered list of codecs that shall be used for codec negotiation for all calls routed through this interface. During a call setup, the first codec that is specified in the VoIP profile is taken. For information on how to configure the codecs, refer to Chapter 38, "VoIP Profile Configuration" on page 365.

You can configure the Context SIP gateway to a registrar with multiple URIs. Optionally, you can configure the Context SIP gateway to send all requests to an outbound proxy or redirect server.

You have several options on how to build a destination URI (To-URI) of an outgoing SIP call. You can use the called party number in conjunction with the specified domain name or you can set a specific URI by the call router, based on other call properties. For examples and information on how to configure the Context SIP gateway, refer to Chapter 52, "Context SIP Gateway Overview" on page 601. For more on SIP interface configuration, refer to Chapter 48, "SIP Interface Configuration" on page 538.

Activating CS Context Configuration

After configuring the CS context and its components, the configuration must be activated. This includes binding the physical ports to the virtual CS interfaces and enabling the gateways, ports, and the CS context.

In order to become functional, each interface must be bound from one port where it receives incoming calls and also forwards outgoing calls. Unlike ISDN interfaces, VoIP interfaces must be bound to a Context SIP gateway.

Note The difference between VoIP and PSTN interfaces is that VoIP interfaces are bound to a Context SIP gateway while PSTN ports are bound to a CS interface. After binding, the BRI, E1, or T2 ports must be enabled to become active.

To bind an ISDN port to an ISDN interface, refer to Chapter 55, "ISDN Interface Configuration" on page 631. Likewise, the Context SIP gateway must be enabled. Additionally, the Context SIP gateway must be bound to a specific IP interface. For more information, refer to Chapter 52, "Context SIP Gateway Overview" on page 601.

In order to become active, the CS context must be enabled. When recovering from the shutdown status, the CS context and call router configuration is checked and possible errors are indicated. The call router debug monitor can be enabled to show the loading of the CS context and call router configuration. Trinity offers a number of possibilities to monitor and debug the CS context and call router configurations. For example, the call router debug monitor enables you to follow the sequence of tables and functions examined by the call router for each call setup. Refer to Chapter 59, "Debug and Monitoring" on page 661 for an introduction to the configuration debugging possibilities in Trinity.

Note You can modify the configuration at runtime; changes will be active after 3 seconds. It is not necessary to shut down the CS context before making configuration changes, a newly created or changed configuration is automatically loaded as long as the context CS is not shut down. All active calls are not affected by this reload.

There are several possibilities to show the actual CS context configuration. For more information on the show command, refer to the respective configuration Chapters or to the Chapter 41, "CS Interface Configuration" on page 416" and Chapter 45, "Call Router Configuration" on page 455.

Procedure: Show the CS context configuration, enable the call router debug monitor and activate them in the CS context

Mode: Context CS

Step	Command	Purpose
1	***device***(ctx-cs)[SWITCH]**#show call-router config detail** <*level*>	Shows the CS context configuration. *Level* could be 1..5. Level 1 shows less, level 5 shows all information.
2	***device*** (ctx-cs)[SWITCH]**#debug cr detail** <*level*>	Enables the call-router debug monitor. *Level* could be 1..5. Level 1 only logs errors, level 5 shows all relevant information to track calls through routing tables.
3	***device*** (ctx-cs)[SWITCH]**#no shutdown**	Enables the CS context, checks the interface and call router configuration
4	***device***(ctx-cs)[SWITCH]**#show call-router status detail** <*level*>	Shows the actual state of the call router. This includes all configured tables as they were read-in from the configuration.

Example: Enable CS Context

The following example shows how to enable the call router debug monitor and how to enable the CS context. It also shows the output from the call router debug monitor.

```
device(cfg)#show call-router config detail 5
Table switch/TAB-ISDN-SERVICE:
Key               Value             Function       Dest-Type      Dest-Name
itc               -
-------------------------------------------------------------------------------
unrestricted-digital -              -              dest-interface IF-LOCAL-BA
default           -                 -              dest-table     TAB-DEST-A

Table switch/TAB-DEST-A:
Key               Value             Function       Dest-Type      Dest-Name
called-e164       -
-------------------------------------------------------------------------------
0                 -                 MAP-CAC-ORANGE dest-interface IF-LOCAL-BA
00                -                 MAP-CLI-MELON  dest-interface IF-NODE-C
07[4-6]           -                 MAP-CAC-APPLE  dest-interface IF-LOCAL-BA
0336652...        -                 -              dest-interface IF-NODE-B
default           -                 -              dest-interface IF-LOCAL-BA

Table switch/CAC-APPLE:
Key               Value             Function       Dest-Type      Dest-Name
called-e164       called-e164
-------------------------------------------------------------------------------
(.%)              1055\1            -              -              -

...

device(cfg)#debug cr
device(cfg)#context cs
device(ctx-cs)[SWITCH]#no shutdown
```

```
02:14:30  CR    > Updating tables in 3 seconds...
02:14:33  CR    > [SWITCH] Reloading tables now
02:14:33  CR    > [SWITCH] Flushing all tables
02:14:33  CR    > [SWITCH] Loading table 'TAB-ISDN-SERVICE'
02:14:33  CR    > [SWITCH] Loading table 'TAB-DEST-A'
02:14:33  CR    > [SWITCH] Loading table 'CAC-APPLE'
02:14:33  CR    > [SWITCH] Loading table 'CAC-ORANGE'
02:14:33  CR    > [SWITCH] Loading table 'CLI-MELON'
02:14:33  CR    > [SWITCH] Loading table 'MAP-CAC-APPLE'
02:14:33  CR    > [SWITCH] Loading table 'MAP-CAC-ORANGE'
02:14:33  CR    > [SWITCH] Loading table 'MAP-CLI-MELON'
02:14:33  CR    > [SWITCH] Loading table 'IF-LOCAL-BA-precall-service'
02:14:33  CR    > [SWITCH] Loading table 'IF-PBX-A-precall-service'
02:14:33  CR    > [SWITCH] Loading table 'IF-device-B-precall-service'
02:14:33  CR    > [SWITCH] Loading table 'IF-device-C-precall-service'
device(ctx-cs)[SWITCH]#
```

Example: Configure the Patton device in an Enterprise Network

Situation: Figure 67 shows an enterprise network with a Patton device configured with a BRI port. A PBX, a LAN, the PSTN, and the company network are connected. The VoIP protocol used is SIP. The voice codec used is G.723, so the DTMF relay is enabled. Because no special dial tones have to be specified, the default tone-set profile is used.

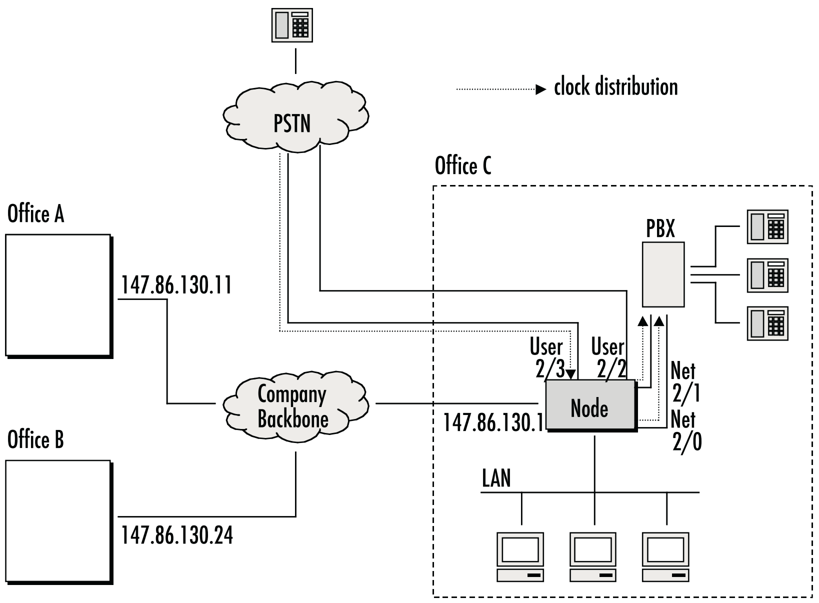


Figure 67. Patton device in an Enterprise network

Call routing is specified as follows:

- Calls from *office C* with number *1xx* to *office A* with a fallback to PSTN
- Calls from *office C* with number *2xx* to *office B* with a fallback to PSTN
- All other calls from *office C* to PSTN

- Calls from *office A* or *B* with number *5xx* to *office C*
- All other calls from *office A* or *B* to the PSTN (local breakout)

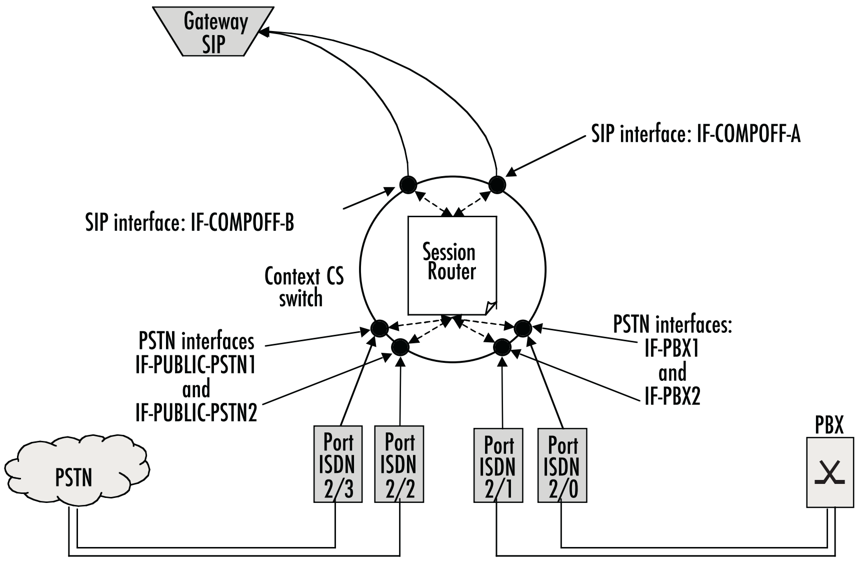

Gateway SIP
SIP interface: IF-COMPOFF-A
SIP interface: IF-COMPOFF-B
Session Router
Context CS switch
PSTN interfaces IF-PUBLIC-PSTN1 and IF-PUBLIC-PSTN2
PSTN interfaces: IF-PBX1 and IF-PBX2
PBX
PSTN
Port ISDN 2/3
Port ISDN 2/2
Port ISDN 2/1
Port ISDN 2/0

Figure 68. CS Configuration

Planning the CS Context

Based on the criteria used in the previous example, the following configuration information applies (see Figure 68):

- It is very important to specify from where to get the clock source for the packaging of the ISDN voice frames. In the example we are connected to the PSTN network and get the clock source from the ISDN over the ISDN port 2/3.
- We need four BRI ports, two for the PSTN and another two for the PBX. (Refer to section "Configuring ISDN Ports" on page 402).
- Furthermore we need four ISDN interfaces. Then we have to bind each BRI port to one of the ISDN interfaces. A hunt group that summarizes two ISDN interfaces is configured later during call router configuration.
- We need a call router routing table to route the calls depending on the called party number. (Refer to section "Configuring Call Routing" on page 400).
- We further need two hunt groups, one that hunts calls to the two BRI interfaces to the PSTN and one for the two BRI interfaces to the PBX.
- Then we need two other hunt group that tries to make a call over a VoIP and if this fails, falls back to the PSTN.
- We enable DTMF relay and specify codec G.723. (Refer to section "Configuring Voice Over IP Parameters" on page 401).

Configuring General CS Settings

First we set clock-source to ISDN port 2/3.

```
device>enable
device#configure
device(cfg)#system
device(sys)#clock-source 2 3
device(sys)#exit
device(cfg)#
```

Configuring Call Routing

Next we create the ISDN interfaces and configure call routing. Each interface is configured to route all incoming calls to the routing table TAB-CALLED-NUMBER. This table is part of the call router and configured below:

```
device(cfg)#context cs
device(ctx-cs)[switch]#interface isdn IF-PBX1
device(if-pstn)[IF-PBX1]#route call dest-table TAB-CALLED-NUMBER
device(if-pstn)[IF-PBX1]#exit

device(ctx-cs)[switch]#interface isdn IF-PBX2
device(if-pstn)[IF-PBX2]#route call dest-table TAB-CALLED-NUMBER
device(if-pstn)[IF-PBX2]#exit

device(ctx-cs)[switch]#interface isdn IF-PUBLIC-PSTN1
device(if-pstn)[IF-PUBL~]#route call dest-table TAB-CALLED-NUMBER
device(if-pstn)[IF-PUBL~]#exit

device(ctx-cs)[switch]#interface isdn IF-PUBLIC-PSTN2
device(if-pstn)[IF-PUBL~]#route call dest-table TAB-CALLED-NUMBER
device(if-pstn)[IF-PUBL~]#exit
device(ctx-cs)[switch]#
```

In addition, we create the two SIP interfaces and configure call routing, as well as the IP address of the remote SIP terminal, which is the IP address of the device in office A or office B, respectively.

```
device(ctx-cs)[switch]#interface sip IF-COMPOFF-A
device(if-sip)[IF-COMP~]#route call dest-table TAB-CALLED-NUMBER
device(if-sip)[IF-COMP~]#remoteip 146.86.130.11
device(if-sip)[IF-COMP~]#bind context sip-gateway SIP_GATEWAY
device(if-sip)[IF-COMP~]#exit

device(ctx-cs)[switch]#interface sip IF-COMPOFF-B
device(if-sip)[IF-COMP~]#route dest-table calledNumberRouting
device(if-sip)[IF-COMP~]#remoteip 146.86.130.24
device(if-sip)[IF-COMP~]#bind context sip-gateway SIP_GATEWAY
device(if-sip)[IF-COMP~]#exit
device(ctx-cs)[switch]#
```

Finally, we configure the call router. Here we create a routing table that examines the called party number of a call and routes numbers starting with a 1 and containing at least 3 digits to the hunt group that tries to reach company office A over VoIP and falls back to the PSTN. We route numbers starting with 2 and containing at least 3 digits to the hunt group that tries to reach company office B over VoIP and falls back to the PSTN. Calls with a prefix of 5 and at least 3 digits are routed to the hunt group that selects a free BRI to the PBX and all other calls are routed to the hunt group that selects a free BRI to the PSTN:

```
device(ctx-cs)[switch]#routing-table called-e164 TAB-CALLED-NUMBER
device(rt-tab)[TAB-CAL~]#route 1.. dest-service HUNT-COMPOFF-A
device(rt-tab)[TAB-CAL~]#route 2.. dest-service HUNT-COMPOFF-B
device(rt-tab)[TAB-CAL~]#route 5.. dest-service HUNT-PBX
device(rt-tab)[TAB-CAL~]#route default dest-service HUNT-PUBLIC-PSTN
device(rt-tab)[TAB-CAL~]#show call-router config
Table switch/TAB-CALLED-NUMBER:
Key              Value            Function       Dest-Type      Dest-Name
called-e164      -
-------------------------------------------------------------------------------
1..              -                -              dest-service   HUNT-COMPOFF-A
2..              -                -              dest-service   HUNT-COMPOFF-B
5..              -                -              dest-service   HUNT-PBX
default          -                -              dest-service   HUNT-PUBLIC-PSTN
device(rt-tab)[TAB-CAL~]#exit
device(ctx-cs)[switch]#
```

The hunt group HUNT-COMPOFF-A tries to reach the company office A routing the call directly to the SIP interface IF-COMPOFF-A. When this call fails (e.g. because the data network is broken), we route the call to the PSTN hunt group. Likewise, hunt group HUNT-COMPOFF-B works, but tries to route the call to the SIP interface IF-COMPOFF-B first.

```
device(ctx-cs)[switch]#service hunt-group HUNT-COMPOFF-A
device(rt-tab)[HUNT-CO~]#no cyclic
device(rt-tab)[HUNT-CO~]#timeout 5
device(rt-tab)[HUNT-CO~]#route call 1 dest-interface IF-COMPOFF-A
device(rt-tab)[HUNT-CO~]#route call 2 dest-service HUNT-PUBLIC-PSTN
device(rt-tab)[HUNT-CO~]#exit
device(ctx-cs)[switch]#service hunt-group HUNT-COMPOFF-B
device(rt-tab)[HUNT-CO~]#no cyclic
device(rt-tab)[HUNT-CO~]#timeout 5
device(rt-tab)[HUNT-CO~]#route call 1 dest-interface IF-COMPOFF-B
device(rt-tab)[HUNT-CO~]#route call 2 dest-service HUNT-PUBLIC-PSTN
device(rt-tab)[HUNT-CO~]#exit
device(ctx-cs)[switch]#
```

The hunt group HUNT-PBX routes the call either to the interface IF-PBX1 or IF-PBX2, depending on which interface there is a free B channel. Likewise the hunt group HUNT-PUBLIC-PSTN works on the PSTN interfaces.

```
device(ctx-cs)[switch]#service hunt-group HUNT-PBX
device(rt-tab)[HUNT-PB~]#cyclic
device(rt-tab)[HUNT-PB~]#route call 1 dest-interface IF-PBX1
device(rt-tab)[HUNT-PB~]#route call 2 dest-interface IF-PBX2
device(rt-tab)[HUNT-PB~]#exit
device(ctx-cs)[switch]#service hunt-group HUNT-PUBLIC-PSTN
device(rt-tab)[HUNT-PU~]#cyclic
device(rt-tab)[HUNT-PU~]#route call 1 dest-interface IF-PUBLIC-PSTN1
device(rt-tab)[HUNT-PU~]#route call 2 dest-interface IF-PUBLIC-PSTN2
device(rt-tab)[HUNT-PU~]#exit
device(ctx-cs)[switch]#exit
device(cfg)#
```

Configuring VoIP Settings

Because we need G.723 as codec we enable DTMF relay:

```
device(cfg)#profile voip SIP-VOIP-PROFILE
device(pf-voip)[sip-VO~]#codec 1 g723-6k3
device(pf-voip)[sip-VO~]#dtmf-relay
device(pf-voip)[sip-VO~]#exit
device(cfg)#
```

We want to use this profile on our SIP interfaces:

```
device(cfg)#context cs
device(ctx-cs)[switch]#interface sip IF-COMPOFF-A
device(if-sip)[IF-COMP~]#use profile voip SIP-VOIP-PROFILE
device(if-sip)[IF-COMP~]#exit
device(ctx-cs)[switch]#interface sip IF-COMPOFF-B
device(if-sip)[IF-COMP~]#use profile voip SIP-VOIP-PROFILE
device(if-sip)[IF-COMP~]#exit
device(cfg)#
```

Configuring BRI Ports

Next step is to configure the BRI ports and to bind the ports to the ISDN interfaces. We configure the layer 2 (Q.921) to use point-to-point mode and layer 3 (Q.931) for user or net operation mode:

```
device(cfg)#port bri 2 0
device(prt-bri)[2/0]#q921
device(q921)[2/0]#protocol pp
device(q921)[2/0]#q931
device(q931)[2/0]#uni-side net
device(q931)[2/0]#encapsulation cc-isdn
device(q931)[2/0]#bind interface IF-PBX1
device(q931)[2/0]#exit
device(q921)[2/0]#exit
device(prt-bri)[2/0]#no shutdown
device(cfg)#port bri 2 1
device(prt-bri)[2/1]#q921
device(q921)[2/1]#protocol pp
device(q921)[2/1]#q931
device(q931)[2/1]#uni-side net
device(q931)[2/1]#encapsulation cc-isdn
device(q931)[2/1]#bind interface IF-PBX1
device(q931)[2/1]#exit
device(q921)[2/1]#exit
device(prt-bri)[2/1]#no shutdown
device(cfg)#port bri 2 2
device(prt-bri)[2/2]#q921
device(q921)[2/2]#protocol pp
device(q921)[2/2]#q931
device(q931)[2/2]#uni-side user
device(q931)[2/2]#encapsulation cc-isdn
device(q931)[2/2]#bind interface IF-PBX1
device(q931)[2/2]#exit
device(q921)[2/2]#exit
device(prt-bri)[2/2]#no shutdown
device(cfg)#port bri 2 1
device(prt-bri)[2/3]#q921
device(q921)[2/3]#q931
device(q921)[2/3]#protocol pp
device(q931)[2/3]#uni-side user
device(q931)[2/3]#encapsulation cc-isdn
device(q931)[2/3]#bind interface IF-PBX1
device(q931)[2/3]#exit
device(q921)[2/3]#exit
device(prt-bri)[2/3]#no shutdown
```

Configuring an SIP VoIP Connection

Next we configure call signaling:

```
device(cfg)#gateway sip sip
device(gw-sip)[sip]#no ras
device(gw-sip)[sip]#faststart
device(gw-sip)[sip]#bind interface eth0
device(gw-sip)[sip]#exit
device(cfg)#
```

Activating the CS Context Configuration

Prior to activating our configuration we use two **show** commands to display part of our configuration:

```
device(cfg)#show call-router config detail 5
Table switch/IF-PBX1-precall-service:
Key               Value             Function       Dest-Type      Dest-Name
-                 -
--------------------------------------------------------------------------------
-                 -                 -              dest-table     TAB-CALLED-NUMBER

Table switch/IF-PBX2-precall-service:
Key               Value             Function       Dest-Type      Dest-Name
-                 -
--------------------------------------------------------------------------------
-                 -                 -              dest-table     TAB-CALLED-NUMBER

Table switch/IF-PUBLIC-PSTN1-precall-service:
Key               Value             Function       Dest-Type      Dest-Name
-                 -
--------------------------------------------------------------------------------
-                 -                 -              dest-table     TAB-CALLED-NUMBER

Table switch/IF-PUBLIC-PSTN2-precall-service:
Key               Value             Function       Dest-Type      Dest-Name
-                 -
--------------------------------------------------------------------------------
-                 -                 -              dest-table     TAB-CALLED-NUMBER

Table switch/IF-COMPOFF-A-precall-service:
Key               Value             Function       Dest-Type      Dest-Name
-                 -
--------------------------------------------------------------------------------
-                 -                 -              dest-table     TAB-CALLED-NUMBER

Table switch/IF-COMPOFF-B-precall-service:
```

```
Key               Value             Function       Dest-Type      Dest-Name
-                 -
-------------------------------------------------------------------------------
-                 -                 -              dest-table     TAB-CALLED-NUMBER

Table switch/TAB-CALLED-NUMBER:
Key               Value             Function       Dest-Type      Dest-Name
called-e164       -
-------------------------------------------------------------------------------
1..               -                 -              dest-service   HUNT-COMPOFF-A
2..               -                 -              dest-service   HUNT-COMPOFF-B
5..               -                 -              dest-service   HUNT-PBX
default           -                 -              dest-service   HUNT-PUBLIC-PSTN
device(cfg)#


device(gw-sip)[gw_name]#exit
device(cfg)#debug cr full-detail
device(cfg)#context cs
device(ctx-cs)[switch]#no shutdown
02:30:26  CR    > Updating tables in 3 seconds...
02:30:28  CR    > [switch] Reloading tables now
02:30:28  CR    > [switch] Flushing all tables
02:30:28  CR    > [switch] Loading table 'IF-PBX1-precall-service'
02:30:28  CR    > [switch] Loading table 'IF-PBX2-precall-service'
02:30:28  CR    > [switch] Loading table 'IF-PUBLIC-PSTN1-precall-service'
02:30:28  CR    > [switch] Loading table 'IF-PUBLIC-PSTN2-precall-service'
02:30:28  CR    > [switch] Loading table 'IF-COMPOFF-A-precall-service'
02:30:28  CR    > [switch] Loading table 'IF-COMPOFF-B-precall-service'
02:30:28  CR    > [switch] Loading table 'TAB-CALLED-NUMBER'
device(ctx-cs)[switch]#
```

Showing the Running Configuration

The configuration script for our application looks as follows:

```
  cli version 3.00

system
  clock-source 2 3

profile voip SIP-VOIP-PROFILE
  codec 1 g723-6k3 rx-length 30 tx-length 30
  codec 2 g711alaw64k rx-length 20 tx-length 20
  codec 3 g711ulaw64k rx-length 20 tx-length 20

context ip router

interface eth0
  ipaddress 147.86.130.1 255.255.225.0
  mtu 1500

interface eth1
  ipaddress 10.0.0.1 255.255.225.0
  mtu 1500
```

```
context cs switch

routing-table called-e164 TAB-CALLED-NUMBER
  route 1.. dest-service HUNT-COMPOFF-A
  route 2.. dest-service HUNT-COMPOFF-B
  route 5.. dest-service HUNT-PBX
  route default dest-service HUNT-PUBLIC-PSTN

interface sip IF-COMPOFF-A
  bind context sip-gateway SIP_GATEWAY
  route call dest-table TAB-CALLED-NUMBER
  remote 146.86.130.11 5060
  privacy
  use profile voip SIP-VOIP-PROFILE

interface sip IF-COMPOFF-A
  bind context sip-gateway SIP_GATEWAY
  route call dest-table TAB-CALLED-NUMBER
  remote 146.86.130.24 5060
  privacy
  use profile voip SIP-VOIP-PROFILE

interface isdn IF-PBX1
  route call dest-table TAB-CALLED-NUMBER

interface isdn IF-PBX2
  route call dest-table TAB-CALLED-NUMBER

interface isdn IF-PUBLIC-PSTN1
  route call dest-table TAB-CALLED-NUMBER

interface isdn IF-PUBLIC-PSTN2
  route call dest-table TAB-CALLED-NUMBER

service hunt-group HUNT-COMPOFF-A
  timeout 5
  drop-cause normal-unspecified
  drop-cause no-circuit-channel-available
  drop-cause network-out-of-order
  drop-cause temporary-failure
  drop-cause switching-equipment-congestion
  drop-cause access-info-discarded
  drop-cause circuit-channel-not-available
  drop-cause resources-unavailable
  route call 1 dest-interface IF-COMPOFF-A
  route call 2 dest-service HUNT-PUBLIC-PSTN

service hunt-group HUNT-COMPOFF-B
  timeout 5
  drop-cause normal-unspecified
  drop-cause no-circuit-channel-available
  drop-cause network-out-of-order
  drop-cause temporary-failure
  drop-cause switching-equipment-congestion
  drop-cause access-info-discarded
  drop-cause circuit-channel-not-available
  drop-cause resources-unavailable
  route call 1 dest-interface IF-COMPOFF-B
  route call 2 dest-service HUNT-PUBLIC-PSTN

service hunt-group HUNT-PBX
  cyclic
  drop-cause normal-unspecified
  drop-cause no-circuit-channel-available
  drop-cause network-out-of-order
  drop-cause temporary-failure
  drop-cause switching-equipment-congestion
  drop-cause access-info-discarded
  drop-cause circuit-channel-not-available
  drop-cause resources-unavailable
  route call 1 dest-interface IF-PBX1
  route call 2 dest-interface IF-PBX2

service hunt-group HUNT-PUBLIC-PSTN
  cyclic
  drop-cause normal-unspecified
  drop-cause no-circuit-channel-available
  drop-cause network-out-of-order
  drop-cause temporary-failure
  drop-cause switching-equipment-congestion
  drop-cause access-info-discarded
  drop-cause circuit-channel-not-available
  drop-cause resources-unavailable
  route call 1 dest-interface IF-PUBLIC-PSTN1
  route call 2 dest-interface IF-PUBLIC-PSTN2

context cs switch
  no shutdown

context sip-gateway SIP_GATEWAY
  interface WAN_SIP
  bind interface eth0 context router port 5060
  context sip-gateway SIP_GATEWAY
  no shutdown

port ethernet 0 0
  medium 10 half
  encapsulation ip
  bind interface eth0 router
  no shutdown

port ethernet 0 1
  medium 10 half
  encapsulation ip
  bind interface eth1 router
  shutdown

port bri 2 0
  clock auto
  encapsulation q921
```

```
q921
  protocol pp
  uni-side auto
  encapsulation q931

q931
  protocol dss1
  uni-side net
  encapsulation cc-isdn
  bind interface IF-PBX1

port bri 2 0
  no shutdown

port bri 2 1
  clock auto
  encapsulation q921

q921
  protocol pp
  uni-side auto
  encapsulation q931

q931
  protocol dss1
  uni-side net
  encapsulation cc-isdn
  bind interface IF-PBX2

port bri 2 1
  no shutdown

port bri 2 2
  clock auto
  encapsulation q921

q921
  protocol pp
  uni-side auto
  encapsulation q931

q931
  protocol dss1
  uni-side user
  encapsulation cc-isdn
  bind interface IF-PUBLIC-PSTN1

port bri 2 2
  no shutdown

port bri 2 3
  clock auto
  encapsulation q921
```

```
q921
  protocol pp
  uni-side auto
  encapsulation q931

q931
  protocol dss1
  uni-side user
  encapsulation cc-isdn
  bind interface IF-PUBLIC-PSTN2

port bri 2 3
  no shutdown
```

Chapter 41 CS Interface Configuration

Chapter contents

Introduction

This chapter provides an overview of interfaces in the CS context and describes the tasks involved in their configuration. Within the CS context, an interface is a logical entity providing call signaling for incoming and outgoing calls to and from telephony ports and voice over IP gateways. It represents logical connections to other equipment or networks. CS interfaces are used as source and destination in the call router and are bound to physical ports or logical gateways.

Interface names can be any arbitrary string with a maximum of 25 characters. For ease of identification, the interface type can be a part of the name. Figure 69 illustrates the function of the CS interfaces. The types of CS interfaces:

- PSTN interfaces telephony. Binding is done from a port to an interface.
- VoIP interface provide voice over IP settings in addition to the general CS interface parameters. These interfaces must be explicitly bound to an existing VoIP gateway.

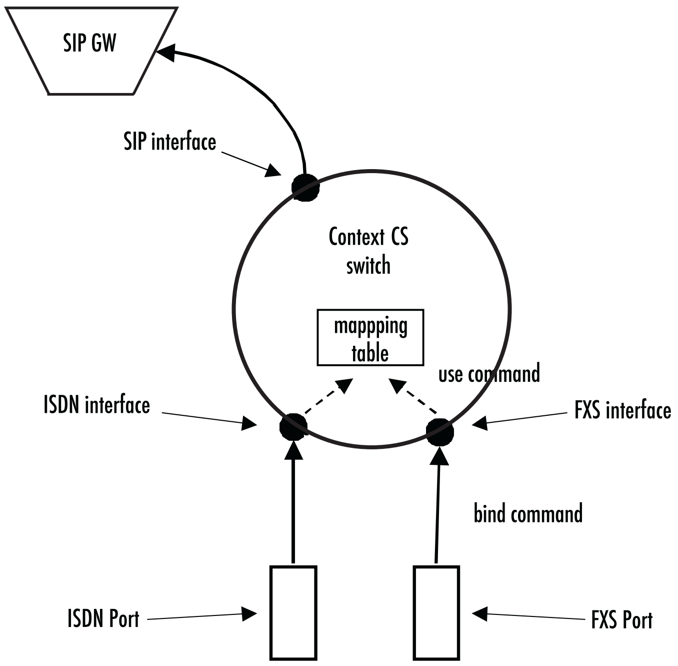


Figure 69. CS interfaces on the CS context

Interfaces can use mapping tables and precall service tables to manipulate call properties before the call is being offered to the call router.

CS Interface Configuration Task List

Several parameters depend upon the interface type. If it is not specifically stated otherwise, the configuration task is valid for all interfaces. This is not described in this chapter, but in Chapter 42, "Tone Configuration" on page 424 and chapter 38, "VoIP Profile Configuration" on page 365. To create and configure CS interfaces you have to perform the configuration tasks listed below.

- Creating and configuring CS interfaces

- Configuring call routing
- Configuring the interface mapping tables (optional)
- Configuring the precall service tables (optional)
- Configuring interface type specific parameters

Creating and Configuring CS Interfaces

To configure CS interfaces, you must first enter the CS context mode where you can create and configure your required interface through the CS interface configuration mode. Each interface has a name that can be any arbitrary string of not more than 25 characters. Use a name describing the purpose of the interface, as shown in the examples or—for ease of identification—the interface type can be used as part of the name. Already-defined CS interfaces can be displayed or deleted as described in the following table.

Procedure: Create and configure CS interfaces.

Mode: Configure

Step	Command	Purpose
1	***node*****(cfg)#context cs**	Enter the CS Context Configuration Mode.
2	***node*****(ctx-cs)**[*switch*]**#interface** *if-type if-name*	Enter the CS Interface Configuration Mode & select the CS interface with type *if-type* and name *if-name* for configuration. Valid interface types are sip, isdn, fxs and fxo.
3	***node***(if-*type*)[*if-name*]#…	Perform the configuration tasks to configure the CS interface.
4	***node*****(ctx-cs)**[*switch*]**#show call-control provider**	Display the configuration of the current CS interface.
5	***node*****(if-*type*)**[*if-name*]**#exit**	Go back to the CS Context Configuration Mode
6	***node*****(ctx-cs)**[*switch*]**#show call-control provider** OR ***node*****(ctx-cs)**[*switch*]**#show call-control status**	Display already defined CS interfaces. **Note:** The show call-control provider command can also be used to display the configuration details of a provider either by specifying its name as a parameter or by being inside its configuration mode.
7	***node*****(ctx-cs)**[*switch*]**#no interface** *if-type if-name*	Delete an existing interface.

Examples: Create CS interfaces and delete another

The following example shows how to create and configure an interface, how to display it, and how to delete another.

```
node>enable
node#configure
node(cfg)#context cs
node(ctx-cs)[switch]#interface isdn IF-PBX1
node(if-pstn)[IF-PBX1]#route call dest-interface TAB-CALLED-NUMBER
node(if-pstn)[IF-PBX1]#show call-control provider
Provider: IF-PBX1
=================
 Binding:                                   (none)
 Protocol:                                  (unknown)
 DTMF Dialing:                              disabled
 Tone-Set Profile:                          (none)
 PSTN Profile:                              default
 Routing Destination:                       router (IF-PBX1-precall-service)
 Active Endpoints:                          0
 Suspended endpoints:                       0
node(if-pstn)[IF-PBX1]#exit
node(ctx-cs)[switch]#show call-control provider
Call Control: switch
====================

 Providers
 ---------

  local
  router
  sn43
  IF-PBX1
  IF-PBX2
  IF-PUBLIC-PSTN1
  IF-PUBLIC-PSTN2
  IF-COMPOFF-A
  HUNT-COMPOFF-A
  HUNT-PBX
  HUNT-PUBLIC-PSTN
node(ctx-cs)[switch]#no interface isdn IF-PBX1
node(ctx-cs)[switch]#
```

Configuring Call Routing

Trinity offers two levels of call routing: basic interface routing and advanced call routing. Basic interface routing allows you to forward all incoming calls on a CS interface to a destination CS interface.

Advanced call routing allows you to route calls to all available CS interfaces, based on a criteria such as calling number, destination number, ISDN bearer capability, or other call properties. Using mapping tables, you can modify call properties like the calling or called party number, URI, etc. Furthermore, you can collect numbers using the digit-collection feature of called party number routing tables. Call services like hunt or distribution groups can be used to distribute calls to multiple destination interfaces.

In the environment of the CS interfaces, it is necessary to specify whether the call will be routed directly to another CS interface (basic interface routing) or to a first lookup table from the call router (advanced call routing).

In this chapter, only the configuration task on a CS interface is described. For configuration of the call routing tables, mapping tables and call services refer to Chapter 45, "Call Router Configuration" on page 455, which also describes the difference between the two levels of call routing in more detail.

Procedure: To configure basic interface routing

Mode: Context CS

Step	Command	Purpose
1	***node*(ctx-cs)**[switch]**#interface** *if-type if-name*	Enters CS Interface Configuration Mode and configure interface *if-type* with name *if-name*
2	***node*(if-*type*)**[*if-name*]**#route call dest-interface** *if-name* OR ***node*(if-*type*)**[*if-name*]**#route call dest-table** *table-name* OR ***node*(if-*type*)**[*if-name*]**#route call dest-service** *service-name*	Specifies a destination interface for incoming calls (basic interface routing) or a destination table or call service (advanced call routing)
3	***node***(*if- type*)[*if-name*]**#exit**	Returns to CS context configuration mode

Example: Configure call routing

The following example shows how to configure basic interface routing.

```
node>enable
node#configure
node(cfg)#context cs
node(ctx-cs)[switch]#interface isdn IF-PBX1
node(if-pstn)[IF-PBX1]#route call dest-interface IF-SIP-0
node(if-pstn)[IF-PBX1]#exit
node(ctx-cs)[switch]#
```

Configuring the Interface Mapping Tables

Call router mapping tables are normally used by the call router to manipulate call properties during the call setup phase, i.e. when a call arrives on a CS interface and is routed to another interface through routing and mapping tables. This imposes a limitation to call property manipulation: When a call property like a party's number is changed during a call, the call is not routed through the call router again and thus, the mapping tables are not processed for the new number. Call property manipulation, e.g. removing a prefix from a number, cannot be done for the new number.

Consider, for example, an ISDN call, which may send a connected party number in the Connect message. This connected party number has the same meaning as the original called party number, but may differ from it. Another example of a call property that changes during a call is a SIP call transfer. A SIP call may be transferred to another user agent having a different URI than the called one. This new URI as well as the derived E.164 number cannot be manipulated using the call router before presenting it to the other party.

To circumvent this limitation, you can use mapping tables directly on an interface. In that case the mapping tables can be thought as input or output filters, which manipulate call properties at any stage of a call.

As with the SIP transfer example, differentiating *called* from *calling* party properties does not make sense for these manipulations, because the calling as well as the called party can be transferred in a SIP call. Therefore, mapping tables that are used on an interface manipulate both *called* and *calling* party properties at the same time! Although, using a calling-e164 mapping table at the interface will change the *calling* and *called* properties of the call.

You can chose different mapping tables for filtering parameters in each direction, input and output. While an input mapping table is applied to all properties that are received by the port or gateway that is bound to the interface before sending them to the peer interface in the CS context, an output mapping table is applied to all properties before sending them to the bound port or gateway.

Refer to the Chapter 45, "Call Router Configuration" on page 455 for more information about how to create and configure mapping tables.

Procedure: To use mapping tables to filter properties on an CS interface

Mode: Context CS

Step	Command	Purpose
1	***node*(ctx-cs)**[switch]**#interface** *if-type if-name*	Enters CS Interface Configuration Mode and configure interface *if-type* with name *if-name*
2	***node***(if-*type*)[*if-name*]**#use mapping-table in** *table-name* AND/OR ***node***(if-*type*)[*if-name*]**#use mapping-table out** *table-name*	Specifies an input and/or an output mapping table that shall be applied to all call properties in the specified direction.

Example: Use interface mapping tables for dialing plan conversion

The following example shows how to configure a dialing plan conversion on an interface. In this case, you can plan your call-routing tables to deal only with international numbers while converting private numbers on the CS interface that interfaces the private network.

```
node(ctx-cs)[switch]#mapping-table e164 to e164 PRIV-TO-GLOB
node(map-tab)[PRIV-TO~]#map (..) to 00419988825\1
node(map-tab)[PRIV-TO~]#exit
node(ctx-cs)[switch]#mapping-table e164 to e164 GLOB-TO-PRIV
node(map-tab)[GLOB-TO~]#map 00419988825(..) to \1
node(map-tab)[GLOB-TO~]#exit
node(ctx-cs)[switch]#interface isdn IF-PHONES
node(if-isdn)[IF-PHON~]#route call dest-table TAB-CALLED-NUMBER
node(if-isdn)[IF-PHON~]#use mapping-table in PRIV-TO-GLOB
node(if-isdn)[IF-PHON~]#use mapping-table out GLOB-TO-PRIV
node(if-isdn)[IF-PHON~]#exit
node(ctx-cs)[switch]#
```

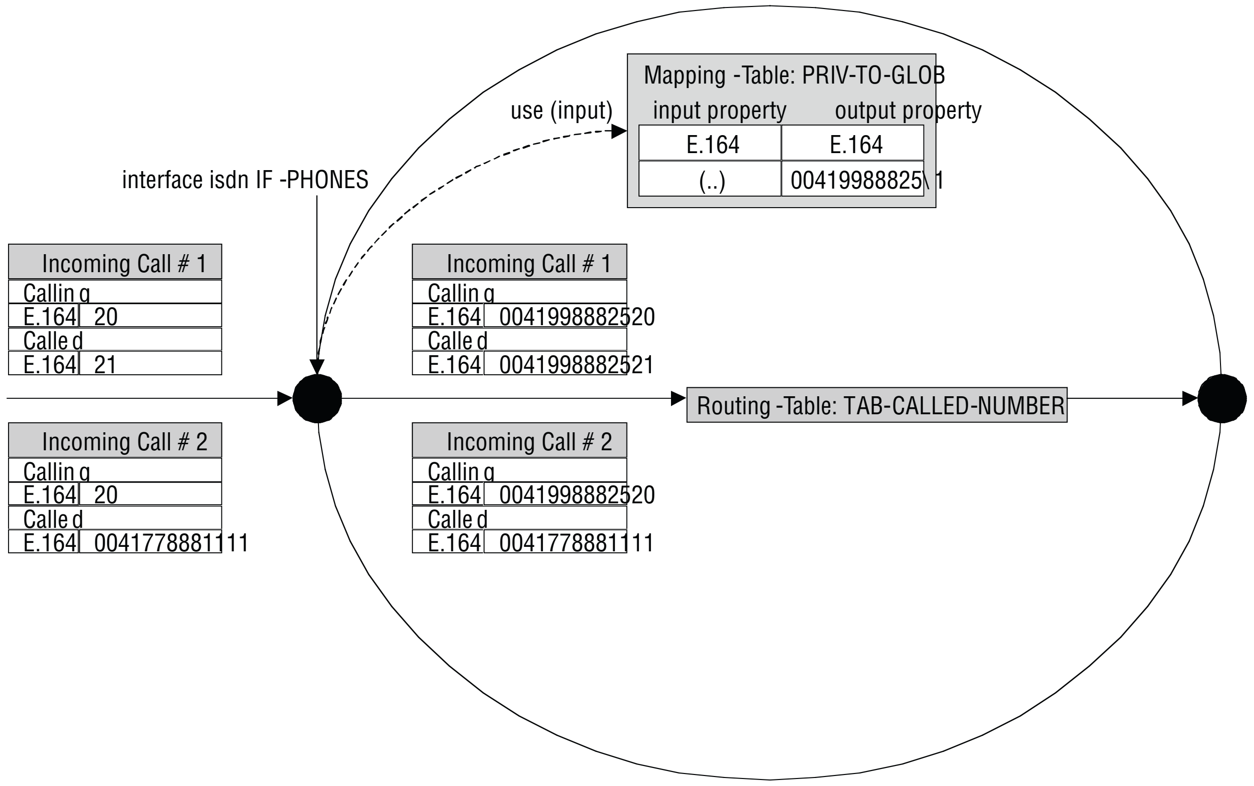


Figure 70. Incoming call passing an interface mapping table

Figure 70 shows two incoming calls arriving to the ISDN interface IF-PHONES. The calling and called party numbers are private numbers containing only two digits. Before accessing the call router, those numbers can be transformed into the global numbering plan. This is why the interface was configured to use mapping table PRIV-TO-GLOB on all incoming call properties.

Incoming call #1 originally has a calling party number of 20 and a called party number of 21. Before offering this call to the call router, mapping table PRIV-TO-GLOB is applied to the called party number and the calling party number. The mapping table adds a prefix of 00419988825 to the called and calling party number.

Incoming call #2 originally has a calling party number of 20 but already a called party number of the global numbering plan. Again, the mapping table is applied to both number, but only the calling party number of 20 is translated into 0041998882520. The called party number does not match an entry in the mapping table, so it is not changed.

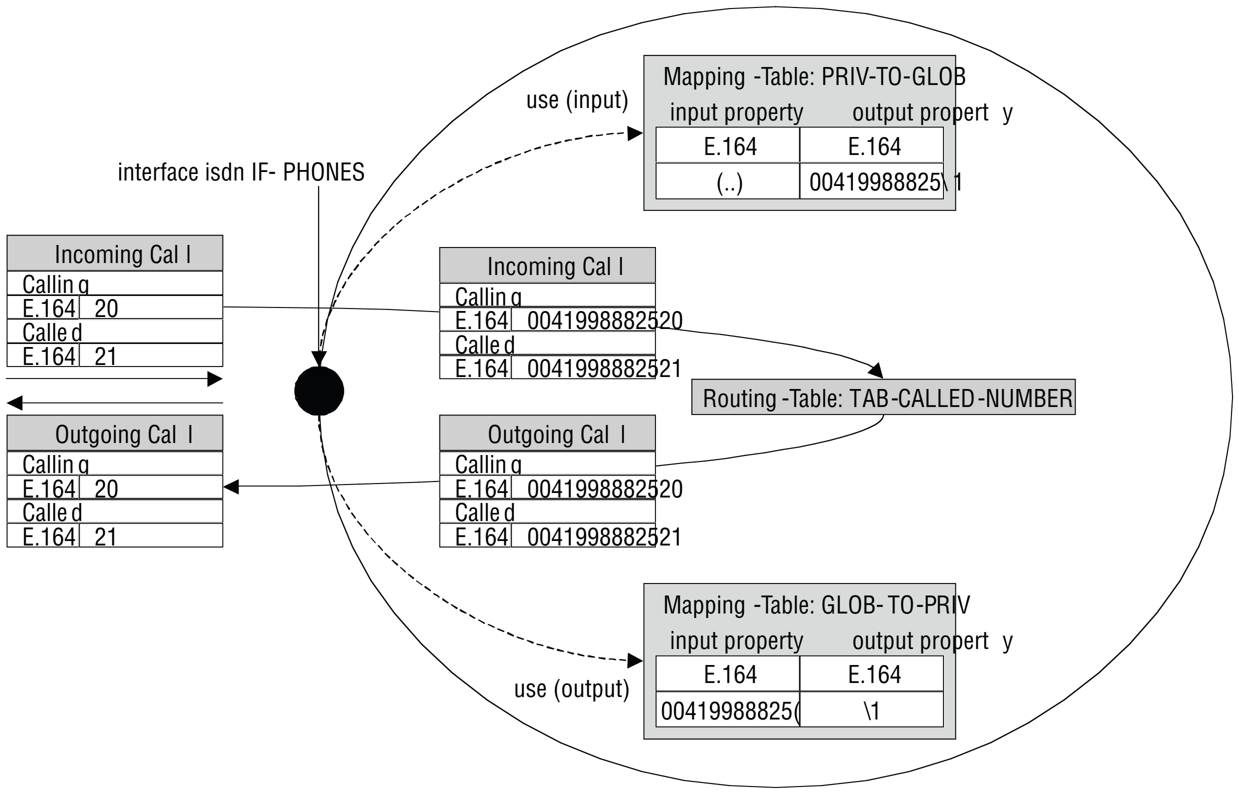


Figure 71. Call passing an input and an output mapping table

Let's assume we manipulate an incoming ISDN call using the PRIV-TO-GLOB mapping table as in the previous example. Figure 71 shows this situation again. Let's further assume the call router routes back the call to the interface IF-PHONES. In that case, the output mapping table used on this interface is applied to all call parameters. The calling and called party number is transformed form the global to the private numbering plan before the call is offered to the remote ISDN terminal.

Note For interface mapping you can use only mapping tables that examine general call parameters. For example, you cannot use a called-e164 to called-e164 mapping table, use a e164 to e164 mapping table instead.

Chapter 42 Tone Configuration

Chapter contents

Introduction

This chapter gives an overview of call-progress-tone profiles and tone-set profiles, and describes the tasks involved in their configuration.

In-band tones keep the user informed about the state of his call or additional services such as call-waiting, hold etc. Other tones can be assigned to any event that occurs during a call, a call waiting tone, for example. The in-band tones are referred to as *call-progress-tones*.

Tone-set Profiles

In traditional PSTN networks the in-band tones (dial tone, alerting tone, busy tone etc.) are generated by the network, i.e. the Central Office switch or a similar device, and are relayed transparently by the Patton device. In voice over IP networks however this model of a network side providing services including in-band tones is not given in all situations. For example, two Patton devices may be connected directly to each other over the access network without the intervention of a traditional Central Office switch. This imposes the need to generate the local in-band tones directly on the gateways since none of the attached ISDN devices (PBXs, phones) will do so itself (ISDN USR side). The in-band tones that can be generated by the Patton device are the following:

- Busy tone—Tone you hear when you try to reach a remote extension but it is busy.
- Confirmation tone—Tone you hear when you enable a supplementary service and the system has accepted and activated it (for future use).
- Congestion tone—Tone you hear when you try to reach a remote extension but the network is busy or out of order (for future use).
- Dial tone—Tone you hear when you lift the handset and the network is ready to accept the dialed digits of the called party number.
- Hold tone—Tone you hear when you are in an active connection and the remote extension sets you ‘On Hold’ to reach a third party extension.
- Release tone—Tone you hear when you are in an active connection and the remote extension terminates the call.
- Ringback tone—Tone you hear when the called party number is complete and the remote extension is ringing.
- Special dial tone—Tone you hear when you lift the handset and the network is ready to accept the dialed digits of the called party number, but on your system is still an activated supplementary service (for future use).
- Special Information tone—Tone you hear when you try to reach a nonexistent remote extension (for future use).
- Waiting tone—Tone you hear when you already have an active connection and a second new extension tries to reach you.

All call-progress-tones are collected in a tone-set profile. A tone-set profile collects typically all required tones for one country. The tone-set profile is assigned to the PSTN interface (ISDN, FXS, FXO) or if it is required to have different tones for individual PSTN interfaces it’s possible to assign for each PSTN interface its own tone-set profile. If no tone-set is assigned to a PSTN interface, the default tone-set is taken. Figure 72 on page 426 illustrates the relation ship between call-progress-tone profiles, tone-set profiles and PSTN interfaces.

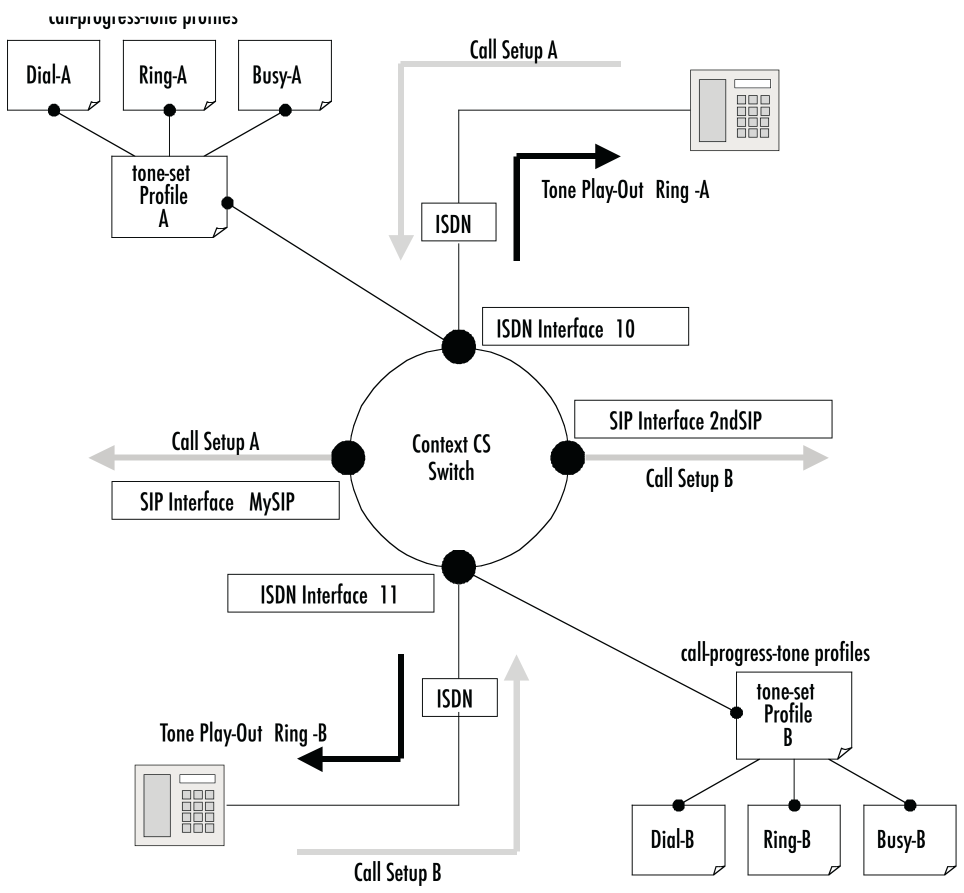


Figure 72. Assign tone-sets to a PSTN interfaces

Note There is a default tone-set named *default*, which maps the three Swiss standard in-band tones. Create a tone-set profile only if this default profile corresponds not with your country.

Tone Configuration Task List

To configure call progress tones, perform the tasks described in the following sections.

- Configuring call-progress-tone profiles
- Configuring tone-set profiles
- Enabling the generation of local in-band tones
- Showing call-progress-tone and tone-set profiles

Configuring Call-Progress-Tone Profiles

Each call-progress-tone consists of a sequence of different tones and pauses. Arbitrary tone cadences can be configured. All country specific tones can be defined with these parameters. Tone configuration knows only one command that has to be used repeatedly. The sequence in which the commands are entered (or appear in the config file) defines the sequence in which the corresponding elements are played.

Procedure: To configure a tone-set profile

Mode: Configure

Step	Command	Purpose
1	***node*(cfg)#profile call-progress-tone** *name*	Creates a call-progress-tone profile with name *name* and enters call-progress-tone configuration mode.
2	***node*(pf-callp)[*name*]#play** *duration frequency1 level1 [frequency2 level2]*	Defines a tone with duration, frequency *frequency1* and volume *level1*. If a second frequency is defined both frequencies are played in parallel and for the same duration
3	***node*(pf-callp)[*name*]#pause** *duration*	Defines a pause of *duration* milliseconds
4	***node*(pf-callp)[*name*]#...**	Repeat step 2 and/or step 3 to define a tone sequence. Always when you enter a **play** or **pause** command, it is appended to the already existing tone.
5	***node*(pf-callp)[*name*]#flush-play-list**	Resets the tone cadence. Same as deleting and re-creating the tone.

Example: Define the Belgian special information tone

The first line defines the first element of the tone: 330ms of 950Hz at –4dB. The second line the element that is played when the first element has finished: 330ms of 144Hz at –4dB, and so on. The last line defines a pause of 1 second after the three tones. The cadence is repeated infinitely.

```
node(cfg)#profile call-progress-tone belgianSpec
node(pf-callp)[belgian~]#play 330 950 -4
node(pf-callp)[belgian~]#play 330 1400 -4
node(pf-callp)[belgian~]#play 330 1800 -4
node(pf-callp)[belgian~]#pause 1000
```

Tones and pauses can be arbitrarily sequenced up to a number of 10 elements per call-progress-tone. The default call-progress-tone is an empty tone. The total number of different *play* elements across all configured call-progress-tones must not exceed 15 (an error is thrown if it does). If the call-progress-tone consists of only one element, this element has infinite duration. The *duration* parameter is ignored in this case.

Configure Tone-Set Profiles

A tone-set profile maps one call-progress-tone profile to each internal call-progress-tone. A tone-set profile typically includes all the call-progress-tones for one country.

Procedure: To configure a tone-set profile

Mode: Configure

Step	Command	Purpose
1	***node*(cfg)#profile tone-set** *name*	Creates tone-set *name* and enters tone-set profile configuration mode.

Step	Command	Purpose
2	***node*(pf-tones)[*name*]#map call_progress_tone { busy-tone | confirmation-tone| congestion-tone | dial-tone | hold-tone | release-tone | ringback-tone | special-dial-tone | special-information-tone | waiting-tone }** *call-progress-tone*	Map a call-progress-tone profile to an internal tone. An internal tone represents the call event for which a tone indication can be provided. Use the CLI help to get a list of all available events.
3		Repeat step 2 for all internal tone events.
4	[*name*](**pf-tones**)[*name*]#**[dtmf-signal-level**	Defines the output level of DTMF signals generated locally. This applies also to relayed DTMF signals.

Example: Configuring a tone-set

The following example shows how to configure a tone-set profile for UK.

```
node(cfg)#profile tone-set UK
node(pf-tones)[UK]#map call_progress_tone dialtone dialUK
node(pf-tones)[UK]#map call_progress_tone alertingtone ringUK
node(pf-tones)[UK]#map call_progress_tone busytone busyUK
```

Enable Tone-Set Profile

A call on the Patton device always has two signaling protocol endpoints. At the moment it is only possible to play locally generated tones on PSTN endpoints (ISDN, FXS) and not on IP based signaling endpoints (SIP). Dependent on the configuration several combinations of signaling protocol endpoints are possible (ISDN–ISDN, FXS-SIP etc.). The Patton device will always generate the tones locally and play it on the PSTN line as long as the other endpoint doesn't notifies availability of in band information and the PSTN endpoint is NOT of type ISDN-USER or FXO. If availability of in band information will be notified by one endpoint, the bearer channel already contains the necessary tone information and must not be generated locally.

If the user has not specified a tone-set profile, the default tone-set will be taken to generate the local in band information. For enabling a user defined tone-set profiles on a specific interface proceed as follows.

Procedure: To assign a tone-set profile to a PSTN interface

Mode: Interface

Step	Command	Purpose
1	***node*(ctx-cs)[switch]#interface** *if-type if-name*	Enter interface configuration mode.

Step	Command	Purpose
2	***node*(if-*type*)[*if-name*]#use profile tone-set** *name*	Assign a user defined tone-set profile to an interface.

Example: Assign tone-set profiles to an ISDN interface

The example shows how to use the SWISS tone-set for the CS context, and use the USA tone-set for an individual interface.

```
node(cfg)#context cs
node(ctx-cs)[switch]#interface isdn bri0
node(if-isdn)[bri0]#use profile tone-set USA
```

Show Call-Progress-Tone and Tone-Set Profiles

Use the show commands to display the call-progress-tone profiles as well as the tone-set profiles.

Procedure: To show call-progress-tone profiles

Mode: Administrator execution

Step	Command	Purpose
1	***node*#show profile call-progress-tone** [*name*]	Display all call-progress-tone profiles or a specific with a name

Example: Show call-progress-tone profile

The following example shows how to display the call-progress-tone profiles.

```
node#show profile call-progress-tone belgianSpec
Profiles:
---------

belgianSpec:
  Play  330ms (950Hz at -4dB)
  Play  330ms (1400Hz at -4dB)
  Play  330ms (1800Hz at -4dB)
  Pause 100ms
```

Procedure: To show tone-set profiles

Mode: Administrator execution

Step	Command	Purpose
1	***node*#show profile tone-set** [*name*]	Display all tone-set profiles or a specific with name *name*

Example: Show tone-set profile

The following example shows how to display the tone-set profile.

```
node#show profile tone-set test
Tone Profile: test
==================

 Used:                                       by 0 module(s)
 DTMF Duration:                              80ms
 DTMF Interspace:                            80ms

 Tones
 -----

  dial-tone:                                 belgianSpec
  ringback-tone:                             defaultAlertingtone
  hold-tone:                                 defaultHoldtone
  waiting-tone:                              defaultWaitingtone
  confirmation-tone:                         defaultConftone
  busy-tone:                                 defaultBusytone
  congestion-tone:                           defaultCongestiontone
  release-tone:                              defaultReleasetone
  special-information-tone:                  defaultSItone
  special-dial-tone:                         defaultSDtone
```

Example: The following example shows how to configure a tone-set profile for UK and apply it to the isdn interface bri0.

Create the call-progress-tone profiles:

```
node(cfg)#profile call-progress-tone dial-UK
node(pf-callp)[dial-UK]#play 5000 350 0 440 0

node(pf-callp)[dial-UK]#profile call-progress-tone alerting-UK
node(pf-callp)[alertin~]#play 400 400 0 450 0
node(pf-callp)[alertin~]#pause 200
node(pf-callp)[alertin~]#play 400 400 0 450 0
node(pf-callp)[alertin~]#pause 2000

node(pf-callp)[alertin~]#profile call-progress-tone busy-UK
node(pf-callp)[busy-UK]#play 400 400 0
node(pf-callp)[busy-UK]#pause 400
node(pf-callp)[busy-UK]#exit

Create the tone-set profile:

node(cfg)#profile tone-set UK
node(pf-tones)[UK]#map call_progress_tone dialtone dial-UK
node(pf-tones)[UK]#map call_progress_tone alertingtone alerting-UK
node(pf-tones)[UK]#map call_progress_tone busytone busy-UK
node(pf-tones)[UK]#exit

Assign the tone-set to the isdn interface bri0

node(cfg)#context cs
node(ctx-cs)[switch]#interface isdn bri0
node(if-isdn)[bri0]#use profile tone-set UK
```

Chapter 43 Authentication Service

Chapter contents

Introduction

This chapter describes how to configure authentication services in Trinity. The *Authentication Service* is a data base that manages *Authentication Credentials* of one or more *Realm*. A *Realm* is an *Authentication Zone* or *Authentication Domain* that defines the authentication responsibility in a network. Each *Authentication Credential* created in an Authentication Service belongs to the defined Realm and exists on a User Name and an optional Password. It is also possible to create an Authentication Service without specifying a Realm, which represents the default realm. Whenever authentication is required and the provided Realm doesn't exist in an Authentication Service, this default realm will be considered to find the right Authentication Credentials.

Authentication Service Configuration Task List

The following section describes how to create a new authentication service and how to enter the configuration mode of an existing service. Additionally, it describes all commands and sub commands of the authentication service configuration mode. All configuration tasks for Authentication Services are listed below.

- Create an Authentication Service (see page 432)
- Configure a Realm (see page 433)
- Configure the authentication protocol (see page 433)
- Create credentials (see page 433)

Creating an Authentication Service

The **authentication-service** command enters the configuration mode of an existing authentication or creates a new one if the requested service does not yet exist. The *no* form of the command destroys the authentication service.

Mode: Configure

Step	Command	Purpose
1	**[*node*](cfg)# [no] authentication-service** *<name>*	Creates/Destroys an authentication service or enters existing authentication-service configuration mode.

Configuring a Realm

The following commands add a new Realm to the authentication service. If more than one Realm has to be entered, the order of the list can be modified by using the *index* and/or *before* and *after* keywords. The no form of the command removes an existing Realm from the list.

Mode: Authentication Service

Step	Command	Purpose
1	[*node*]**(ls)[name]# [no] realm** <*name*> OR [*node*]**(ls)[name]#realm** <*index*> <*name*> OR [*node*]**(ls)[name]#realm before** <*index*> <*name*> OR [*node*]**(ls)[name]#realm after** <*index*> <*name*>	Adds or removes a Realm to/from the authentication service.

Configuring the Authentication Protocol

The **protocol** command specifies the protocol.

Mode: Authentication Service

Step	Command	Purpose
1	[*node*]**(ls)[name]#protocol {http}**	Specifies the authentication protocol to be used.

Creating Credentials

The following command creates Authentication Credentials identified by the entered username. The no form of the command removes an existing Credential. It is possible to enter this command without a password.

Mode: Authentication Service

Step	Command	Purpose
1	[*node*]**(ls)[name]# [no] username** <*user*> **password** <*password*>	Creates or removes authentication credentials.

Configuration Examples

```
authentication-service AUTH_SRV
  realm 1 voip-public
  realm 2 voip-intranet
  realm 3 ms-exchange
  username 433 password fK+bfnzL45Goh/VdjrWxAA== encrypted
  username john.doe password D60t7CBZ58k7JK2jxdlw4w== encrypted
```

Chapter 44 Location Service

Chapter contents

Introduction

This chapter describes how to configure location services in Trinity.

Location Service Configuration Task List

The following section describes how to create a new location service and how to enter the configuration mode of an existing service. Additionally, it describes all commands and sub commands of the location service configuration mode. All configuration tasks for Location Services are listed below:

- Create a Location Service (see page 435)
- Add a domain (see page 435)
- Create an identity (see page 436)
- Create an identity group (see page 449)
- Inherit from an identity group to an identity (see page 449)

Creating a Location Service

The **location-service** command enters the configuration mode of a location service. If the requested service does not yet exists, a new one will be created. The no form of the command removes an existing location service.

Mode: Configure

Step	Command	Purpose
1	[*device*]**(cfg)# [no] location-service** <*name*>	Creates/Destroys a location service or enters configuration mode.

Adding a Domain

The **domain** command specifies the domains that the location service is responsible for. If the application needs information from the location service, it performs a lookup with the Host Part of the Request-URI or the From-URI to find the right instance. The header selection from which the URI will be taken depends on the call direction (Outgoing/Incoming SIP-Call) and the requested information. The SIP environment determines which format the Domain has; it can either be a Domain-Name, a Host-Name or a Host-Address. If all components of the SIP environment are set up to operate in one specified domain, Domain is a Domain-Name. If point-to-point routing is used, Domain is either a Host-Name or a Host-Address. In this case, Host-Name is a FQDN (Full Qualified Domain Name).

Domain Examples:

Domain-Name: *biloxy.com*
Host-Name: *sip-ua.biloxy.com* or *sip-server.biloxy.com*
Host-Address: *192.168.10.1* or *10.10.10.1*

In case of point-to-point routing, local host-addresses may not be added to the domain list of a location service; these addresses are known by the application. But, if an identity exists in two different location services and the

Context SIP Gateway has more than one transport binding, it is recommended to add the local host addresses as *Domain* to the appropriated location services.

Mode: Location Service

Step	Command	Purpose
1	[*device*]**(svc-ls)[ls]# [no] domain** <*name*> OR [*device*]**(svc-ls)[ls]#domain** <*index*> <*name*> OR [*device*]**(svc-ls)[ls]#domain before** <*index*> <*name*> OR [*device*]**(svc-ls)[ls]#domain after** <*index*> <*name*>	Adds a new domain to the location service. If more than one domain has to be entered, the order of the list can be modified by using the index and/or the insert keywords *before* and *after*. The no form of the command removes an existing domain from the list.

Configuring Default Responsibility

A location service can be configured to accept responsibility for any domain. This eases the matching of dynamic ip addresses into the location service. When the Sip gateway does a lookup in the database it considers the location service where a configured domain matches the host part of the identity in the first priority. If no location services are found, the first location service which is responsible for any domain is taken.

Mode: Location Service

Step	Command	Purpose
1	[*device*]**(svc-ls)[ls]#[no] match-any-domain**	Adds or removes the responsibility for any possible domain or ip address.

Creating an Identity

An identity represents one of multiple possible addresses over which a user is reachable (e.g. sip:john@patton.com). This leads to a huge range of configuration possibilities in the identity.

According to the relationship between an identity and user, there can be many different aspects configured. If you are the user agent for a certain identity, use the outbound faces to specify the behavior when sending requests. If you are not the user agent of an identity but know this identity, then use the inbound faces to configure the behavior when this identity sends requests.

When creating an identity, it is important to consider that the name of the identity is always used as user-part when building a sip-uri. The name of the identity is also used when comparing to or matching with a sip-uri.

Mode: Location Service

Step	Command	Purpose
1	[*device*]**(svc-ls)[ls]#[no] identity** <*name*>	Adds a new identity to the location service. The no form of the command removes an existing identity.

Mode: Identity

Step	Command	Purpose
1	[*device*]**(identity)[device]# [no] display-name** *<display-name>*	Adds a display-name to the identity. The no form removes the display-name.

Mode: Identity

Step	Command	Purpose
1	[*device*]**(identity)[device]# [no] phone-context** *<phone-context>*	Adds a phone-context to the identity. The no form removes the phone-context.

An alias is an alternative way to express the user-part of an identity. An alias is never used to build a sip-uri and will never be used in communication with another device. The alias is used for comparing or matching the identity with a sip-uri received from an external device.

Mode: Identity

Step	Command	Purpose
1	[*device*]**(identity)[device]# [no] alias name** *<alias>* *OR* [*device*]**(identity)[device]# [no] alias range** *<start><end>* *OR* [*device*]**(identity)[device]# [no] alias expression** *<expression>*	Adds a new alias to the identity. The no form of the command removes an existing alias. **[no]** removes the alias again. Range values must be numerical and the start value must be smaller than the end value.
2	[*device*]**(identity)[device]# alias none**	Clears the entire alias list for this identity.

RFC3261 defines a user-param for SIP URIs. The value of this parameter can be configured for identities in the location service. The configured value will then be applied to any SIP URI for which the user part matches the identity.

- **phone:** This identity represents a telephone number. This is typically used when a phone-context has been configured.
- **dialstring:** This identity represents a dialstring.
- **ip:** This identity represents an ordinary SIP user. This is the default value assumed by SIP also if no user-param is specified. Thus it usually does not make any difference whether “user ip” or “no user” is specified even though the generated SIP URIs will be different.

Mode: Identify

Step	Command	Purpose
1	**[device](identify)[device]# [no] user {phone | dialstring | ip}**	Defines the value of the user-param in SIP URIs where this identity is present.

The huge amount of possible configuration parameters has been separated into different configuration entities called the faces. The faces refer to authentication, registration and call. Each face works in two directions: inbound and outbound. Outbound faces refer to requests originating from your identity. Inbound faces refer to requests destined to your identity.

- Authentication outbound face (see page 438)
- Authentication inbound face (see page 439)
- Registration outbound face (see page 440)
- Registration inbound face (see page 444)
- Call outbound face (see page 445)
- Call inbound face (see page 447)

Authentication outbound face

The authentication outbound face is used to provide authentication credentials to challenges from other user agents or proxies.

Mode: Identity

Step	Command	Purpose
1	**[*device*](identity)[device]# [no] authentication outbound**	Adds a new face to the identity. The no form of the command removes an existing face with all content in it.

An authentication entry establishes a link between an identity and exactly one pair of credentials in an authentication-service. To link multiple credentials to an identity, there must be one authentication entry in the authentication outbound face for each pair of credentials to link. The parameter username selects the entry in the authentication-service to use. The parameter username can be omitted if and only if the name of the identity matches exactly the username in the authentication-service.

Mode: Authentication outbound

Step	Command	Purpose
1	[*device*]**(authout)#authenticate authentication-service** <*authentication-service*> **[username** <*username*>**]** OR [*device*]**(authout)#authenticate** <*index*> **authentication-service** <*authentication-service*> **[username** <*username*>**]** OR [*device*]**(authout)#authenticate before** <*index*> **authentication-service** <*authentication-service*> **[username** <*username*>**]** OR [*device*]**(authout)#authenticate after** <*index*> **authentication-service** <*authentication-service*> **[username** <*username*>**]**	Adds a new authentication entry to the authentication outbound face. If more than one authentication entry has to be entered, the order of the list can be modified by using the index and/or the insert keywords before and after.

Mode: Authentication outbound

Step	Command	Purpose
1	[*device*]**(authout)#no authenticate [**<*index*>**]**	Removes the authentication entry at the index or removes all authentication entries if no index is given.

Mode: Authentication outbound

Step	Command	Purpose
1	[*device*]**(authout)#authenticate none**	Removes all authentication entries and disables explicitly authentication outbound.

Authentication inbound face

The authentication inbound face is used when you want to challenge other user agents.

Mode: Identity

Step	Command	Purpose
1	[*device*]**(identity)[device]# [no] authentication inbound**	Adds a new face to the identity. The no form of the command removes an existing face with all content in it.

Mode: Authentication inbound

Step	Command	Purpose
1	[*device*]**(authin)#authenticate authentication-service** <*authentication-service*> **[username** <*username*>**]** OR [*device*]**(authin)#authenticate** <*index*> **authentication-service** <*authentication-service*> **[username** <*username*>**]** OR [*device*]**(authin)#authenticate before** <*index*> **authentication-service** <*authentication-service*> **[username** <*username*>**]** OR [*device*]**(authin)#authenticate after** <*index*> **authentication-service** <*authentication-service*> **[username** <*username*>**]**	Adds a new authentication entry to the authentication inbound face. If more than one authentication entry has to be entered, the order of the list can be modified by using the index and/or the insert keywords before and after.

Mode: Authentication inbound

Step	Command	Purpose
1	[*device*]**(authin)#no authenticate [**<*index*>**]**	Removes the authentication entry at the index or removes all authentication entries if no index is given.

Mode: Authentication inbound

Step	Command	Purpose
1	[*device*]**(authin)#authenticate none**	Removes all authentication entries and disables explicitly authentication inbound.

Registration outbound face

The registration outbound face is used to register an identity on an external registrar. Then, the registrar forwards calls from the registered identity to your identity.

Mode: Identity

Step	Command	Purpose
1	[*device*]**(identity)[device]# [no] registration outbound**	Adds a new face to the identity. The no form of the command removes an existing face with all content in it.

Mode: Registration outbound

Step	Command	Purpose
1	**device(regout)# [no] register [auto|none|back-to-back]**	Enables registration with auto or disables registration explicitly with none. For the back-to-back option, refer to section "SIP B2BUA Dynamic Registration" on page 443.

Mode: Registration outbound

Step	Command	Purpose
1	[*device*]**(regout)# [no] registrar** *<host>* [*<port>*]	Configures the address of the registrar to send your register requests.

Mode: Registration outbound

Step	Command	Purpose
1	[*device*]**(regout)# [no] lifetime** *<seconds>*	Configures the desired lifetime of the registration. When no lifetime is configured the desired lifetime is set to 3600 seconds.

Mode: Registration outbound

Step	Command	Purpose
1	[*device*]**(regout)# [no] retry-timeout (on-client-error|on-sysstem-error|on-server-error)** *<seconds>*	Configures the time to wait after a failed registration according to three different error categories. After this time we begin to register from new. If no retry-timeout is configured the retry-timeout is set to 10 seconds.

Mode: Registration outbound

Step	Command	Purpose
1	[*device*]**(regout)#proxy** *<host>* [*<port>*] **[strict-route]** OR [*device*]**(regout)#proxy** *<index> <host>* [*<port>*] **[strict-route]** OR [*device*]**(regout)#proxy before** *<index> <host>* [*<port>*] **[strict-route]** OR [*device*]**(regout)#proxy after** *<index> <host>* [*<port>*] **[strict-route]**	Adds a new proxy entry to the registration outbound face. For each proxy configured there is a route-header added. If more than one proxy entry has to be entered, the order of the list can be modified by using the index and/or the insert keywords before and after.

Mode: Registration outbound

Step	Command	Purpose
1	[*device*]**(regout)#proxy** <*index*> **down** <*positions*> OR [*device*]**(regout)#proxy** <*index*> **up** <*positions*>	If multiple proxies are configured the entry at the index can be moved in the proxy list up or down the number of positions given in the command.

Mode: Registration outbound

Step	Command	Purpose
1	[*device*]**(regout)#no proxy [**<*index*>**]**	Removes the proxy entry at the index or remove all proxy entries if no index is given.

Mode: Registration outbound

Step	Command	Purpose
1	[*device*]**(regout)#proxy none**	Removes all proxy entries and disables explicitly the use of a proxy.

Mode: Registration outbound

Step	Command	Purpose
1	[*device*]**(regout)#[no] preferred-transport-protocol (tcp | udp)**	Selects the preferred transport protocol for conveying the REGISTER request.

Registration Priority. The *q* parameter is a value between 0 and 1 and specifies the weight of a REGISTER message. It is mainly used to control the call handling priority and sequence in forking applications with support for multiple registrations of the same user.

The default value of 1 means that the *q* value will not be added to the contact header. By default, the *q* value is disabled.

Mode: Registration outbound

Step	Command	Purpose
1	[*device*]**(regout)#[no] priority <priority>**	Specifies the weight of the registration in per mille (Example: 800 -> q='0.8'). Default = Disabled

DNS Security Check. Upon receipt of a redirection response (a 300, 301 or 302 SIP response), the SIP User Agent formulates a new REGISTER request based on the request specified in the Contact field. It is possible to configure to register using the contact only if it matches one of the addresses previously received from the DNS resolution.

Mode: Registration outbound

Step	Command	Purpose
1	[*device*]**(regout)#[no] check-dns-addresses**	Use only registrar pertaining to addresses previously received from DNS resolution.

Example: Enable DNS Security Check

```
location-service LOC
 domain 1 example.com

 identity 100

   registration outbound
     check-dns-addresses
     register auto
```

Support broken SIP proxy. To direct SIP requests to outbound proxies without having a proxy header in the message a new command is introduced. This command specifies a destination host (with optional port) to which SIP messages will be sent regardless of domain, proxies and registrar.

Note This feature breaks RFC 3261! Use this feature only if you are certain that the addressed behaves as expected and is able to process such messages.

Mode: Registration outbound

Step	Command	Purpose
1	[*device*]**(regout)#[no] force-destination [host [port]]**	Specifies a host and optional port to send registration messages, overriding proxies, registrar and domain.

SIP B2BUA Dynamic Registration

This feature gives the possibility to trigger an outbound registration when a matching inbound registration has been detected. As prerequisite you will need to define an inbound registration for the desired identity in the location service. You will need to define a registrar in the outbound registration configuration as well.

Mode: location-service

Step	Command	Purpose
1	**device(ls)[name]#[no] identity <name>**	Adds a new identity to the location service.
1	**device(identity)# [no] authentication inbound**	Define if inbound authentication is used or not.
1	**device(identity)# [no] authentication outbound**	Define if outbound authentication is used or not.
1	**device(identity)[name]# [no] registration inbound**	Add inbound registration

Step	Command	Purpose
1	**device(identity)[name]# [no] registration out-bound**	Add outbound registration
1	**device(regout)# [no] register [auto|none|back-to-back]**	Set the back-to-back parameter in the outbound registration configuration

Registration inbound face

The registration inbound face is used when you want to allow external user agents to register, so that you can route calls to the registered contacts.

Mode: Identity

Step	Command	Purpose
1	[*device*]**(identity)[device]# [no] registration inbound**	Adds a new face to the identity. The no form of the command removes an existing face with all con-tent in it.

Mode: Registration inbound

Step	Command	Purpose
1	[*device*]**(regin)# [no] lifetime [default** <*sec-onds*>**] [min** <*seconds*>**] [max** <*seconds*>**]**	Configures the range of the expiration time accepted for inbound registration. If there is a time requested which is out of the range the time is set to a value which fits the range. If there is no time requested it is set to the default lifetime.

Mode: Registration inbound

Step	Command	Purpose
1	[*device*]**(regin)# [no] contact** <*host*> **[**<*port*>**]** <*context cs*> <*interface sip*> **[priority** <*pro-mille*>**]**	Adds a contact to the registration inbound face. The interface and context parameter defines where the sip-location-service routes a call des-tined to that contact. The higher priority a contact has the sooner a contact is chosen. The default priority is 1000 which is also the maximum prior-ity. The no form of the command removes a con-tact.

Call outbound face

The call outbound face is used to configure call properties for outgoing calls.

Mode: Identity

Step	Command	Purpose
1	[*device*]**(identity)[device]# [no] call outbound**	Adds a new face to the identity. The no form of the command removes an existing face with all content in it.

Mode: Call outbound

Step	Command	Purpose
1	[*device*]**(callout)device# [no] use profile (sip|tone-set|voip)** <*profile*>	Adds or removes a profile to use if an outgoing call destines this identity

Mode: Call outbound

Step	Command	Purpose
1	[*device*]**(callout)device# [no] preferred-transport-protocol (tcp|udp)**	Selects which protocol to prefer if an outgoing call destines this identity.

Mode: Call outbound

Step	Command	Purpose
1	[*device*]**(callout)device# [no] traffic-class** <*traffic-class*>	Selects which traffic class to set if an outgoing call destines this identity.

Mode: Call outbound

Step	Command	Purpose
1	[*device*]**(callout)#proxy** <*host*> **[**<*port*>**] [strict-route]** OR [*device*]**(callout)#proxy** <*index*> <*host*> **[**<*port*>**] [strict-route]** OR [*device*]**(callout)#proxy before** <*index*> <*host*> **[**<*port*>**] [strict-route]** OR [*device*]**(callout)#proxy after** <*index*> <*host*> **[**<*port*>**] [strict-route]**	Adds a new proxy entry to the call outbound face. If the from-uri of a call originating from us matches the identity for each proxy configured there is a route-header added. If more than one proxy entry has to be entered, the order of the list can be modified by using the index and/or the insert keywords before and after.

Mode: Call outbound

Step	Command	Purpose
1	[*device*]**(callout)#proxy** *<index>* **down** *<positions>* OR [*device*]**(callout)#proxy** *<index>* **up** *<positions>*	If multiple proxies are configured the entry at the index can be moved in the proxy list up or down the number of positions given in the command.

Mode: Call outbound

Step	Command	Purpose
1	[*device*]**(callout)#no proxy [***<index>***]**	Removes the proxy entry at the index or remove all proxy entries if no index is given.

Mode: Call outbound

Step	Command	Purpose
1	[*device*]**(callout)#proxy none**	Removes all proxy entries and disables explicitly the use of a proxy.

Configuring the Dynamic Registrar Use. Upon receipt of a redirection response (a 300, 301 or 302 SIP response), the SIP User Agent formulates a new REGISTER request based on the request specified in the Contact field. It is possible to configure that all the subsequent SIP requests be directly sent to the redirected registrar if the REGISTER succeeded.

Mode: Call outbound

Step	Command	Purpose
1	[*device*]**(callout)#[no] request-uri dynamic-registrar**	Enable the use of the dynamic registrar to direct the request. Default: Disabled.

Example: Enable the Use of the Dynamic Registrar

```
location-service LOC
 domain 1 example.com

 identity 100

   call outbound
     request-uri dynamic-registrar
```

Support broken SIP proxy. To direct SIP requests to outbound proxies without having a proxy header in the message a new command is introduced. This command specifies a destination host (with optional port) to which SIP messages will be sent regardless of domain, proxies and registrar.

Note This feature breaks RFC 3261! Use this feature only if you are certain that the addressed behaves as expected and is able to process such messages.

Mode: Call outbound

Step	Command	Purpose
1	**device(callout)#[no] force-destination manual [host [port]]**	Specifies a host and optional port to send INVITE messages, overriding proxies, registrar and domain.
2	**node(callout)#[no] force-destination registrar address**	Allows sending INVITE requests to the IP Address of the registrar where a given identity is currently registered to. For this purpose the destination IP address is forced to the detected registrar address independently of any URLs present in any SIP header of the INVITE packet.

Calls using flows. The concept of flows can be used for calls as well. When flows are opened by registrations, these are reused for outgoing calls, but only if the target resolution procedures for calls results in an IP address for which a flow already exists. Otherwise, a new flow is created for the duration of the call.

This does not affect incoming calls. Incoming calls are accepted through flows or outside of flows in the same way.

Mode: Call outbound

Step	Command	Purpose
1	**[device](callout)# [no] nat-traversal flows**	Configured if INVITE requests are sent out through flows.

Call inbound face

The call inbound face is used to configure call properties for incoming calls.

Mode: Identity

Step	Command	Purpose
1	[*device*]**(identity)[device]# [no] call inbound**	Adds a new face to the identity. The no form of the command removes an existing face with all content in it.

Mode: Call inbound

Step	Command	Purpose
1	[*device*]**(callin)device# [no] use profile (sip|tone-set|voip)** *<profile>*	Adds or removes a profile to use if an incoming call destines this identity.

Configuring SIP Transaction Timeout and Penalty Box

The transaction timeout defines how long the SIP gateway tries to establish a transaction to a certain destination without success before the transaction is considered as failed and a possible alternative destination is tried. This is the case when a DNS-lookup returns multiple ip-addresses for a destination. Therefore, the invite transaction timeout is taken for INVITE transactions and the non-invite transaction timeout for any other transaction.

When the penalty box is used, such a failed destination is put into the penalty box for 5 minutes. Any destination which is not in the penalty box has a higher priority for use as destinations listed in the penalty box.

For outgoing calls, the invite-transaction-timeout, the non-invite-transaction-timeout and the use of the penalty box is taken from the Identity corresponding to the request-uri or the identity-group "default" in the location service corresponding to the host part of the request-uri.

For incoming calls, the invite-transaction-timeout, the non-invite-transaction-timeout and the use of the penalty box is taken from the Identity corresponding to the from-uri or the identity-group "default" in the location service corresponding to the host part of the from-uri.

Mode: Identity/Call outbound

Step	Command	Purpose
1	[*device*]**(callout)# [no] invite-transaction-timeout** <*seconds*>	Sets the transaction timeout for invite transactions directed to the identity. Default: 32 seconds.

Mode: Identity/Call outbound

Step	Command	Purpose
1	[*device*]**(callout)# [no] non-invite-transaction-timeout** <*seconds*>	Sets the transaction timeout for non-invite transactions directed to the identity. Default: 32 seconds.

Mode: Identity/Call outbound

Step	Command	Purpose
1	[*device*]**(callout)# [no] penalty-box**	Enables or disables the use of penalty box for transactions directed to the identity. Default: disabled.

Creating an Identity Group

Multiple identities with the same properties can be grouped in an identity-group. The identity-group can be configured exactly in the same way and with the same parameters as an identity. An identity-group can only inherit configurations to identities, but they never play an active role.

The special identity-group inherits parameters to identities which are unknown or not configured. Even dynamically created identities from registration inbound inherits from the identity-group DEFAULT. Configured identities do not inherit from the identity-group DEFAULT unless is it explicitly configured to do so.

Mode: Location Service

Step	Command	Purpose
1	[*device*]**(svc-ls)[ls]#(svc-ls)[ls]#[no] identity-group** <*name*>	Adds a new identity-group to the location service. The no form of the command removes an existing identity-group.

Inheriting from an Identity Group to an Identity

An identity only inherits parameters from an identity-group if the parameters are not configured in the identity itself. Some commands allow the identity to explicitly disable some configurations that were otherwise inherited.

Mode: Location Service

Step	Command	Purpose
1	[*device*]**(svc-ls)[ls]#identity** <*name*> **inheritance** <*identity-group*>	Configures the identity to inherit parameters which are not configured from the identity-group.

Mode: Location Service

Step	Command	Purpose
1	[*device*]**(svc-ls)[ls]#identity-group** <*name*> **inheritance** <*identity-group*>	Configures the identity-group to inherit parameters which are not configured from another identity-group.

Mode: Location Service

Step	Command	Purpose
1	[*device*]**(svc-ls)[ls]#identity** <*name*> **no-inheritance**	Configures the identity to not inherit parameters from the identity-group.

Mode: Location Service

Step	Command	Purpose
1	[*device*]**(svc-ls)[ls]#identity-group** <*name*> **no-inheritance**	Configures the identity-group to not inherit parameters from the other identity-group.

Configuring the Message Waiting Indication Feature for SIP

Note Message Waiting Indication is programmed in three sections of Trinity, the FXS interface, ISDN interface, and the SIP Location service. The information below refers to information for configuring the Message Waiting Indication feature for SIP.

The SIP part of Trinity receives message waiting information according to the subscribe notify mechanism described in RFC 3265 and RFC 3842. It is possible to choose between *explicit* and *implicit subscription*. In **explicit subscription**, Trinity establishes the subscription by sending SUBSCRIBE messages to a configured message server. In **implicit subscription**, the message server gets the subscriber information due to configuration or another mechanism like registration. In any case, Trinity accepts and processes NOTIFY messages with message waiting information, which is then forwarded to the according destination.

Subscription

If an identity is added to a location-service which is bound to a context sip-gateway, the following procedure takes place:

1. Determine if identity should be subscribed.
   – The location-service containing the identity must have at least one domain configured.
   – The identity must have a message inbound face configured or inherited.
   – The subscription command must be set to “explicit”.
2. Build the address to subscribe. The name of the identity builds the user-part. The first domain configured in the location-service builds the host-part.
3. Build the address of the message server. The message server configured in the message inbound face is taken as request-uri. If no message server is configured the first domain configured in the location-service builds the request-uri.
4. Build expire header. If a lifetime is configured in the message inbound face the expire header is set to desired lifetime else the lifetime is set to 3600 seconds.
5. Build contact address. The spoofed-contact parameter of the sip-interface in the context sip gateway, through which the SUBSCRIBE request is sent, is set as contact address. If no spoofed-contact is configured the ip-address and port of the sip-interface in the context sip-gateway through which the SUBSCRIBE request is sent builds the contact address.
6. Send the SUBSCRIBE request.

If one of these steps has no result and fails, the subscription fails. After a certain timeout (which is configurable in the registration outbound face), the request is re-issued.

Outgoing SUBSCRIBE requests can provide authentication credentials.

Notification

If the gateway receives an incoming NOTIFY request, the following procedure takes place:

1. Determine to which sip interface in the context cs the request should be forwarded. This happens according the same rules as an incoming INVITE is forwarded.

2. Get Identity. All location services bound to the context sip-gateway are searched for the identity:
   – the identity matching the to-uri
   – the identity matching request-uri
   – the identity-group default
3. Get message inbound face. The identity found must have a message inbound face configured.
4. Check subscription. The option “subscription” must be set to “implicit” or “explicit”. In the case of explicit subscription the real state of the subscription is not examined. According to RFC 3265 NOTIFY requests must be handled even before subscription is completed.
5. Check event header. The event-header of the NOTIFY request must be set to “message-summary”.
6. Check content header. If present, the content header must be set to “application/simple-message-summary”
7. Forward message waiting information. Forward content of NOTIFY message through call-control according normal call routing to the destination provider.
8. Return 200 OK if all of these steps are successful.

If one of these steps fails the notification fails and an according message is sent.

Incoming NOTIFY requests can be challenged to provide authentication credentials.

Configuration

The message inbound face is used to subscribe an identity on a message server and enables the reception of NOTIFY messages with message waiting information.

Mode: Identity

Step	Command	Purpose
1	[*device*]**(identity)[device]#[no] message inbound**	Adds a new face to the identity. The no form of the command removes an existing face with all content in it.

Mode: Message inbound

Step	Command	Purpose
1	[*device*]**(msgin)#[no] subscribe [implicit | explicit | none]**	Enables subscription implicit, explicit or disables subscription with none.

Mode: Message inbound

Step	Command	Purpose
1	[*device*]**(msgin)#[no] message server** <*host*> [<*port*>]	Configures the address of the message server to send your subscription requests for explicit subscription. When no message-server is configured, the subscription requests are sent to the first domain entry in the location-service.

Mode: Message inbound

Step	Command	Purpose
1	[*device*]**(msgin)#[no] lifetime** <*seconds*>	Configures the desired lifetime of the explicit subscription. When no lifetime is configured the desired lifetime is set to 3600 seconds.

Mode: Message inbound

Step	Command	Purpose
1	[*device*]**(msgin)#[no] retry-timeout (on-client-error | on-system-error | on-server-error]** <*seconds*>	Configures the time to wait after a failed subscription according to three different error categories. After this time we begin to subscribe from new. If no retry-timeout is configured the retry-timeout is set to 10 seconds.

Mode: Message inbound

Step	Command	Purpose
1	[*device*]**(msgin)#proxy** <*host*> [<*port*>] **[strict-route]** *OR* [*device*]**(msgin)#proxy** <*index*> <*host*> [<*port*>] **[strict-route]** *OR* [*device*]**(msgin)#proxy before** <*index*> <*host*> [<*port*>] **[strict-route]** *OR* [*device*]**(msgin)#proxy after** <*index*> <*host*> [<*port*>] **[strict-route]**	Adds a new proxy entry to the message inbound face. For each proxy configured there is a route-header added to the SUBSCRIBE requests in explicit subscription. If more than one proxy entry has to be entered, the order of the list can be modified by using the index and/or the insert keywords before and after.

Mode: Message inbound

Step	Command	Purpose
1	[*device*]**(msgin)#proxy none**	Removes all proxy entries and disables explicitly the use of a proxy.

Message Waiting Indication through Call-Control

The message waiting indication received on the SIP side is transported through the call control to the destination interface. Currently, the SIP interface is the only source of message waiting indication information and the interface is the only possible destination of this information. The information is routed like a call through the call-control. Therefore, it must exist a valid routing path for a call from the SIP interface handling the NOTIFY to the destination interface. This must be the case, because the phone connected to the interface must be able to get calls from SIP before it makes any sense to get message waiting information for missed calls from SIP.

The message waiting information can be transported through routing-tables, service aaa, service limiter, service priority, service distribution-group and service hunt-group.

- In case of routing through **routing tables**, the message waiting information will reach the right destination only if the parameter according which the routing occurs is set to exactly the same value in an incoming INVITE request for that interface as in the incoming NOTIFY request for that interface.
- In the case of "**called-e164**", this is surely the same, but pay attention to the manipulation of parameters.
- In case of service **distribution-group**, the message waiting information is transported to any valid destination of the distribution-group.
- In case of service **hunt-group**, the message waiting information is transported only to the first destination of the hunt group.
- In **any other of the named services**, the message waiting information is routed transparently through.

If the message waiting information is routed to a destination service or interface that is not explicitly listed here, the information will be dropped.

Show Location-Service

This command displays the location-service information of all the configured location-services and their identities. It is possible to specify the name of which location-service you want to display and also a specific identity.

Mode: Interface SIP

Step	Command	Purpose
1	**device>show location-service [name] [identity]**	Display the information of location-services and Identities.

Configuration Examples

In this configuration example, inheritance is used.

Example:

```
location-service PATTON
  domain patton.com

  identity-group REGISTER

    authentication outbound
      authenticate 1 authentication-service AUTH_PATTON username john
    registration outbound
      registrar sip.domain.com
      lifetime 600
      register auto

  identity 300 inherits REGISTER
identity 400 inherits REGISTER
```

Exactly the same can be configured without inheritance. All inherited parameters can be configured in the identity itself. Inheritance is useful if multiple identities share the same configuration.

Example:

```
location-service PATTON
  domain patton.com

  identity 300

    authentication outbound
      authenticate 1 authentication-service AUTH_PATTON username john

    registration outbound
      registrar sip.domain.com
      lifetime 600
      register auto

  identity 400

    authentication outbound
      authenticate 1 authentication-service AUTH_PATTON username john

    registration outbound
      registrar sip.domain.com
      lifetime 600
      register auto
```

Chapter 45 Call Router Configuration

Chapter contents

Introduction

This chapter provides an overview of call router tables, mapping tables and call services and describes the tasks involved in configuring the call router in Trinity. This chapter includes the following sections:

- Call router configuration task list
- Call router configuration tasks
- Examples

There are two options for deciding where an incoming call on a CS interface is forwarded to:

- Basic interface call routing: Basic interface routing can be configured directly on the CS interfaces. It's also called *direct call routing*.
- Advanced call routing: More complex call forwarding decisions can be configured in the call router.

The call router is a very efficient and flexible tool for routing calls between CS interfaces. Based on a set of routing criteria, the call router determines the destination (interface) for every incoming call. The forwarding decisions and features are based on a set of routing tables, mapping-tables and call services.

Each routing table is responsible for a specific routing criterion such as the called party number or the bearer capability of the call. Multiple tables can be linked together to form a decision tree. The mapping tables can be used to modify call properties like the calling and called party numbers according to the network requirements. Call services can be used in the routing path to observe the call state and spawn other calls. The hunt-group service is an example for a call service. Figure 73 illustrates direct call and advanced call routing. In this chapter, advanced call routing is explained. For configuring direct call routing refer to chapter 45, "Call Router Configuration" on page 455.

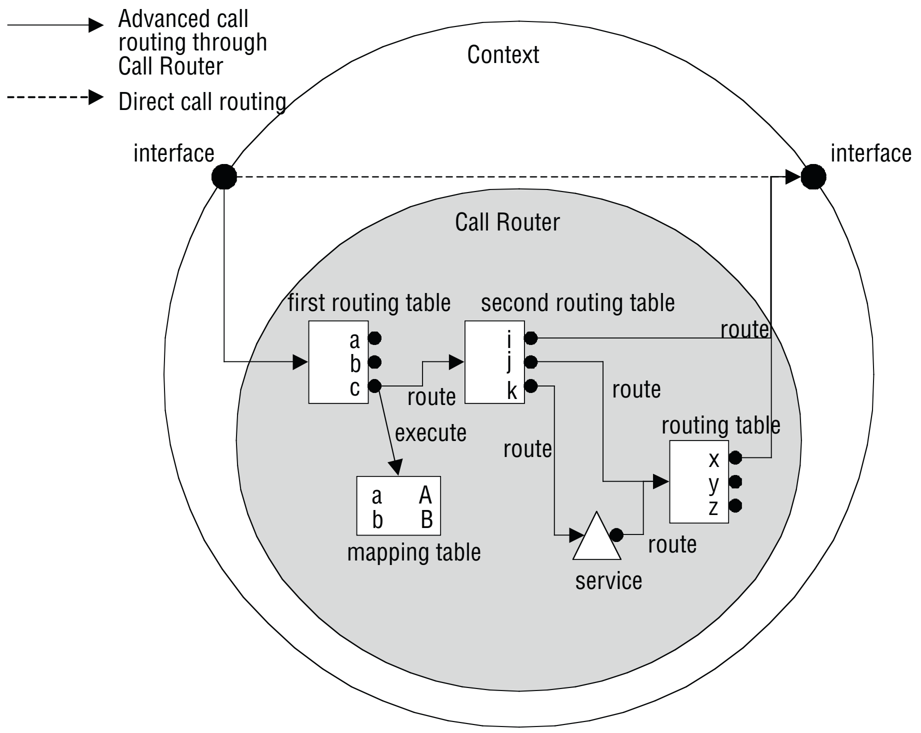


Figure 73. Direct call routing vs. advanced call routing

Due to the tree search algorithm implemented in the call router very large routing tables can be scanned very quickly with minimal impact on the call setup delay. The Trinity call router supports the following routing criteria:

- Calling party number (*calling-e164*); also called *source-Nr*, *A-Nr*, *MSN*, *DDI*, or *CLIP*
- Called party number (*called-e164*); also called *destination-Nr* or *B-Nr*
- Calling and called party number type
- Calling and called party number plan
- Calling and called party name, the display name
- Calling and called party IP address (for VoIP calls)
- Calling and called party URIs (for SIP calls)
- Presentation indicator; whether the number shall be presented to the other party
- Screening indicator; whether the number has been screened
- Information transfer capability; also called ISDN bearer capability or ISDN service
- Day of week; Monday–Sunday
- Time of day; *hour:minute:second*
- Date; *day.month.year*

IMPORTANT

The call router allows you to solve practically any call routing and call property manipulation requirement that you may have. The call router is very flexible in allowing the construction of decision trees based on linked routing tables. However you should take care not to use too many tables and an over-elaborate structure. The configuration may become large and difficult to manage. For complex configurations we recommend offline editing and configuration downloads.

Call Router Configuration Task List

To configure the call router, perform the tasks in the following sections and in the order as listed below.

- Map out the goals for the call router
- Enable advanced call routing on circuit interfaces
- Configure general call router behavior
- Configure call router tables
- Configure mapping tables and complex functions
- Configure call services
- Deleting routing tables and functions
- Activate the call router configuration
- Test the call router configuration

Map the Goals for the Call Router

There are many possible policies and factors that may influence the call router configuration. Some examples include:

- On-net off-net call routing
- ISDN service routing
- Carrier selection
- Service quality
- Fallback strategies
- Network and gateway selection

Other factors that must be taken into account include the following:

- Available number ranges (DDI, MSN, PISN)
- Potential restrictions imposed by neighboring equipment (Remote Gateways, PBXs) on the number length or range to be used.

The call router is able to accommodate almost every combination of these requirements through a customized configuration.

In order to keep this configuration compact we recommend that you first define the routing requirements and restrictions that apply to your installation. Then define the routing and mapping tables and the call services that you need to fulfill these requirements. Finally define the decision tree (i.e. the sequence in which the tables are linked together). In this step you may realize that you need multiple tables of the same type to achieve your goals. On the other hand an alternative sequence may help you to reduce the number of tables or the size of each table while still achieving the set goal. Only when you are happy with the planned tables, functions and sequence should you start configuration.

Enable Advanced Call Routing on Circuit Interfaces

To activate the advanced call routing the first routing table in the CS interface has to be specified as you can see in figure 73. Make sure the route parameter on the CS interface is configured in order to forward the incoming calls to the first table of the call router.

Procedure: To enable advanced call routing on a circuit interface

Mode: Context CS

Step	Command	Purpose
1	node(ctx-cs)[switch]#**interface** *if-type if-name*	Change to Interface Configuration Mode to specify the entry table in the interface
2	*node*(if-*type*)[*if-name*]#**route call dest-interface** *interface-name* *OR* *node*(if-*type*)[*if-name*]#**route call dest-table** *table-name* *OR* *node*(if-*type*)[*if-name*]#**route call dest-service** *service-name*	Use these commands to route a call (1) directly to an interface specified with the *interface-name* parameter; (2) to the call router, using the *table-name* table as first routing table; (3) directly to a service specified with the *service-name* parameter.

Configure General Call Router Behavior

Configure address completion timeout

A call that is routed through a called party number routing table possibly has a called party number that is too short for a routing decision to be made. In this case the call router waits for additional digits being entered by the calling user. When the user does not enter additional digits during the address completion timeout, the call router drops the call. You can configure the address completion timeout following the procedure below. The default address completion timeout is 12 seconds and is restarted after each digit that is sent during overlap dialing.

Note The address completion timeout is active when the call router waits for mandatory digits before being able to complete call routing. Contrary to this, the digit collection timeout, described below, waits for additional optional digits.

Procedure: To configure the address completion timeout

Mode: Context CS

Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]# **address-completion timeout** *timeout*	Configures the address completion timeout by specifying the *timeout* in seconds. If not configured, the default address completion timeout is 12 seconds.

Example: Configure address completion timeout

```
node[switch]#address-completion timeout 20
```

Configures the address completion timeout to 20 seconds. A call with an incomplete called party number is dropped 20 seconds after receiving the last called party address update (overlap dialed digit).

Configure default digit collection timeout and terminating character

You can enter called party routing table entries that specify an incomplete number (extended by the T-indicator; refer to section "Called Party Number Routing Table" on page 466). When the call router selects such an entry, the call router waits for additional digits and starts the digit collection timeout. When the digit collection timeout elapses or when the user enters the terminating character, the call is placed to the destination specified with the routing table entry.

The digit collection timeout has a default value of 5 seconds, the terminating character is the pound character (#) by default.

Note The digit completion timeout is active when the call router waits for optional digits of a called party number before placing the call to the selected destination. Contrary to this, the address completion timeout, described above, waits for mandatory digits.

Procedure: To configure the digit collection timeout and terminating character

Mode: Context CS

Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]# **digit-collection timeout** *timeout*	Configures the digit collection timeout by specifying the *timeout* in seconds. If not configured, the default digit collection timeout is 5 seconds.
2	*node*(ctx-cs)[*switch*]# **digit-collection terminating-char** *char* *OR* **no digit-collection terminating-char**	Configures the digit collection terminating character by specifying the *char*. This can be any character out of **0123456789*#**. The default terminating character is the pound (#). You can also use this command in the no-form to disable that the user can stop digit collection by a terminating character.

Example: Configure address completion timeout

```
node[switch]#digit-collection timeout 3
```

```
node[switch]#digit-collection terminating-char *
```

Configures the digit collection timeout to 3s. The digit-collection timeout can be stopped by the user entering the asterisk (*) character.

Configure Number Prefix for ISDN Number Types

The called and calling party numbers in an ISDN signaling message are of a defined number type; national, international or unknown. Depending on where the message originates (PSTN, mobile network, PBX) this number type may differ. Table 27 illustrates the three number types.

Table 27. ISDN number types

Type	Format Description	Example
unknown	as dialed with all leading zeros or other prefix numbers	0041 99 888xxxx
national	(area code) (local extension number)	99 888xxxx
international	(country code) (area code) (local extension number)	41 99 888xxxx

The missing prefix in the national and international number types can complicate the call router configuration, so Trinity offers the possibility to expand these numbers before entering the first call router table.

Note The configured prefix is not removed at the exit of the call router (i.e. when a destination interface is found), but the number has the number type *unknown*.

Procedure: To configure number prefix

Mode: Context CS

Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]#**national-prefix** *prefix*	Adds *prefix* to all E.164 numbers of type national before entering the call router.
2	*node*(ctx-cs)[*switch*]#**international-prefix** *prefix*	Adds *prefix* to all E.164 numbers of type international before entering the call router.

Example: Configure number prefix

```
node[switch]#national-prefix 0041

Input: 99888xxxx Result: 004199888xxxx

node[switch]#international-prefix 00

Input: 4199888xxxx Result: 004199888xxxx
```

Configure Call Routing Tables

Routing tables are identified by names that can be any arbitrary string. For ease of identification the table type is typically used as part of the name.

Call router tables are created by entering the **routing-table** command, which also brings you into the routing table configuration mode. There you can add, modify or delete entries of the routing tables. Refer to the individual table types detailed below on how to configure table entries.

Note The sequence of the lines is not important. The call router creates a search tree out of the table lines to ensure optimal search speed.

Note To remove a specific entry of a table, enter the table configuration mode and use the no-form on a previously entered entry. To remove a whole table, use the **no** form of the **table mode** command.

Create a routing table

A routing table forwards the call to another table, interface or service based on a specific call property like the called party number or the current date. The call router provides a number of different routing table types. A routing table looks like the following:

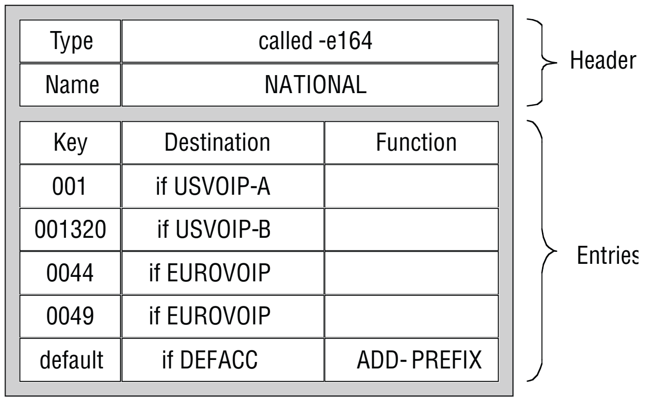


Figure 74. Routing table outline

Each table contains a header and one or more entries. The header declares the type of the routing table as well as its name.

The name of the routing table is unique inside the context and serves as identifier for referencing the table from other tables or interfaces. The routing table type specifies which call property the table shall examine. Table 28 lists the call properties that can be used as a routing table type.

Table 28. Routing table types

Type	Description
called-e164	Route calls based on the called party E.164 number. Entries of called-e164 tables can use wildcards to summarize routes. Digit collection can be configured on a per-entry basis.
calling-e164	Route calls based on the calling party E.164 number. Entries of calling-e164 tables can use wildcards to summarize routes.

Table 28. Routing table types (Continued)

Type	Description
calling-second-e164	Route calls based on the second calling party E.164 number. For having a second calling party E.164 number available, mappings from call-properties in the incoming interface of the calls are needed. The same rules as for the normal calling party E.164 number apply.
calling-second-type-of-number	Route calls based on the second calling party number type. For having a second calling party number type available, mappings from call-properties in the incoming interface of the calls are needed. The same rules as for the normal calling party number type apply.
calling-second-numbering-plan	Route calls based on the second calling party numbering plan. For having a second calling party numbering plan available, mappings from call-properties in the incoming interface of the calls are needed. The same rules as for the normal calling party numbering plan apply.
calling-second-pi	Route calls based on the second presentation indicator. For having a second presentation indicator available, mappings from call-properties in the incoming interface of the calls are needed. The same rules as for the normal presentation indicator apply.
calling-second-si	Route calls based on the second screening indicator. For having a second screening indicator available, mappings from call-properties in the incoming interface of the calls are needed. The same rules as for the normal screening indicator apply.
called-type-of-number	Route calls based on the called party number type. ISDN distinguishes different type of numbers.
calling-type-of-number	Route calls based on the calling party number type. ISDN distinguishes different type of numbers.
called-numbering-plan	Route calls based on the called party numbering plan. ISDN distinguishes different numbering plans.
calling-numbering-plan	Route calls based on the calling party numbering plan. ISDN distinguishes different numbering plans.
called-name	Route calls based on the display name of the called party.
calling-name	Route calls based on the display name of the calling party.
called-ip	Route calls based on the signaling IP address of the destination VoIP peer.
calling-ip	Route calls based on the signaling IP address of the origination VoIP peer.
called-uri	Route calls based on the URI of the destination VoIP peer (for SIP calls: the To-URI).
calling-uri	Route calls based on the URI of the origination VoIP peer (for SIP calls: the From-URI).
calling-pi	Route calls based on the presentation indicator.
calling-si	Route calls based on the screening indicator.
itc	Route calls based on the information transfer capability (bearer capability) to distinguish speech from data calls.
time	Route calls based on the current time of day. .
date	Route calls based on the current date.

Table 28. Routing table types (Continued)

Type	Description
day-of-week	Route calls based on the current day of week.

Besides the header (name and type) a routing table contains multiple entries. Each entry specifies a specific value of the routing table type and a destination interface and an optional function. When a call arrives at a routing table, the following procedure is applied:

1. Examine the call property as specified with the routing table type.
2. Select the best matching entry. This means that the key of each of the entries is compared to the call property and the entry that matches best is chosen.
3. Execute the entry. This means executing the referenced function of the entry if specified, and routing the call to the specified destination interface, table or service.

Procedure: To create a routing table and add entries

Mode: Context CS

Step	Command	Purpose
1	node(ctx-cs)[switch]#**routing-table** *table-type table-name*	Create a routing table *table-name* of the specified *table-type*. This enters the table mode where entries can be added or removed. To enter a previously created table from the context CS mode, you may leave away the *table-type*.
2	node(rt-tab)[table-name]#**route** *key* **dest-interface** *if-name function* OR node(rt-tab)[table-name]#**route** *key* **dest-table** *table-name function* OR node(rt-tab)[table-name]#**route** *key* **dest-service** *service-name function*	Add an entry to the routing table for destination interface *if-name*, destination table *table-name* or destination service *service-name*. Optionally you can specify a *function* (mapping-table or complex function) that shall be executed before the call is routed to the destination interface, table or service. The format of the *key* depends on the type of table. The next sections explain key formats for the different table types.
3		Repeat step 2 to add lines for additional table entries.

Example: Called party number routing table

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table called-e164 NATIONAL
node(rt-tab)[switch.NATIONAL]#route 001 dest-interface USVOIP-A
node(rt-tab)[switch.NATIONAL]#route 001320 dest-interface USVOIP-B
node(rt-tab)[switch.NATIONAL]#route 0044 dest-interface EUROVOIP
node(rt-tab)[switch.NATIONAL]#route 0049 dest-interface EUROVOIP
node(rt-tab)[switch.NATIONAL]#route default dest-interface DEFACC ADD-PREFIX
```

Called Party Number Routing Table

The called party number (called-e164) table is used to route calls based on the called party E.164 number in the call set-up message. The call router scans the table to find the longest matching *key* starting with the **first** digit.

Regular expressions

The *key* of an entry can be either a complete number or a partial number with wildcard digits, represented by a period (.) character. Each (.) represents a wildcard for an individual digit. For example, if the *key* is defined as *888 . . . .*, then any called party number beginning with 888, plus at least four additional digits matches this entry.

In addition to the period (.), there are several other symbols that can be used as wildcard characters in the *key*. These symbols provide additional flexibility in designing call routing and decrease the need for multiple entries in configuring number ranges.

The following table shows the wildcard characters that are supported:

Table 29. Wildcard symbols used as keys in E.164 tables (calling-e164, called-e164)

Symbol	Description
.	Indicates a single-digit placeholder. For example, 888 . . . . matches any dialed number beginning with 888, plus at least four additional digits. Note that the key only specifies the prefix. Thus the number may be longer, but also matches.
[ ]	Indicates a range of digits. A consecutive range is indicated with a hyphen (-); e.g. [5-7]. A non-consecutive range is indicated without a delimiter. For example, [58]. Both can be used in combination; e.g. [5-79], which is the same as [5679]. A (^) symbol may be placed right after the opening bracket to indicate that the specified range is an exclude list. For example, [^01] specifies the same range as [2-9]. **Note:** Only single-digit ranges are supported. You cannot specify the range of numbers between 99 and 102 using [99-102].
( )	Indicates a pattern. For example, 888(2525). It is used in conjunction with the symbol (?), (%) or (+) or when replacing a number in a mapping table.
?	Indicates that the preceding digit or pattern occurred zero or one time. Enter Ctrl-V before entering (?) from your keyboard, since the CLI normally uses the question mark to display help texts.
%	Indicates that the preceding digit or pattern occurred zero or more times. This functions the same as the asterisk (*) used in regular expression. Here the percent (%) symbol is used to be able to handle the asterisk (*) as part of a dialed number.
+	Indicates that the preceding digit or pattern occurred one or more times.
T	Enables digit-collection for this entry. The call router pauses to collect additional dialed digits. The default digit collection timeout is 5 seconds. The collection can be aborted pressing the pound (#) key. **Note:** The terminating character is used only to terminate the timeout and is therefore removed from the dialed number. **Note:** The timeout (T) symbol is only allowed for Called Party Number table entries, while all other wildcards are also allowed for Calling Party Number tables.

The next table shows some examples of how these wildcard symbols are applied to the *key* of a table entry:

Table 30. Examples of using wildcard symbols

Expression	Description
88825.+	88825, followed by one or more wildcard digits. This expression implies that the number must contain at least 6 digits starting with 88825; for example, 888251, 8882512 or 888251234567890
88825.%	88825, followed by zero or more wildcard digits. This expression implies that the string must contain at least 88825; for example, 88825, 888256, 8882567. **Note:** The ".%" expression postfix can be left away, because the expression is always compared as prefix to the dialed number. Thus each expression automatically contains a ".%" postfix.
88825+	8882, followed by 5 repeated one ore more times; for example, 88825, 888255, or 8882555555555
888(25)+	888, followed by 25 repeated one or more times; for example, 88825, 8882525 or 8882525252525
0?111	An optional 0, followed by 111; for example, 0111, 111, 11123456789
8882[56]...	8882 followed by 5 or 6, plus at least three more wildcard digits.
.%45$	Any number that has a postfix of 45; for example, 45, 045, 0041998882545. **Note:** The dollar sign ($) at the end is used to disable prefix matching.

In addition to wildcard characters, the following characters can also be used in the key of the table entry:

- Asterisk (*) and pound sign (#) – These characters can be used anywhere in the key like other digits, for example, they can be used as the leading character (e.g. *21), which is handled like normally dialed number.
- Dollar sign ($) – Disables prefix matching. Must be used at the end of the dial string.

Digit collection

Fixed-length dialing plans, in which all the numbers have fixed length, are sufficient for most voice networks, because the telephone number strings are of known lengths. Some voice networks, however, require variable-length dial plans, particularly for international calls, which use telephone numbers of different length. Furthermore some voice networks do not support overlap dialing. In this case the call-router must collect the digits before placing a call to that network with the complete number.

If you enter the timeout T-indicator at the end of the key in a Called-Party Number table, the call router accepts a fixed-length number and then waits for additional dialed digits. The timeout character must be an uppercase T. The following example shows how the T-indicator is set to allow variable-length numbers:

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table called-e164 collect
node(rt-tab)[swtich.collect]#route 0041T dest-interface CHVoIP-A
```

In the example above, the call router accepts the digits 0041, and then waits for an unspecified number of additional digits as long as the interdigit timeout has not expired. When the interdigit timeout expires, the router places the call.

The default value for the interdigit timeout is 5 seconds and can be configured using the **digit-collection timeout** command in the context CS configuration mode. You may want to override this default timeout for a specific entry. Just place the timeout in seconds after the T-indicator; e.g. T3 to set the inter digit timeout to 3 seconds for that entry.

The user may press the pound (#) as terminating character to immediately place the call. If the pound (#) character is entered while the router is waiting for additional digits, the pound (#) character is treated as a terminator; it is not treated as part of the dialed number sent across the network. But if the pound (#) character is entered before the router begins waiting for additional digits (meaning that the pound (#) is entered as part of the fixed-length key), then the pound (#) character is treated as a dialed digit.

For example, if the key is configured as 888 . . . . T, then the entire dialed string of 888#2525 is collected, but if the dialed string is 888#252#5, the #5 at the end of the dialed number is not collected, because the final pound (#) character is treated as terminator. You can change the default terminating character using the **digit-collection terminating-char** command in the context CS configuration mode. You may want to override this default terminating character for a specific entry. Just place the character after the timeout and a comma; for example, T3,* to set the terminating character to asterisk (*).

Digit collection variants

There are three different ways how a called party routing-table can be used to perform address completion or digit collection. Consider the following examples:

```
routing-table called-e164 TAB-PREFIX
  route 099 dest-interface IF-OUT

routing-table called-e164 TAB-COMPLETE
  route 099.... dest-interface IF-OUT

routing-table called-e164 TAB-COLLECT
  route 099T dest-interface IF-OUT
```

Now assume someone picks up a phone and dials a number using overlap dialing. After picking up the phone, an empty called party number is offered to the routing tables. All three routing tables require the called party number to contain at least the prefix 099. Thus the number is incomplete and the call router waits for additional digits being entered. Now the user presses the digit one (1). The resulting called party number is 0991. The call router is again asked for routing the call to a destination. The TAB-PREFIX table performs a prefix match with its only entry and finds out that the number is long enough, so TAB-PREFIX immediately routes the call to the destination interface IF-OUT. Unlike the first table, TAB-COMPLETE needs at least three more digits. Thus the address is not complete yet and the call router waits for more digits. TAB-COLLECT has enough digits but uses the T-indicator to perform digit-collection. The call router waits for the digit-collection timeout and then places the call to the destination interface IF-OUT.

Note There is a difference between the address completion and the digit collection timeout. The address completion timeout is active when a route is incomplete, e.g. when the dialed number of 0991 is tried to match to the entry *099....*. In this case, the call router cannot forward the call to the destination unless the user enters three more digits. Thus the address completion timeout is active when the call router waits for mandatory digits. The address completion timeout can be configured using the **address-completion timeout** command in the context CS mode. If the user does not enter more digits, the address completion timeout elapses and the call is dropped. The digit collection timeout is active when a route is complete but a T-indicator is specified on the selected route, e.g. when the dialed number of 0991 is tried to match the entry *099T*. In this case the call router waits for some period of time for the user to enter additional optional digits. The digit collection timeout can be configured using the **digit-collection timeout** command in the context CS mode. If the user does not enter more digits, the digit collection timeout elapses and the call is forwarded to the destination interface.

Example: Simple called party number routing table

The following table routes call based on the called party number. An internal number starting with 5 and containing at least 3 digits is routed to the interface that is connected to the local PBX. An international call to the US is routed to the VoIP interface USVOIP. All other calls (local, national and international) are routed to the interface that is connected to the PSTN.

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table called-e164 DIST
node(rt-tab)[switch.DIST]#route 5.. dest-interface PBX
node(rt-tab)[swtich.DIST]#route 001T dest-interface
USVOIP node(rt-tab)[DIST]#route default dest-interface PSTN
```

Example: Digit collection of any number

If you want to route calls from interface A directly to interface B, wanting to collect dialed digits, you have to route calls from interface A to a routing table like the one shown in this example. This table does not require the dialed number to be of any format or length but waits for arbitrary number of digits that can be entered using overlap dialing. This allows the user to enter any number to reach the destination interface.

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table called-e164 COLLECT
node(rt-tab)[switch.COLLECT]#route T dest-interface OUT
```

Example: Called party number routing table explained

Regular expressions are very powerful for E.164 lookups. A routing table always tries to find the table entry with the best matching prefix. Determining the best match is often not a very simple process. There are several rules that define the match quality of a rule for an entered number:

1. A longer rule matches better than a shorter one.
2. An explicitly specified digit matches better than a wildcard.
3. A wildcard that include less possible digits matches better that a wildcard that include more possible digits.

Consider the following routing table:

```
routing-table called-e164 test
  route 1       dest-interface IF1entry #1
  route 1[0-4]  dest-interface IF2entry #2
  route 11      dest-interface IF3entry #3
  route 111T    dest-interface IF4entry #4
  route default dest-interface IF5entry #5
```

Note The numbers that are normally dialed are longer than the prefixes listed in the table test. For example, if the numbering plan is defined using five digits, a user normally dials a number like 12345 to reach a destination. Anyway the lookup result must be the same for en-bloc and for overlap dialing. This example shows how the table is looked up after each overlap-dialed digit and how a destination is finally found. This selected destination is the same as if the user dialed the number en-bloc.

The following table contains a list of overlap-dialed number examples that use the routing table above for a lookup:

Dialed Number	Selected Entry	Description
empty (after picking up the phone without dialing a number beforehand)	—	The number is incomplete for entries #1-#4, which all want at least know whether the number starts with a 1. Thus no entry is selected and the call router waits for additional mandatory digits or drops the call after 12 seconds (address-completion timeout).
1	—	No entry is selected. Though the dialed number matches entry #1 there are other entries that are still incomplete (the entered number is a prefix of the entry). In this state the call router waits for additional mandatory digits or drops the call after 12 seconds (address-completion timeout).
2	#5	No entry matches, so the default entry is selected; the call is placed immediately.
11	—	No entry is selected. Though the dialed number completely matches entry #1, #2 and #3, entry #4 is still incomplete. The call router waits for additional mandatory digits or drops the call after 12 seconds (address-completion timeout).
12	#2	Entry #1 and #2 match the dialed number of 12, but entry #2 matches better because the expression is more precise (longer) than entry #1. Thus the call is immediately routed to interface IF2.
19	#1	Entry #1 is the only that matches. The call is immediately placed.
111	#4	All entries match the dialed number of 111, but entry #4 matches best because the expression is more precise (longer) than entry #1-#3. Entry #4 is selected but the call is not placed immediately because the entry contains the T-indicator. The router waits for additional digits and then places call to interface IF4 when the digit-collection timeout elapses. Note: If the user enters an additional digit during digit-collection on a T-indicator, the router must not change the destination entry anymore.
112	#3	Entry #1, #2 and #3 match the dialed number of 112. Entry #1 has only an expression of one digit while entry #2 and #3 have an expression that specify two digits. Entry #3 matches better than entry #2 because entry #3 explicitly specifies the digits while entry #2 contains a wildcard for the second digit. Thus entry #3 is selected and the call is placed immediately to interface IF3.
121	#2	Entry #1 and #2 match the dialed number of 121, but entry #2 matches better. The call is immediately placed to IF2.

Dialed Number	Selected Entry	Description
191	#1	Only entry #1 matches the dialed number of 191. Thus the call is routed immediately to interface IF1.
1111	#4	The lookup procedure is the same as for dialed number 111. The call router waits for additional digits and places the call after the digit-collection timeout to interface IF4.
1111#	#4	Same as for 1111, but the pound (#) terminates the digit collection; the call is immediately placed to interface IF4.

Calling party number routing table

The calling party number (calling-e164) table is used to route calls based on the calling-e164 in the call setup message. This number in general corresponds to the extension number of a PBX or MSN of an ISDN terminal. The table can be used to route calls from extensions, which have particular call routing requirements (i.e. Terminals which require non VoIP capable ISDN services). The call router looks for the longest match starting with the first digit of the calling party number.

Note The calling party number is sometimes inserted or modified by a PBX. Sometimes there is no calling party number at all. This all depends on the equipment you connect to the device.

Note The T-indicator cannot be used in calling party number tables. (Overlap dialing only makes sense for called party numbers).

Example: Calling party number routing table

This example shows how to create a calling party number routing table that routes calls based on the last three digits of the calling party number (the extension part). The key *.%52[35]$* means that every number that starts with any digit (.) appearing zero or more times (%) followed by 52 and a 3 or 5 matches the entry. For example, the following calling party numbers match the first entry: 0998882523 or 0998882525 or simply 523 or 525.

Note This table does not contain a default entry. All calls where the calling party number does not match to one of the entries are dropped.

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table calling-e164 EXTS
node(rt-tab)[switch.EXTS]#route .%52[35]$ dest-interface breakout
node(rt-tab)[switch.EXTS]#route .%572 dest-interface DefAcc
```

Number Type Routing Table

The calling or called party number type (calling-type-of-number or called-type-of-number) table is used to route calls based on the calling or called party type of number field in the ISDN setup message. This can be used, for example, to differentiate between national and international calls.

Note When you specified a national or international prefix using the commands national-prefix or international-prefix respectively. in the context CS configuration mode, the calling or called party number is extended with the specified prefix and the type-of-number is set to unknown in the incoming interface. Thus an international number can enter the call-router as unknown number.

Note This property is only set by incoming ISDN calls. A call that arrives the call router from a FXS or SIP interface has a number type of Unknown as those interfaces do not support the number type property.

The call router can route calls according to the following number types. These values beside default can be used for the key parameter to create a routing table entry:

- unknown—Unknown number type. This is the default value for calls that arrive through an interface that does not support the number type property.
- international—International number; the number does not include prefix or escape digits.
- national—National number; the number does not include prefix or escape digits.
- network-specific—Network specific number, used to indicate administration or service number specific to the serving network, e.g. used to access an operator.
- subscriber—Subscriber number; the number does not include prefix or escape digits.
- abbreviated—Abbreviated number.

Example: Calling type-of-number routing table

The following example routes calls with an international calling party number to the next table TAB-INCOMING-INT, calls with a national calling party number to the next table TAB-INCOMING-NAT and all other calls to the next table TAB-INCOMING-UNKNOWN.

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table calling-type-of-number TON
node(rt-tab)[TON]#route international dest-table TAB-INCOMING-INT
node(rt-tab)[TON]#route national dest-table TAB-INCOMING-NAT
node(rt-tab)[TON]#route default dest-table TAB-INCOMING-UNKNOWN
```

Numbering Plan Routing Table

The calling party numbering plan or called party numbering plan (calling-number-plan or called-numbering-plan) table is used to route calls based on the calling or called party numbering plan field in the ISDN setup message. This can be used to differentiate calls between numbers of an ISDN, data, telex, national standard or private numbering plan.

Note This call property is only set by incoming ISDN calls. A call that arrives the call router from a FXS or SIP interface has a numbering plan of unknown.

The call router can route calls according to the following numbering plans. These values beside default can be used for the key parameter to create a routing table entry:

- unknown—Unknown numbering plan. This is the default value for calls that arrive through an interface that does not support the numbering plan property.
- isdn-telephony—ISDN/Telephony numbering plan according to CCITT Recommendation E.164/E.163).

- data—Data numbering plan according to CCITT Recommendation X.121.
- telex—Telex numbering plan according to CCITT Recommendation F.69.
- national-standard—Numbering plan according to a national standard.
- private—Any private numbering plan.

Example: Calling numbering-plan routing table

The following example shows how to create a routing table with name NP that routes calls based on the calling part numbering plan. Calls with calling party numbers formed according to the ISDN/Telephony numbering plan are routed to the next table *TAB-INCOMING-ISDN*, calls with calling party numbers formed according to the data numbering plan to the next table *TAB-INCOMING-DATA* and all other calls to the next table *TAB-INCOMING-UNKNOWN*.

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table calling-numbering-plan NP
node(rt-tab)[NP]#route isdn-telephony dest-table TAB-INCOMING-ISDN
node(rt-tab)[NP]#route data dest-table TAB-INCOMING-DATA
node(rt-tab)[NP]#route default dest-table TAB-INCOMING-UNKNOWN
```

Name Routing Table

The calling display name or called display name (calling-name or called-name) table is used to route calls based on the human-readable name of the calling or called party. The key you specify in a routing table entry must be identical to the display name of the calling or called party for the entry to match.

Note Incoming SIP calls use this call property to store the display name part of the SIP URI. Other interfaces set the display name to the empty string.

Example: Calling display name routing table

This example routes calls based on the display name part of the From-URI of an incoming SIP call.

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table calling-name name
node(rt-tab)[switch.device]#route “John Smith” dest-table TAB-FROM-JOHN
node(rt-tab)[switch.device]#route Administrator dest-table TAB-FROM-ADMIN
node(rt-tab)[switch.device]#route default dest-table TAB-FROM-UNKNOWN
```

IP Address Routing Table

The calling party IP address (calling-ip) table is used to route calls based on the signaling IP address of the originating VoIP peer. The called party IP address (called-ip) table is used to route calls based on the called IP address property. The called IP address of incoming calls is set to 0.0.0.0 unless modified by a previous mapping table in the routing path. Thus the called IP address table is of limited use only.

You may specify a whole subnet with the key parameter of the routing table entry. The format of the key parameter is IPaddress[/mask-size]; the mask size may be omitted.

Note Incoming SIP calls use the calling party IP address property to store the IP address of the remote SIP user agent, respectively. Other interfaces like ISDN or FXS set the IP address to 0.0.0.0.

Example: Calling IP address routing table

The following example routes all calls from a remote SIP user agent in the LAN to the next table *TAB-FROM-LAN* and all other calls to *TAB-FROM-WAN*.

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table calling-ip IP
node(rt-tab)[switch.IP]#route 172.16.32.0/24 dest-table TAB-FROM-LAN
node(rt-tab)[switch.IP]#route default dest-table TAB-FROM-WAN
```

URI Routing Table

The calling party URI (*calling-uri*) table is used to route calls based on the URI of the originating VoIP peer, e.g. the From-URI of the incoming SIP call. The called party URI (*called-uri*) table is used to route calls based on the *To-URI*. The called URI of incoming calls is not set unless modified by a previous mapping table in the routing path.

You can use regular expressions to specify the parts of an URI that must match in order to route the call to a specified destination.

The following example shows how to create a routing table to route all SIP calls from John Smith to the next table *TAB-FROM-JOHN* while all other calls are routed to the next table *TAB-FROM-UNKNOWN*.

Mode: Context CS

Step	Command	Purpose
1	**node(ctx-cs)[switch]#routing-table calling-uri** name	Creates a routing-table that examines the From-URI of an incoming SIP call
2	**node(rt-tab)[***switch.device***]#route sip:john\.smith@.% dest-table TAB-FROM-JOHN**	Routes all SIP calls from john.smith@<anywhere> to the table TAB-FROM-JOHN. Note that the dot (.) between john and smith must be escaped with a backslash (\), because the dot (.) means 'any character' in a regular expression.
3	**node(rt-tab)[***name***]#route default dest-table TAB-FROM-UNKNOWN**	Routes all other SIP calls to the table TAB-FROM-UNKNOWN

Presentation Indicator Routing Table

The presentation indicator (calling-pi) table is used to route calls based on the presentation indicator of the calling party number. A user that doesn't want its number being displays sets the presentation indicator to restricted. There is no presentation indicator on the called party number. Thus you cannot create a called-pi table.

Note Incoming ISDN calls set the presentation indicator according to the received ISDN Setup message. SIP interfaces set the presentation indicator to allowed.

The call router can route calls according to the following presentation indicators. These values beside default can be used for the key parameter to create a routing table entry:

- allowed—Presentation of the calling party number is allowed. This is the default value for calls that arrive through an interface that does not support presentation indicators.

- restricted—Presentation of the calling party number is restricted.
- interworking—The calling party number is not available due to interworking.

Note At the originating user-network interface, the presentation indicator is used for indicating the intention of the calling user for the presentation of the calling party number to the called user. You possibly want to remove the calling party number when the user set the presentation indicator to restricted. To achieve this, route restricted calls to a mapping table that sets the calling-e164 to the empty string (“”) as it is shown in the example below.

Example: Presentation indicator routing table

This example uses a pseudo routing table that just forwards all calls to the interface *IF-OUT* but first executes the mapping table *NO-CNPN*. This mapping table examines the presentation indicator and modifies the calling party number. If the presentation indicator is restricted, the calling party number is cleared. For all other presentation indicator values, the calling party number is not modified.

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table calling-pi PI
node(rt-tab)[switch.PI]#route default dest-interface IF-OUT NO-CNPN
node(rt-tab)[switch.PI]#exit
node(rt-tab)[switch]#mapping-table calling-pi to calling-e164 NO-CNPN
node(map-tab)[switch.NO-CNPN]#map restricted to “”
```

Screening Indicator Routing Table

The screening indicator (*calling-si*) table is used to route calls based on the screening indicator of the calling party number. A network validates the calling party’s number and puts the validation result to the screening indicator. This allows a network to transparently pass the calling party number set by the calling user, but tell the called user whether or not the number selected by the calling party really belongs to him or her.

Note Incoming ISDN calls set the screening indicator according to the received ISDN Setup message. Other interfaces set the screening indicator to user-not-screened.

The call router can route calls according to the following screening indicators. These values beside default can be used for the key parameter to create a routing table entry:

- user-not-screened—The calling party number is provided by the user but not screened by the network. Thus the calling party possibly send a number that is not owned by the calling party. (This is the default value for calls that arrive through an interface that does not support presentation indicators).
- user-passed—The calling party number is provided by the user, screened by the network and really belongs to the calling party.
- user-failed—The calling party number is provided by the user, screened by the network and does not belong to the calling party.
- network—The calling party number, because it is provided by the network, can be trusted.

Note You possibly want to remove the calling party number when the calling party number is not screened or screening failed. To achieve this, route these calls to a mapping table that sets the calling-e164 to the empty string (“”). If you want to drop calls when the calling party number is not screened or screening failed, use the routing destination none.

Example: Screening indicator routing table

A call that is routed to this table is examined for the screening indicator of the calling party number. If the calling party provided a number that was not accepted by the network (user-failed), we drop the call. Else we route the call to the interface *IF-OUT* but first execute the mapping table *NO-CNPN*. This mapping table again examines the screening indicator. If the user provided a number that was not screened by the network, the table sets the calling party number to an empty string. For all other screening indicator values, the calling party is not modified.

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table calling-si SI
node(rt-tab)[switch.SI]#route user-failed none
node(rt-tab)[switch.SI]#route default dest-interface IF-OUT NO-CNPN
node(rt-tab)[switch.SPI]#exit
node(rt-tab)[switch]#mapping-table calling-si to calling-e164 NO-CNPN
node(map-tab)[switch.NO-CNPN]#map user-not-screened to “”
```

Information Transfer Capability Routing Table

The information transfer capability (*itc*) table is used to route calls based on the information transfer capability field of the bearer capability information element in the ISDN setup message. This can be used to differentiate between ISDN data services and ISDN speech connections.

Note Terminals connected to analog extensions (e.g. of a PBX) do not supply information transfer capability values in their call set-up, so it is up to the configuration of the analog port on the Terminal Adapter, NT or PBX to insert this value. The configuration of this value is however often omitted or wrong. The ITC value may therefore not be a reliable indication to differentiate between analogue speech, audio or Fax Group 3 connections. Also, calls from SIP and FXS interfaces do not differentiate between bearer capabilities. They always set the information transfer capability property to *3.1kHz Audio*.

The call router can route calls according to the following information transfer capabilities. These values beside default can be used for the key parameter to create a routing table entry:

- speech—Voice terminals (Telephones)
- unrestricted-digital—Unrestricted digital information (64kBit/s)
- restricted-digital—Restricted digital information (64kBit/s)
- 3k1-audio—Transparent 3.1kHz audio channel. This is the default value set by interfaces that do not support the ITC property.
- 7k-audio—Transparent 7.1kHz audio channel
- video—Video conference terminals

Example: Information transfer capability routing table

The following example creates an itc routing table that routes all unrestricted digital calls to the interface *IF-ACCESS*, all 7kHz audio calls to the interface *IF-LOCAL-BREAKOUT*, all video calls to the interface *IF-ACCESS* and all other calls to the interface *IF-VOIP-CARRIER-A*.

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table itc ITC
node(rt-tab)[ITC]#route unrestricted-digital dest-interface IF-ACCESS
node(rt-tab)[ITC]#route 7k-audio dest-interface IF-LOCAL-BREAKOUT
node(rt-tab)[ITC]#route video dest-interface IF-ACCESS
node(rt-tab)[ITC]#route default dest-interface IF-VOIP-CARRIER-A
```

Call-router Support for Redirecting Number and Redirect Reason

The call.router can be used to manipulate and make routing decisions depending on the call-control redirecting number and redirect reason properties. The following command creates a call-router routing-table, which uses the redirecting number for routing decisions. The contents of the routing table are the same as for any other E.164 number based call-router routing table.

Mode: context cs

Step	Command	Purpose
1	[*name*] **(ctx-cs)**[router]**# routing-table calling-redir-e164 <table-name>**	Creates a redirecting number routing table.

The following command creates a redirect-reason call-router routing-table. Possible reason values for routing decisions are:

- cd: Call deflection
- cfb: Call forwarding on busy
- cfd: Call forwarding by called DTE
- cfnr: Call forwarding on no reply
- cfu: Call forwarding unconditional
- ooo: Called DTE out of order
- default: Any other unhandled case

Mode: context cs

Step	Command	Purpose
1	[*name*] **(ctx-cs)**[*router*]**# routing-table calling-redir-reason <table-name>**	Creates a redirect reason routing table.

Both the redirecting-number and the redirect-reason can also be used in any call-router mapping tables.

Time of Day Routing Table

The time table is used to route calls based upon the current system time during one day, i.e. an 24hr. period from midnight to midnight. Times are matched within the ranges defined in the time routing table.

The key parameter of the routing table entry has the format: *hh:mm:ss-hh:mm:ss*

The full range must be specified. The range must not cross a day boundary at midnight.

Example: Time of day routing table

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table time WORKDAY
node(rt-tab)[switch.WORKDAY]#route 08:00:00-16:59:59 dest-table TAB-BEST-QUAL
node(rt-tab)[switch.WORKDAY]#route 17:00:00-20:59:59 dest-interface IF-VOIP-A
node(rt-tab)[switch.WORKDAY]#route 21:00:00-23:59:59 dest-interface IF-VOIP-B
node(rt-tab)[switch.WORKDAY]#route 00:00:00-07:59:59 dest-interface IF-VOIP-B
```

Day of Week Routing Table

The day-of-week table is used to route calls according to the day of the week. The days are defined by the long lowercase names Monday; Tuesday; Wednesday; Thursday; Friday; Saturday; and Sunday. To configure weekday routing table entries use the following commands starting in the CS context configuration mode.

Example: Day of week routing table

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table day-of-week TAB-DAY
node(rt-tab)[switch.TAB-DAY]#route saturday dest-table TAB-LEAST-COST
node(rt-tab)[switch.TAB-DAY]#route sunday dest-table TAB-LEAST-COST
node(rt-tab)[switch.TAB-DAY]#route default dest-interface IF-VOIP
```

Date Routing Table

The date table is used to route calls according to the current system date. It can be used, for example, to represent holidays in the routing decision tree. The table matches exact dates or date ranges.

The key parameter of the routing table entry has the format: *dd:mm:yyyy-dd:mm:yyyy*

The full range must be specified.

Example: Date routing table

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table day-of-week HOLIDAY2001
node(rt-tab)[switch.HOLIDAY~]#route 01.01.2001-02.01.2001 dest-table TAB-HOL
node(rt-tab)[switch.HOLIDAY~]#route 05.01.2001-05.01.2001 dest-table TAB-HOL
node(rt-tab)[switch.HOLIDAY~]#route 24.12.2001-31.12.2001 dest-table TAB-HOL
node(rt-tab)[swicth.HOLIDAY~]#route default dest-interface IF-VOIP
```

Deleting Routing Tables

To remove individual routing tables you can use the **no** form of the **routing table** command. Alternatively you can remove specific entries of a routing table by entering the routing table configuration mode and use the **no** form of the **route** command.

Procedure: To delete an entry from a routing table

Mode: Context CS

Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]#**routing-table** *table-name*	Enter the routing table from which you want to remove an entry. **Note:** You do not have to enter the type of the table when just entering it. The type must only be specified when creating a table.
2	*node*(rt-tab)[*table-name*]#**no route** *key*	Remove the entry with the specified *key*.
3		Repeat Step 2 to remove additional entries.

Example: Remove entries from a routing table

The running-config shows the following table:

```
routing-table called-e164 MY-TABLE
  route 10 dest-interface IF1
  route 11 dest-interface IF2
  route 12 dest-interface IF3
  route default dest-interface IF4
```

To remove the first two entries from the table enter the following commands:

```
node(cfg)#context cs
node(ctx-cs)[switch]#routing-table MY-TABLE
node(rt-tab)[switch.MY-TABLE]#no route 10
node(rt-tab)[switch.MY-TABLE]#no route 11
```

The resulting running-config is:

```
routing-table called-e164 MY-TABLE
  route 12 dest-interface IF3
  route default dest-interface IF4
```

Procedure: To delete an entire routing table

Mode: Context CS

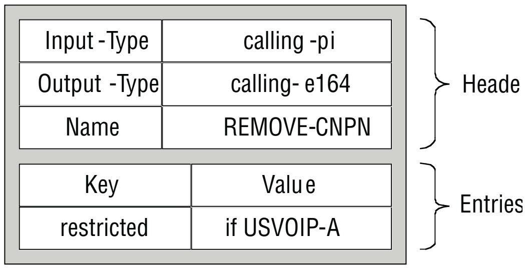
Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]#**no routing-table** *table-name*	Delete the routing table *table-name*. **Note:** You do not have to enter the type of the table when just deleting it. The type must only be specified when creating a table.

Example: Remove an entire routing table

```
node(cfg)#context cs
node(ctx-cs)[switch]#no routing-table MY-TABLE
```

Configure Mapping Tables

Mapping tables are used to modify the call setup message and thus influence the routing decision and or the outgoing setup message leaving the call router. Mapping tables are identified by a name (string) and referenced in the routing tables for execution. Like the routing table, a mapping table finds the best matching entry and executes it; but unlike the routing table, the execution means manipulating a call property. Thus a mapping table always examines one call property (the input-type) and changes another property (the output-type). As for the routing tables, the call router provides a number of different mapping table types. A mapping table looks like the following:



Input -Type	calling -pi
Output -Type	calling- e164
Name	REMOVE-CNPN

Key	Valu e
restricted	if USVOIP-A

Figure 75. Mapping table outline

Each table contains a header and one or more entries. The header declares the input and output-type of the mapping table as well as its name.

Unlike a routing table, a call property pair characterizes a mapping table, the input and output-type. While the input-type defines which call property is examined by the call router, the output-type defines which property is modified once the best matching entry is found, for example, you may want to find a best matching entry in a mapping table based on the presentation indicator and, once found, you want to manipulate the calling party number of the call. In this case you chose an input-type of calling-pi and an output-type of calling-e164

There are three different kinds of call properties, calling party properties, called party properties and generic properties. When a call setup is to be routed by the call router, most properties appear as calling and as called party properties, for example, you have a calling party number and a called party number to examine. There are other properties (e.g. the information transfer capability), which is a property of the whole call and not of either party.

You can create a mapping table that examines and modifies a specific kind of property, e.g. the called party number. In this case you have to specify an input-type of called-e164 and an output-type of called-164. If you want to replace both, the called and the calling party property with the same mapping table, you can create a mapping table with input-type e164 and output-type e164, i.e. without prefixing the input- and output-type with "called-".

The name of the mapping table is unique inside the context and serves as identifier for referencing the mapping table from other routing tables. Almost every type explained with the routing table above can be used as input and output-type of a mapping table.

Table 31. Mapping table types

Type	Description Input-Type	Description Output-Type
called-e164	Select an entry based on the called party E.164 number. You can use wildcards to summarize entries.	Modifies the called party E.164 number. If the input-type is a called-e164 or a calling-e164 type, you can use replacement operators to use parts of the lookup key.
calling-e164	Select an entry based on the calling party E.164 number. You can use wildcards to summarize entries.	Modifies the calling party E.164 number. If the input-type is a called-e164 or a calling-e164 type, you can use replacement operators to use parts of the lookup key.
e164	If the output-type is also a generic kind of property, this mapping table is applied to both, this calling-e164 and the called-e164 property.	If the input-type is also a generic kind of property, this mapping table is applied to both, the calling-e164 and the called-e164 property.
called-type-of-number	Select an entry based on the called party number type. ISDN distinguishes different type of numbers.	Sets the called party number type.
calling-type-of-number	Selects an entry based on the calling party number type. ISDN distinguishes different type of numbers.	Sets the calling party number type.
type-of-number	If the output-type is also a generic kind of property, this mapping table is applied to both, this calling-type-of-number and the called- type-of-number.	If the input-type is also a generic kind of property, this mapping table is applied to both, the calling-type-of-number and the called-type-of-number property.
called-numbering-plan	Selects an entry based on the called party numbering plan. ISDN distinguishes different numbering plans.	Sets the called party numbering plan.
calling-numbering-plan	Selects an entry based on the calling party numbering plan. ISDN distinguishes different numbering plans.	Sets the calling party numbering plan.
numbering-plan	If the output-type is also a generic kind of property, this mapping table is applied to both, this calling-numbering-plan and the called-numbering-plan.	If the input-type is also a generic kind of property, this mapping table is applied to both, the calling-numbering-plan and the called-numbering-plan property.
called-name	Selects an entry based on the display name of the called party.	Sets the display name of the called party.
calling-name	Selects an entry based on the display name of the calling party.	Sets the display name of the calling party.

Table 31. Mapping table types (Continued)

Type	Description Input-Type	Description Output-Type
name	If the output-type is also a generic kind of property, this mapping table is applied to both, this calling-name and the called-name.	If the input-type is also a generic kind of property, this mapping table is applied to both, the calling-name and the called-name property.
called-ip	Selects an entry based on the remote signaling IP address of the destination VoIP peer.	Sets the remote IP address of the destination VoIP peer.
calling-ip	Selects an entry based on the remote signaling IP address of the origination VoIP peer.	Sets the remote IP address of the origination VoIP peer.
ip	If the output-type is also a generic kind of property, this mapping table is applied to both, this calling-ip and the called-ip.	If the input-type is also a generic kind of property, this mapping table is applied to both, the calling-ip and the called-ip property.
calling-pi	Selects an entry based on the presentation indicator.	Sets the presentation indicator.
calling-si	Selects an entry based on the screening indicator.	Sets the screening indicator.
calling-second-e164	Select an entry based on the second calling party E.164 number. For having a second calling party E.164 number available, mappings from call-properties in the incoming interface of the calls are needed. The same rules as for the normal calling party E.164 number apply.	Modifies the second calling party E.164 number. The same rules as for the normal calling party E.164 number apply.
calling-second-type-of-number	Select an entry based on the second calling party number type. For having a second calling party number type available, mappings from call-properties in the incoming interface of the calls are needed. The same rules as for the normal calling party number type apply.	Modifies the second calling party number type. The same rules as for the normal calling party number type apply.
calling-second-numbering-plan	Select an entry based on the second calling party numbering plan. For having a second calling party numbering plan available, mappings from call-properties in the incoming interface of the calls are needed. The same rules as for the normal calling party numbering plan apply.	Modifies the second calling party numbering plan. The same rules as for the normal calling party numbering plan apply.

Table 31. Mapping table types (Continued)

Type	Description Input-Type	Description Output-Type
calling-second-pi	Select an entry based on the second presentation indicator. For having a second presentation indicator available, mappings from call-properties in the incoming interface of the calls are needed. The same rules as for the normal presentation indicator apply.	Modifies the second presentation indicator. The same rules as for the normal second presentation indicator apply.
calling-second-si	Select an entry based on the second screening indicator. For having a second screening indicator available, mappings from call-properties in the incoming interface of the calls are needed. The same rules as for the normal screening indicator apply.	Modifies the second screening indicator. The same rules as for the normal second screening indicator apply.
itc	Selects an entry based on the information transfer capability (bearer capability) to distinguish speech from data calls.	Sets the information transfer capability.
time	Route calls based on the current time of day.	Cannot be set.
date	Route calls based on the current date.	Cannot be set.
day-of-week	Route calls based on the current day of week.	Cannot be set.

Besides the header (name, input and output-type) a mapping table contains multiple entries. Each entry specifies a specific value of the routing table input-type and a value that shall be applied to the call property specified with the output-type of the table. When a call arrives a mapping table, the following procedure is applied:

1. Examine the call property as specified with the mapping table input-type.
2. Select the best matching entry. This means that the key of each of the entries is compared to the call property and the entry that matches best is chosen.
3. Execute the entry. This means replacing the property specified with the output-type of the mapping table with the value of the selected entry.

Let's examine the mechanism of mapping tables in more detail. Figure 76 shows three mapping tables and a call that is routed through this mapping table. The call contains various call properties that are examined and modified by the mapping tables:

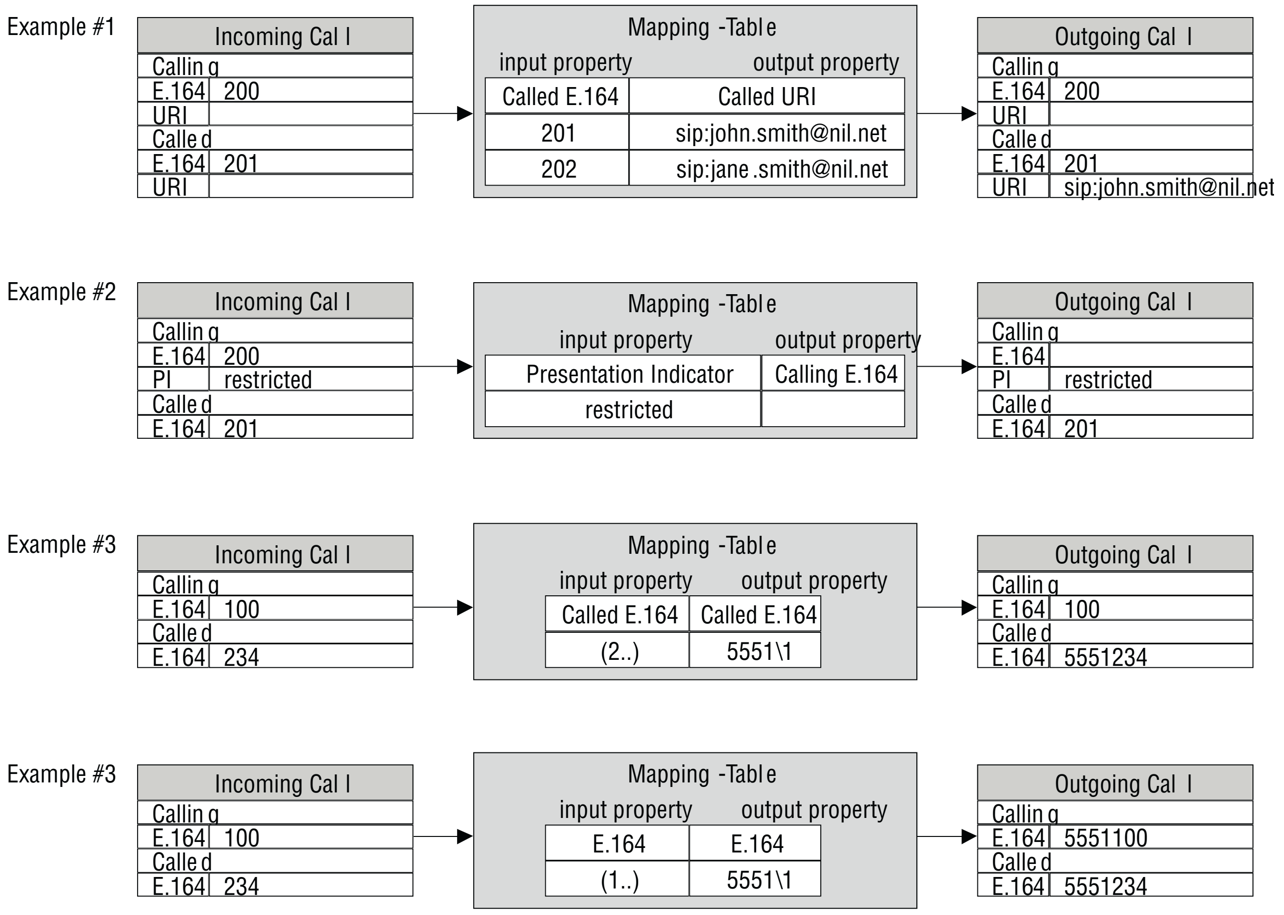


Figure 76. Mapping table examples

Example #1: This example shows a mapping table that selects the best matching entry based on the called party number of a call and, once found, sets the called party URI. In the example a call arrives to the mapping table with a called party number of 201. The mapping table selects the first entry, which matches best, and sets the called party URI of the call to *sip:john.smith@nil.net*.

Example #2: This example shows a mapping table that selects the best matching entry based on the presentation indicator and, once found, sets the called party number. In the example a call arrives to the mapping table with a presentation indicator of *restricted*. The mapping table selects the only entry (which matches) and sets the called party number to the empty string.

Example #3: This example shows a mapping table that selects the best matching entry based on the called party number and, once found, changes the same property, the called party number. This is a very powerful method to manipulate numbers using regular expressions. In this example a call arrives to the mapping table with a called party number of 234. The mapping table selects the only entry (which matches) and adds a prefix of 5551 to the called party number.

Example #4: This example shows the same input call properties as in example #3. The mapping table is also almost the same, but unlike in the previous example, here we don't look for a specific number type (e.g. called party number, calling party number), but for any E.164 number property of the call. The output property is also a generic number. In this case the mapping table replaces both, the calling and the called party number. For example, the mapping table applies its algorithm to all E.164 numbers of the call.

Note Like a routing table, a mapping table selects the best matching entry based on the input property type. This is done using the best matching prefix method for E.164 numbers and string compare for other properties. Unless you don't have a default entry in a mapping table, no action is performed when no match can be found and the call is not dropped. In the above example #3, if a call arrives the mapping table with a called party number of 200, which does not match the entry (1..), the called party number is not changed.

Procedure: To create a mapping table and add entries

Mode: Context CS

Step	Command	Purpose
1	node(ctx-cs)[switch]#**mapping-table** *input-type* **to** *output-type table-name*	Create a mapping table *table-name* that examines the call property specified with *input-type* and modifies the call property specified with *output-type*. This enters the table mode where entries can be added or removed. To enter a previously created table from the context CS mode, you may leave away the *input-type* and *output-type*.
2	node(map-tab)[table-name]#**map** *key* **to** *value*	Add an entry to the mapping that sets the *output-type* call property to *value* if the *output-type* call property matches the *key*. The format of the *key* depends on the *input*-type of the table, and the format of the *value* depends on the *output-type* of the table.
3		Repeat step 2 to add lines for additional table entries.

Example: Called and calling party manipulation mapping table

```
node(cfg)#context cs
node(ctx-cs)[switch]#mapping-table e164 to uri SETURI
node(rt-tab)[switch.SETURI]#map 100 to sip:john.smith@nil.net
node(rt-tab)[switch.SETURI]#map 101 to sip:jane.smith@nil.net
```

E.164 to E.164 Mapping Tables

As with routing tables you can use regular expressions when selecting an entry in a mapping table based on a calling or called party number. If the output property type of a mapping table is also a calling or called party number, you may use parts of the matched expressions when building the modified number as shown in example #3 above.

Detailed Example: You have an internal dial plan that uses three digit numbers starting with a 2 (e.g. 200, 201, etc.). So when an internal subscriber makes a call, its calling party number contains three digits.

1. You want to route calls to the public switched telephone network (PSTN), that is reachable over and ISDN interface. From the PSTN provider you have an assigned number range from 099-8882500 to 099-8882599.
2. You want to pass the last two digits of your internal subscribers when they are making calls to the PSTN. Thus subscriber 244 should make a call to the PSTN using a calling party number of 099-8882544.

To achieve this, create a mapping table that looks like the following:

```
node(cfg)#context cs
node(ctx-cs)[switch]#mapping-table calling-e164 to calling-e164 MAP-PSTN
node(rt-tab)[switch.MAP-PSTN]#map 2(..) to 09988825\1
```

When a call reaches this table with a calling party number of 244, this number is tried to match to the entries of this table:

```
2(44) matches 2(. .)
```

Thus the only entry is selected and executed. This means setting the calling party number to 09988825\1. The last part of the value (a backslash followed by a single digit number) is a placeholder and means that the first pattern (expression in brackets) of the key shall be used instead.

Thus the called party number is replaced with the specified prefix 09988825 concatenated with the bracketed pattern in the key (44). The result is 0998882544.

Like this you can use brackets around any party of the expression of the key and use the part that matches to this bracket in the value you set.

Example: Mapping table to add a prefix to the called party number

Input:called-e164 = 0998882525

Output:called-e164 = *50998882525

```
node(cfg)#context cs
node(ctx-cs)[switch]#mapping-table called-e164 to called-e164 ADD-PFX
node(rt-tab)[switch.ADD-PFX]#map (.%) to *5\1
```

The input 0998882525 matches the expression (.%) – any character repeated zero or more times.

The first bracket encloses the whole number: (.%) == (0998882525) -> \1 = 0998882525

The output is built as concatenation of *5 and the first bracket \1.

The called party number is set to *50998882525

Example: Mapping table to remove a prefix from the called party number

Input:called-e164 = *50998882525

Output:called-e164 = 0998882525

```
node(cfg)#context cs
node(ctx-cs)[switch]#mapping-table called-e164 to called-e164 REM-PFX
node(rt-tab)[switch.REM-PFX]#map *5(.%) to \1
```

The input *50998882525 matches the expression *5(.%) – the prefix *5 followed by any character repeated zero or more times.

The first bracket encloses the number after the prefix: *5(.%) == *5(0998882525) -> \1 = 0998882525

The output is built from the first bracket \1.

The called party number is set to 0998882525.

Example: Mapping table to truncate the called party number

Input:called-e164 = 0998882525

Output:called-e164 = 525

```
node(cfg)#context cs
node(ctx-cs)[switch]#mapping-table called-e164 to called-e164 TRUNC
node(rt-tab)[TRUNC]#map .%(...) to \1
```

The input 0998882525 matches the expression .%(…) – any character repeated zero or more times followed by three mandatory digits.

The first bracket encloses the last three digits: .%(…) == 0998882(525) -> \1 = 525

The output is built from the first bracket \1.

The called party number is set to 525.

Example: Mapping table to remove the calling party number when restricted

Input:calling-e164 = 0998882525; calling-pi = restricted

Output:calling-e164 = “”; calling-pi = restricted

```
node(cfg)#context cs
node(ctx-cs)[switch]#mapping-table calling-pi to calling-e164 REM-CNPN
node(rt-tab)[switch.REM-CNPN]#map restricted to “”
```

The input (presentation indicator) restricted matches the expression restricted.

The output (calling party number) is an empty string (“”).

The calling party number is cleared.

Output:calling-e164 = 0778881111; called-e164 = 0778881111

```
node(cfg)#context cs
node(ctx-cs)[switch]#mapping-table called-e164 to calling-e164 COPY-PN
node(rt-tab)[switch.COPY-PN]#map (.%) to \1
```

The input called party number 0778881111 matches the expression (.%) – any character repeated zero or more times.

The first bracket encloses the last whole called party number: (.%) == (0778881111) -> \1 = 0778881111

The output (calling party number) is built from the first bracket \1.

The calling party number is set to 0778881111.

Custom SIP URIs from Called-/Calling-e164 Properties

The regular expression engine used for mapping-tables and routing-tables in the Call Router can handle not only handle digits, but whole strings. You can construct custom SIP URIs from call leg properties as called and calling e.164. See the following examples.

Example 1: Use regular expressions to create mapping tables that map the called-party-e164 number to the called-party-URI for SIP calls. The following example shows how to build a SIP To-URI from the called-party number.

Mode: Context CS

Step	Command	Purpose
1	**node(ctx-cs)[switch]#mapping-table called-e164 to called-uri** name	Creates a mapping table that examines the called-party number and sets the called-party URI (To-URI).
2	**node(map-tab)[***switch.device***]#map (.+) to sip:user_\1@example.com**	If the called-party number exists (at least one digit) the called-party URI is set to user_<called-e164>@example.com
3	**node(map-tab)[***name***]#map default to sip:anonymous@example.com**	If the called-party number does not exist (calls that don't match to the rule of step 2), the called-party URI is set to anonymous@example.com

Example 2: Use a mapping table to set the display name field of To-URIs for outgoing SIP calls from the called-party number of an incoming call. The following example shows how to set the called-party name based on the called-party number.

Mode: Context CS

Step	Command	Purpose
1	**node(ctx-cs)[switch]#mapping-table called-e164 to called-name** name	Creates a mapping table that examines the called-party number and sets the called-party name.
2	**node(map-tab)[***switch.device***]#map (.%) to \1**	Copies the whole called-party number to the called-party name.

Other Mapping Tables

Example: Mapping table to set the called party number type to international (unconditionally)

Input:called-e164 = 0041998882525; calling-type-of-number = unknown

Result:called-e164 = 0041998882525; calling-type-of-number = international

```
node(cfg)#context cs
node(ctx-cs)[switch]#mapping-table called-type-of-number to called-type-of-number
SET-INT
node(rt-tab)[SET-INT]#map default to international
```

Any called party number type matches the default entry. Note that the input-type of the table does not matter when the mapping table contains only the default entry. Anyway an input-type must be specified when creating the mapping table.

The output (called party number type) is international.

The called party number type is set to international.

Example: Mapping table to replace called party numbers (translation table)

```
node(cfg)#context cs
node(ctx-cs)[switch]#mapping-table called-e164 to called-e164 TRANS
node(rt-tab)[switch.TRANS]#map 550 to 250
node(rt-tab)[switch.TRANS]#map 551 to 251
node(rt-tab)[switch.TRANS]#map 552 to 252
node(rt-tab)[switch.TRANS]#map 553 to 253
node(rt-tab)[switch.TRANS]#map 554 to 254
node(rt-tab)[switch.TRANS]#map 555 to 255
node(rt-tab)[switch.TRANS]#map 556 to 256
node(rt-tab)[switch.TRANS]#map 557 to 257
node(rt-tab)[switch.TRANS]#map 558 to 258
node(rt-tab)[switch.TRANS]#map 559 to 259
```

Note The translation table above can be reduced using regular expressions.

```
node(cfg)#context cs
node(ctx-cs)[switch]#mapping-table called-e164 to called-e164 TRANS
node(rt-tab)[switch.TRANS]#map 55(.) to 25\1
```

Deleting Mapping Tables

To remove individual mapping tables you can use the **no** form of the **mapping table** command. Alternatively you can remove specific entries of a mapping table by entering the mapping table configuration mode and use the **no** form of the **map** command.

Procedure: To delete an entry from a mapping table

Mode: Context CS

Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]#**mapping-table** *table-name*	Enter the mapping table from which you want to remove an entry. **Note:** You do not have to enter the type of the table when just entering it. The type must only be specified when creating a table.
2	*node*(map-tab)[*table-name*]#**no map** *key*	Remove the entry with the specified *key*.
3		Repeat Step 2 to remove additional entries.

Example: Remove entries from a mapping table

The running-config shows the following table:

```
mapping-table called-e164 to called-e164 MY-TABLE
  map 10 to 20
  map 11 to 21
  map 12 to 22
  map 13 to 23
```

To remove the first two entries from the table enter the following commands:

```
node(cfg)#context cs
node(ctx-cs)[switch]#mapping-table MY-TABLE
node(map-tab)[switch.MY-TABLE]#no map 10
node(map-tab)[switch.MY-TABLE]#no map 11
```

The resulting running-config is:

```
mapping-table called-e164 to called-e164 MY-TABLE
  map 12 to 22
  map 13 to 23
```

Procedure: To delete an entire mapping table

Mode: Context CS

Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]#**no mapping-table** *table-name*	Delete the mapping table *table-name*. **Note:** You do not have to enter the type of the table when just deleting it. The type must only be specified when creating a table.

Example: Remove an entire mapping table

```
node(cfg)#context cs
node(ctx-cs)[switch]#no mapping-table MY-TABLE
```

Creating Complex Functions

Complex functions allow combining mapping tables, which need to be executed in sequence. This is useful if, for example, the calling and the called party number have to be modified in the same step. Complex function names can be any arbitrary string.

Procedure: To create a complex number manipulation function

Mode: Context CS

Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]#**complex-function** *function-name*	Create a complex function *function-name*.
2	node(func)[function-name]#**execute** *function* or node(func)[function-name]#**execute** *index function*	Add or inserts an entry to the complex function. *function* can be another complex function or a mapping table that shall be executed. **Note:** Unlike routing and mapping tables, complex functions are ordered lists of entries, which means that the entries are executed in the order of appearance. You can optionally specify an *index* of where to insert the function.
3		Repeat step 2 to add lines for additional functions to execute.

Example: Create a complex function

```
node(cfg)#context cs
node(ctx-cs)[switch]#complex function CARRIER-TO-LOCAL
node(func)[switch.CARRIER~]#execute 1 TRUNCATE-3-CNPN
node(func)[switch.CARRIER~]#execute 2 DDI-TO-PISN
```

Deleting Complex Functions

To remove individual complex functions you can use the **no** form of the **complex function** command. Alternatively you can remove specific entries of a complex function by entering the complex function configuration mode and use the **no** form of the **execute** command.

Procedure: To delete an entry from a complex function

Mode: Context CS

Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]#**complex-function** *function-name*	Enter the complex function from which you want to remove an entry.
2	*node*(func)[switch.*table-name*]#**no execute** *index*	Remove the entry with the specified index.
3		Repeat Step 2 to remove additional entries.

Example: Remove entries from a complex function

The running-config shows the following complex function:

```
complex function MY-FUNC
  execute 1 MAP1
  execute 2 MAP2
  execute 3 MAP3
  execute 4 MAP4
```

To remove the first two entries from the complex function enter the following commands. Pay attention on the index. When removing the first entry, the MAP2 function becomes entry with index 1. Thus you have to specify index 1 twice to remove the first two entries:

```
node(cfg)#context cs
node(ctx-cs)[switch]#complext-function MY-FUNC
node(func)[switch.MY-FUNC]#no execute 1
node(func)[switch.MY-FUNC]#no execute 1
```

The resulting running-config is:

```
complex function MY-FUNC
  execute 1 MAP3
  execute 2 MAP4
```

Procedure: To delete an entire complex function

Mode: Context CS

Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]#**no complex-function** *function-name*	Delete the complex function *function-name*.

Example: Remove an entire complex function

```
node(cfg)#context cs
node(ctx-cs)[switch]#no complex-function MY-FUNC
```

Digit Collection & Sending-complete Behavior

Sending-complete

The call-router can be configured how to handle address-complete indications (e.g. ISDN Sending-Complete IE). On ISDN and SIP interfaces, an **address-complete-indication** command controls the tunneling of **address-complete** indications separately for incoming and outgoing calls.

Each interface can be configured to pass transparently address-complete indications, or to explicitly insert or remove it for incoming and outgoing calls. In addition the behavior of the call-router can be configured.

Some call signaling protocols allow a user to dial a destination by using the overlap-sending procedure. These protocols include analog telephony (FXS/FXO) and ISDN. ISDN support address-complete indication using the Sending-Complete Information Element. Other protocols wait on a timeout or send a terminating character, e.g. pound (#), to indicate address-completion. The following commands control the tunneling of address-complete indications from the signaling protocol to the internal call-control representation and vice-versa. In addition, the context cs introduce command extensions that control the address-completion handling in the internal call-router.

Ingress interface

On the ingress interface (ISDN or SIP) the **address-complete-indication accept <type>** command configures how to map the address-complete indication of the signaling-protocol to the internal call-control. There the indication can be used for call-routing (see below) or mapped again to an address-complete indication on an egress interface (see below).

The possible values for the type argument are:

- **transparent**: Transparently passes an address-complete indication (e.g. ISDN Sending Complete Information Element) to the call-control.
- **set**: Always sets the address-complete indication flag towards the call-control even when no such indication is received from the calling party. This configuration can be used to disable overlap-sending on an interface.
- **clear**: Never signals the address-complete indication towards the call-control even when the indication is received from the calling party. This configuration may be used in rare cases to solve interoperability problems with attached PBXs.

Some interface types do not support all of the mentioned arguments. The following table shows the supported address-complete indication conversion types and the default setting for each interface type:

Interface Type	Transparent	Set	Clear
ISDN	supported; default	supported	Supported
SIP	not supported	supported	supported; default

Call-router

The call-router is extended to be able to set the address-complete indication or append a terminating-character in some circumstances. This includes:

- Setting the address-complete indication when the digit-collection timeout elapses
- Appending the terminating-character when the digit-collection timeout elapses
- Filtering-out the terminating-character and optionally set the address-complete indication
- Setting the address-complete indication when a called-party number matches a fully specified call-router rule ($-terminated entry).
- Appending the terminating-character when the called-party number matches a fully specified call-router rule ($-terminated entry).

The command **digit-collection timeout <secs>** has been extended with two optional arguments that specify whether a terminating-character is appended or the address-complete indication set when the digit-timeout elapses. The command syntax is

[no] digit-collection timeout <secs> [append-terminating-char] [set-address-complete-indication]

The extensions configure the action(s) to be performed when the digit-collection timeout elapses. The digit-collection timeout runs when a call is routed to a called-e164 routing-table of which a T-terminated rule is selected. Two actions are available; both can be enabled independently. The **append-terminating-char** action appends the configured terminating-character when the timeout elapses. The **set-address-complete-indication** action sets the address-complete indication. Whether or not the egress interface propagates the address-complete indication depends on the interface configuration (see below).

The following table shows the different digit-collection timeout configurations and their effect on a T-terminated route when the digit-collection timeout elapses. Important settings are marked bold.

Digit Collection Timeout Configuration			Ingress Properties		Egress Properties (Result of call-routing)	
Timeout	**append-terminating-char**	**set-address-complete-indication**	**called-e164**	**Address-Complete Indication**	**called-e164**	**Address-Complete Indication**

Digit Collection Timeout Configuration			Ingress Properties		Egress Properties (Result of call-routing)	
no	no	no	123	false	no call	
				true		
		yes		false		
				true		
	yes	no		false		
				true		
		yes		false		
				true		
yes	no	no		false	123	false
				true		true
		yes		false		**true**
				true		true
	yes	no		false	**123#**	false
				true		true
		yes		false		**true**
				true		true

The command **digit-collection terminating-char <char>** has been extended with two optional arguments that specify whether the terminating-character is re-appended or the address-complete indication is set when a digit-collection timeout is stopped by the user sending the terminating-character. The new command syntax is:

[no] digit-collection terminating-char <char> [append-terminating-char] [set-address-complete-indication]

The extensions configure what action(s) shall be performed when the digit-collection timeout is stopped by the reception of a terminating-character. Normally, without specifying an action, the received terminating-character is removed from the called-party number. The **append-terminating-char** action re-appends the terminating-character. The **set-address-complete-indication** action sets the address-complete indication. Whether or not the exit interface propagates the address-complete indication depends on the interface configuration (see below).

The next table shows the different digit-collection terminating-character configurations and their effect on a T-terminated route when the terminating-character is received.

Digit Collection Terminating-Char Configuration			Ingress Properties		Egress Properties (Result of call-routing)	
Terminating-Char	**append-terminating-char**	**set-address-complete-indication**	**called-e164**	**Address-Complete Indication**	**called-e164**	**Address-Complete Indication**
No	no	no	123#	False	123#	false
				true		true
		yes		false		false
				true		true
	yes	no		false		false
				true		true
		yes		false		false
				true		true
Yes	no	no		false	123	false
				true		true
		yes		false		**true**
				true		true
	yes	no		false	**123#**	false
				true		true
		yes		false		**true**
				true		true

The command **digit-collection full-match** has been introduced. The syntax is:

digit-collection full-match [append-terminating-char] [set-address-complete-indication]

This command configures what action(s) shall be performed when a dollar-($)-terminated rule is selected in a called-e164 routing-table. We call such an entry a fully specified number entry. The **append-terminating-char** action appends the terminating-character; while the **set-address-complete-indication** action sets the address-complete indication. Whether or not the egress interface propagates the address-complete indication depends on the interface configuration (see below).

Egress interface

On the egress interface (ISDN) the **address-complete-indication emit <type>** command configures how the internal representation of the address-complete indication shall be mapped to the representation of the signaling-protocol.

The possible values for the type argument are:

- **transparent**: Transparently passes an address-complete indication to the signaling-protocol (e.g. ISDN by sending a Sending Complete Information Element).
- **set**: Always sends a Sending Complete Information Element with the SETUP message. This configuration can be used to disable overlap-sending on an interface.
- **clear**: Never sends a Sending Complete Information Element. This configuration may be used in rare cases to solve interoperability problems with attached PBXs, e.g. when the attached PBX does not support the Sending Complete Information Element.

Some interface types do not support all arguments. SIP does not support this configuration at all, because SIP does not support overlap dialing. The following table shows the supported address-complete indication conversion types and the default setting for each interface type:

Interface Type	Transparent	Set	Clear
ISDN	supported; default	supported	Supported
SIP	—	—	—

The following procedure demonstrates how to disable overlap-sending for incoming SIP calls. SIP does not provide an overlap-dialing procedure; so for most applications, address-complete indications should be cleared.

Mode: context cs / interface sip

Step	Command	Purpose
1	***node*(if-sip)[*if-name*]#address-complete-indication clear**	Clear the address-complete indication for all incoming calls over this interface

The following procedure demonstrates how to create a routing-table that allows overlap dialing. When the timeout elapses or when the terminating-character is received, the address-complete indication shall be set toward the egress interface.

Mode: context cs

Step	Command	Purpose
1	***node*(cts-cs)[*ctx-name*]#routing-table called-e164 <tab-name>**	Creates a routing-table that examines the called-party number
2	***node*(rt-tab)[tab-*name*]#route T dest-interface <if-name>**	Routes any number (after the waiting for digits) to the egress interface

Step	Command	Purpose
3	***node*(rt-tab)[tab-*name*]#exit**	Leaves the routing table configuration mode and returns to the context cs configuration mode
4	***node*(cts-cs)[*ctx-name*]#digit-collection timeout 5 set-address-complete-indication**	Configures the digit-collection timeout to 5 seconds and sets the address-complete indication if the timeout elapses
5	***node*(cts-cs)[*ctx-name*]#digit-collection terminating-char # set-address-complete-indication**	Configures the terminating-character that can stop the digit-collection timeout and immediately place the call to '#' and sets the address-complete indication if the character is received

Creating Call Services

Routing tables, mapping tables and complex functions only manipulate address properties of a call (like the called party number). A call service is another call router entity that actively accesses the state of a call. A call service is able to spawn other calls or merge existing calls together. This allows building services like hunt groups etc.

You can view a call service as a virtual endpoint that accepts a call and creates new calls to other destinations. A call that is routed to a call service is not served by a human being but by a machine that accepts the call and performs needed actions.

The hunt group service, for example, accepts a call that is routed to it and spawns a second call that is placed to the first final destination. If this destination is not reachable, another destination is tried until one of the configured destinations accept the call.

Creating a Hunt Group Service

A hunt group service hunts an incoming call to multiple interfaces. figure 77 shows an example scenario where a call from a SIP interface is first processed by several tables. The second table decides that the call must be forwarded to the PSTN. The device is connected to the PSTN over four BRIs, which are bound to the context CS interfaces IF-BRI0 up to IF-BRI3. All four ISDN interfaces lead to the same provider. Since the call router does not know the load on the BRIs, it has to be able to try BRI0 and, if BRI0 already serves two calls, use BRI1, and so on.

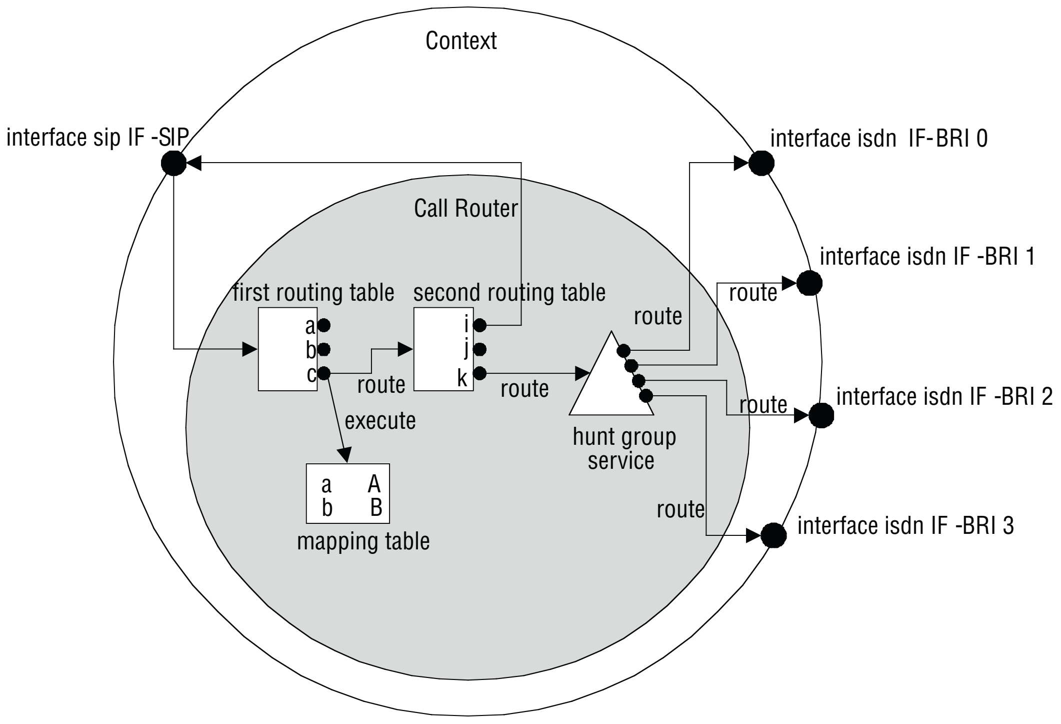


Figure 77. Hunt group service

The hunt group service accepts a call routed to it by a routing table or directly from an interface and creates another call that is offered to one of the configured destination interfaces.

The interface tried first (IF-BRI0) may drop the call telling the service that the interface has no resources to handle the call (e.g. no circuit channel available). Note that only the call between the hunt group and the destination interface is dropped, while the original call between the SIP interface and the hunt group service remains connected.

The hunt group then decides to try the next destination (IF-BRI1), which in turn also drops the call due to unavailable resources. In our example the hunt group then tries the third and eventually the fourth destination. When an interface accepts a call, the interface hunting is complete and the hunt group service merges the original with the new call to the interface that accepted the call.

You can influence the algorithm of the hunt group by several configuration commands. You can specify whether the hunt group shall always start with the same destination interface or whether it shall immediately try the next one in a round-robin fashion. This is called cyclic operation mode.

You can specify a timeout after which the next destination interface is tried when there is no answer at all from the destination interface.

You can specify drop causes that trigger hunting for the next destination. All other causes (e.g. user busy) will drop the original call.

Note You can hunt a call over different interface types, not only over ISDN interfaces. You can, e.g. create a hunt group to try to call over a SIP interface and, if this call fails, do a fallback to an ISDN interface.

Procedure: To create and configure a hunt group service

Mode: Context CS

Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]#**service hunt-group** *service-name*	Creates a new hunt group service and enters hunt group configuration mode.
2	*node*(svc-hunt)[service-name]#**cyclic** *or* *node*(svc-hunt)[service-name]#**no cyclic**	Configure the hunt group for cyclic operation mode. Subsequent calls try another first destination in a round-robin method. Default is not to use cyclic mode – always to start with the first configured destination. **Note:** When you use the hunt-group for a fallback scenario, you must switch off cyclic operation mode.
3	*node*(svc-hunt)[service-name]#**timeout** *timeout*	Configures a timeout in seconds after which the next destination is tried when the current destination does not answer at all. (Some interface (e.g. SIP) may wait an arbitrary long time until an answer is returned.) Default is not to use a timeout.
4	*node*(svc-hunt)[service-name]#**drop-cause** *cause* or *node*(svc-hunt)[service-name]#**no drop-cause** *cause*	Enables or disables another drop cause to the list. When an interface has a problem placing a call to the final destination it drops the call specifying a drop cause (e.g. user busy, no resource available). Some drop causes drop the original call while other causes trigger the hunt for another destination. This command can be used to configure the hunt behavior on destination call drop. See the list below for a summary of all available drop causes and their default state.
5	*node*(svc-hunt)[service-name]#**route call dest-interface** *interface-name* or *node*(svc-hunt)[service-name]#**route call dest-table** *table-name* or *node*(svc-hunt)[service-name]#**route call dest-service** *service-name*	Adds a route to a destination. This is the destination that is tried during hunt group's interface hunting. The destination can either be an interface or you can route the call again to a routing table or directly to another service. This allows you cascading services.
6		Repeat step 5 to add additional hunting destinations.

When a destination interface drops the call, the hunt group service has to decide whether this is because the interface is not able to handle the call (e.g. no bearer channel available) or whether the destination user does not want or is not able to communicate (e.g. user busy). In the first case, the hunt group service hunts for the next destination interface, while in the second case the original call is dropped with the same cause.

The following table lists all drop causes and specifies whether the cause is used for hunting the next destination or dropping the original call. The behavior can be configured for each hunt group individually for each cause using the **drop-cause** command in the hunt group service mode.

Table 32. Hunt group drop causes

Class	Cause	Default Behavior of the Hunt Group Service	Description
Normal Event	unallocated-number	Drop original call	The number is sent in the correct format. However, the number is not assigned to the destination equipment.
	no-route-to-network	Drop original call	The destination is asked to route the call through an unrecognized network. This cause indicates that the equipment sending this cause has received a request to route the call through a particular transit network, which it does not recognize. The equipment sending this cause does not recognize the transit network either because the transit network does not exist or because that particular network, while it does exist, does not serve the equipment that is sending this cause.
	no-route-to-destination	Drop original call	The call routes through an intermediate network that does not serve the destination address. The called user cannot be reached because the network through which the call has been routed does not serve the destination desired.
	channel-unacceptable	Drop original call	The service quality of the specified channel is insufficient to accept the connection. The call attempt failed because the channel cannot be used.
	call-awarded	Drop original call	The user assigns an incoming call that is connecting to an already established call channel.
	normal-call-clearing	Drop original call	Normal call clearing occurs. This cause indicates that the call is being cleared because one of the users involved in the call has requested that the call be cleared. Under normal situations, the source of this cause is not the network.
	user-busy	Drop original call	The called system acknowledges the connection request but cannot accept the call because all channels are in use. It is noted that the user equipment is compatible with the call.

Table 32. Hunt group drop causes (Continued)

Class	Cause	Default Behavior of the Hunt Group Service	Description
Normal Event (Cont.)	no-user-responding	Drop original call	The connection fails because the destination does not respond to the call. This cause is used when a user does not respond to a call establishment message with either an alerting or a connect indication with the prescribed period of time allocated.
	no-answer-from-user	Drop original call	The destination responds to the connection request but fails to complete the connection within the prescribed time. This cause is used when the user has provided an alerting indication but has not provided a connect indication within a prescribed period of time.
	subscriber-absent	Drop original call	The remote device you attempted to reach is unavailable and has disconnected from the network.
	call-rejected	Drop original call	The destination can accept the call but rejects it for an unknown reason. This cause indicates that the equipment sending this cause does not wish to accept this call, although it could have accepted the call because the equipment sending this cause is neither busy nor incompatible.
	number-changed	Drop original call	The number used to set up the call is not assigned to a system. This cause is returned to a calling user when the called party number indicated by the calling user is no longer assigned.
	non-selected-user-clearing	Drop original call	The destination can accept the call but rejects it because it is not assigned to the user.
	destination-out-of-order	Drop original call	The destination cannot be reached because of an interface malfunction, and a signaling message cannot be delivered. This can be a temporary condition, but it could last for an extended period.
	invalid-number-format	Drop original call	The connection fails because the destination address is presented in an unrecognizable format, or the destination address is incomplete.
	facility-rejected	Drop original call	The network cannot provide the facility requested by the user.

Table 32. Hunt group drop causes (Continued)

Class	Cause	Default Behavior of the Hunt Group Service	Description
Normal Event (Cont.)	response-to-status-inquiry	Drop original call	The status message is generated in direct response to receiving a status inquiry message.
	normal-unspecified	Drop original call	Reports the occurrence of a normal event when no standard cause applies.
Resource Unavai-lable	no-circuit-channel-available	Hunt for next desti-nation	The connection fails because no appro-priate channel is available to take the call.
	network-out-of-order	Hunt for next desti-nation	The destination cannot be reached because of a network malfunction, and the condition can last for an extended period. An immediate reconnect attempt will probably fail.
	temporary-failure	Hunt for next desti-nation	An error occurs because of a network malfunction. The problem will be resolved shortly.
	switching-equipment-congestion	hunt for next desti-nation	The destination cannot be reached because the network switching equip-ment is temporary overloaded.
	access-info-discarded	Hunt for next desti-nation	The network cannot provide the requested access information. This cause indicates that the network could not deliver access information to the remote user as requested.
	circuit-channel-not-available	Hunt for next desti-nation	The equipment cannot provide the requested channel for an unknown rea-son.
	resources-unavailable	Hunt for next desti-nation	The requested channel or service is unavailable for an unknown reason.
Service or Option Not Available	qos-unavailable	Drop original call	The network cannot provide the requested quality of service.
	facility-not-subscribed	Drop original call	The remote equipment supports the requested supplementary service by subscription only. This cause indicates that the network could not provide the requested supplementary service because the user has not completed the necessary administrative arrangements with its supporting networks.
	bearer-capability-not-authorized	Drop original call	The user requests a bearer capability the network provides, but the user is not authorized to use it. This can be a sub-scription problem.

Table 32. Hunt group drop causes (Continued)

Class	Cause	Default Behavior of the Hunt Group Service	Description
Service or Option Not Available (Cont.)	bearer-capability-not-available	Drop original call	The network normally provides the requested bearer capability, but it is unavailable at the present time. This can be due to a temporary network problem or subscription problem.
	service-or-option-not-available	Drop original call	The network or remote equipment cannot provide the requested service option for an unspecified reason. This can be a subscription problem.
Service or Option Not Implemented	bearer-capability-not-implemented	Drop original call	The network cannot provide the bearer capability requested by the user.
	channel-type-not-implemented	Drop original call	The network or the destination equipment does not support the requested channel type.
	facility-not-implemented	Drop original call	The remote equipment does not support the requested supplementary service.
	only-restricted-digital-available	Drop original call	The network cannot provide unrestricted digital information bearer capability. This cause indicates that a device has requested an unrestricted bearer service but the equipment sending this cause only supports the restricted version of the requested bearer capability.
	service-or-option-not-implemented	Drop original call	The network or remote equipment cannot provide the requested service option for an unspecified reason. This can be a subscription problem.
Invalid Message	invalid-call-reference	Drop original call	The remote equipment receives a call with a call reference value that is not currently in use.
	channel-does-not-exist	Drop original call	The receiving equipment is requested to use a channel that is not activated on the interface for calls. This cause indicates that the equipment sending this cause has received a request to use a channel not activated on the interface for a call.
	call-identity-does-not-exist	Drop original call	This cause indicates that a call resume has been attempted with a call identity, which differs from that in use for any presently suspended call.

Table 32. Hunt group drop causes (Continued)

Class	Cause	Default Behavior of the Hunt Group Service	Description
Invalid Message (Cont.)	call-identity-in-use	Drop original call	This cause indicates that the network has received a call suspends request. The call suspend request contained a call identity which is already in use for a suspended call within the domain of interfaces over which the call might be resumed.
	no-call-suspended	Drop original call	The network receives a call resume request when there is not a suspended call pending. This can be a transient error that will be resolved by successive call retries.
	call-has-been-cleared	Drop original call	The network receives a call resume request. This call resume request contains a call identity that once indicated a suspended call. However, the suspended call was cleared either by timeout or by the remote user.
	incompatible-destination	Drop original call	Indicates that an attempt is made to connect incompatible equipment.
	invalid-transit-network	Drop original call	This cause indicates that a transit network identification of an incorrect format was received.
	invalid-message	Drop original call	Received an invalid message with no standard cause.

Table 32. Hunt group drop causes (Continued)

Class	Cause	Default Behavior of the Hunt Group Service	Description
Protocol Error	mandatory-ie-missing	Drop original call	The receiving equipment receives a message that does not include one of the mandatory information elements. This cause indicates that the equipment sending this cause has received a message that is missing a call property that must be present in the message before that message can be processed.
	message-type-not-implemented	Drop original call	The receiving equipment receives an unrecognized message, because the message type is invalid or the message type is valid but not supported.
	message-type-not-state-compatible	Drop original call	The remote equipment receives an invalid message with no standard cause. This cause indicates that the equipment sending this cause has received a message such that the procedures do not indicate that this is a permissible message to receive while in the current call state.
Protocol Error (Cont.)	ie-does-not-exist	Drop original call	The remote equipment receives a message that includes information elements or call properties that are not recognized.
	invalid-ie-contents	Drop original call	The remote equipment receives a message that includes invalid information in the information element or call property.
	recovery-on-timer-expiry	Drop original call	Your call was not completed, probably because an error occurred.
	protocol-error	Drop original call	An unspecified protocol error with no other standard cause occurred.
Interworking	interworking	Drop original call	An event occurs, but the network does not provide causes for the action it takes. The precise problem is unknown.

Example: Create a hunt group service

This example shows how to configure the hunt group service as shown in figure 77 on page 498.

```
node(cfg)#context cs
node(ctx-cs)[switch]#service hunt-group HUNT-BRI
node(svc-hunt)[HUNT-BRI]#cyclic
node(svc-hunt)[HUNT-BRI]#timeout 6
node(svc-hunt)[HUNT-BRI]#route dest-interface IF-BRI0
node(func)[HUNT-BRI]#route dest-interface IF-BRI1
node(func)[HUNT-BRI]#route dest-interface IF-BRI2
node(func)[HUNT-BRI]#route dest-interface IF-BRI3
```

Defining the Hunt-group Behavior for Inband Information

This command defines the behavior of a Hunt-Group service if a hunt-peer indicates the availability of inband information. Choose from three options:

- **Transparent:** This is the default behavior. Inband information is always passed through from hunt-peers to the call originator.
- **Block:** Inband information is never passed through from hunt-peers to the call originator.
- **Succeed:** If a hunt-peer announces inband information either in a provisional state or in disconnecting state, hunting is going to be stopped and the call is switched through the current destination. This happens even the peer disconnects the call with a cause that is configured as an indication to proceed hunting.

Mode: Service Hunt-Group

Step	Command	Purpose
1	**[node](svc-hunt)[*name*]# inband-information { transparent | block | succeed }**	Defines the Hunt-Group behavior if a destination announces inband information. Default: transparent

Creating a Distribution Group Service

A distribution group service distributes a call to multiple destinations interfaces. figure 78 shows an example scenario where a call from a SIP interface is first processed by several tables. The second table decides that the call must be forwarded to phones that are connected to various ISDN interfaces. The distribution now lets all four phones ring at the same time.

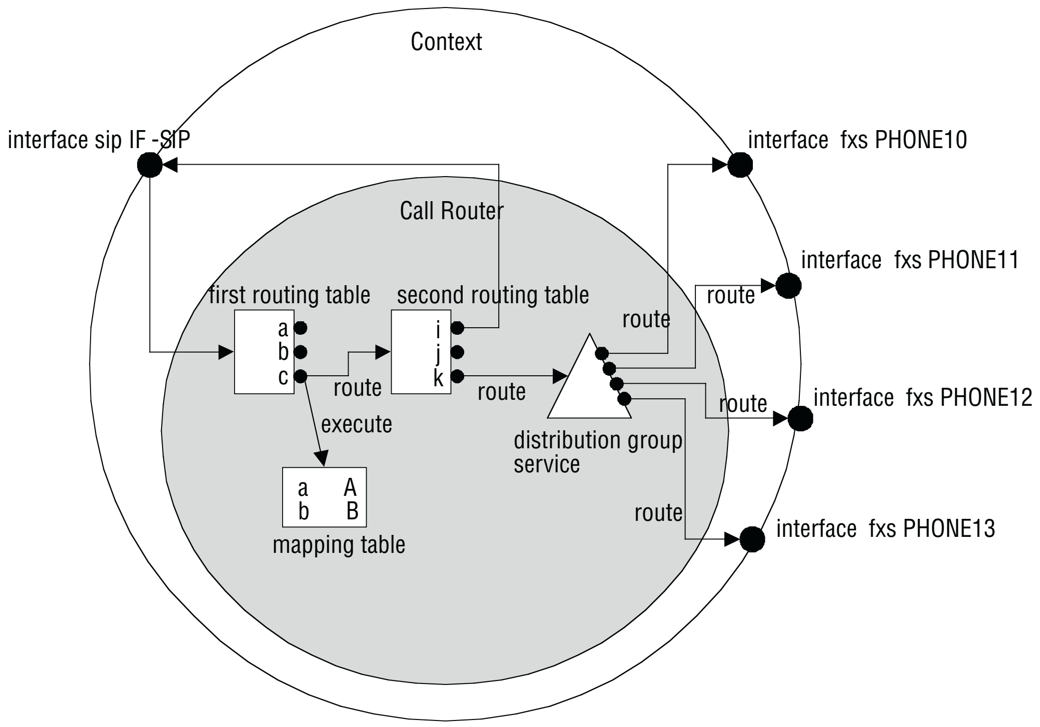


Figure 78. Distribution group service

The distribution group service accepts a call routed to it by a routing table or directly from an interface and creates four other calls that are offered to each of the configured destination interfaces.

All phones connected to the ISDN interfaces (*PHONE10–PHONE13*) start ringing. Eventually one of the phones (e.g. *PHONE10*) goes off-hook. The other three calls to interfaces *PHONE11*, *PHONE12*, and *PHONE13* are immediately dropped and the phones on these interfaces stop ringing. Now the distribution service is no longer needed. Thus the service merges the original call to the accepted destination call to interface *PHONE10*.

You can configure how the distribution algorithm works in many ways. You can specify the maximum number of destination interfaces that are called at the same time. Then you can specify a timeout after which a next destination is added to the destination calls. This makes it possible to configure a scenario where a call is offered to two destinations, and (when no one answers) stop ringing the first phone but try another third destination.

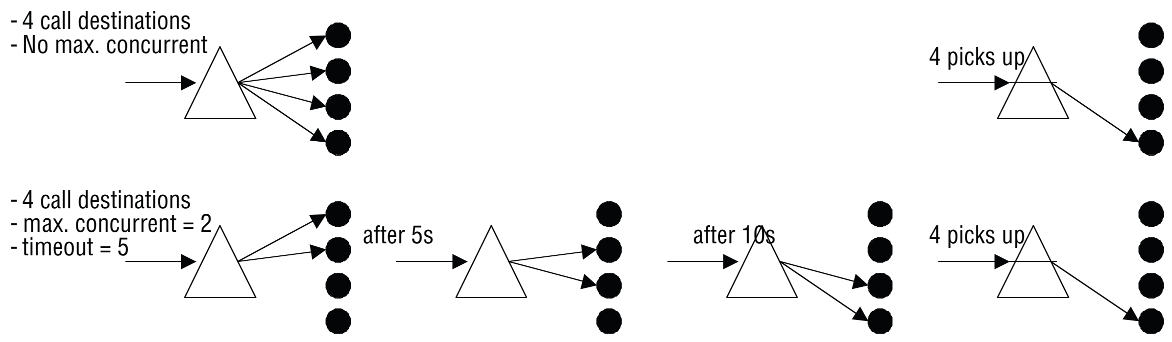


Figure 79. Distribution group service examples

Procedure: To create and configure a distribution group service

Mode: Context CS

Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]#**service distribution-group** *service-name*	Creates a new distribution group service and enters distribution group configuration mode.
2	*node*(svc-hunt)[*service-name*]#**cyclic**	Configure the distribution group for cyclic operation mode. Subsequent calls try another first destination in a round-robin method. Default is not to use cyclic mode – always to start with the first configured destination(s).
3	*node*(svc-hunt)[*service-name*]#**max – concurrent** *max-concurrent*	Configures how many destinations shall be called at the same time. If you also configure a timeout, the first call is cleared and an additional call is made after that timeout. Thus only the specified number of destinations is ringing at the same time.
4	*node*(svc-hunt)[service-name]#**time-out** *timeout*	Configures a timeout in seconds after which one destination is dropped and a next destination is called.
5	*node*(svc-hunt)[service-name]#**route call dest-interface** *interface-name* OR *node*(svc-hunt)[service-name]#**route call dest-table** *table-name* OR *node*(svc-hunt)[service-name]#**route call dest-service** *service-name*	Adds a route to a destination. This is the interface, table or service that is tried to call during the distribution group's attempt to make calls to the destinations. The destination can either be an interface or you can route the call again to a routing table or directly to another service. This allows you cascading services.
6		Repeat step 5 to add additional hunting destinations.

Note If you specified the maximum number of concurrent destinations and the distribution group tried each destination, the final destinations ring until someone picks up one of the phones.

Note It does not make sense to configure the maximum number of concurrent destinations but no timeout, though the software does not prevent this configuration.

Distribution-Group Min-Concurrent Setting

A new command in the call-control's distribution-group service lets the user specify how many of the configured call destinations should be tried first: **min-concurrent <number>**. Together with the **max-concurrent** and **timeout** commands, you can configure the behavior of the call distribution. The distribution-group starts with *min-concurrent* calls and after the *timeout* one destination call is added until *max-concurrent* is reached. Then, again after the timeout, a call to the next destination is placed while the first destination call is dropped.

Mode: service distribution-group <name>

Step	Command	Purpose
1	[*name*]**(svc-dist)[***name***]# min-concurrent** <number>	Sets the minimum number of concurrent calls.

Example: To configure a distribution group that first rings on the phone#1 and then, after 5 seconds, on phone#1 and phone#2, enter the following commands:

```
node(svc-dist)[service-name]#min-concurrent 1
node(svc-dist)[service-name]#max-concurrent 2
node(svc-dist)[service-name]#timeout 3
node(svc-dist)[service-name]#route call dest-interface PHONE1
node(svc-dist)[service-name]#route call dest-interface PHONE2
```

Call-router 'limiter' Service

The call-router 'limiter' service limit offers a flexible technique to limit the maximum number of concurrent calls within the system. It also limits the call-setup rate within a system. Calls exceeding the defined limits can either be simply dropped or they can be handled differently. A call is counted as an active call as soon as the call-setup message reaches the limiter service. The call remains active until the signaling has completed tearing down the call.

In the figure is a call-router configuration which uses the limiter service to limit the maximum number of concurrent calls between SIP and ISDN to 20. In this scenario, if the limit is reached, any additional call received from sip will be dropped. If however an additional call arrives from ISDN, it will be forwarded to the special ISDN interface called 'voicemail'.

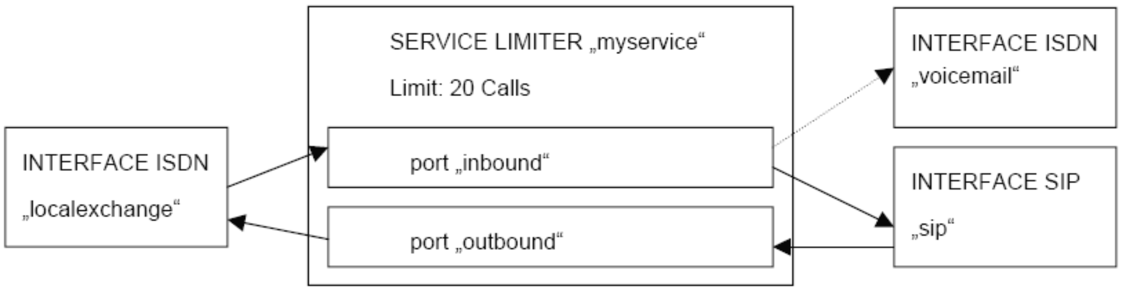


Figure 80. 'Limiter' service diagram

```
context cs switch
interface isdn localexchange
route call dest-service mylimiter.inbound
interface isdn voicemail
interface sip sip
bind gateway sip
route call dest-service mylimiter.outbound
service limiter mylimiter
max-calls 20
port inbound
route call dest-interface sip
exceed max-calls route call dest-interface voicemail
port outbound
route call dest-interface localexchange
```

Similarly the call-setup-rate could be limited by using the 'exceed max-call-rate' instead of the 'exceed max-calls' command in the limiter-port configuration mode. There is no limitation on the number of ports a limiter can have. You can create as many as you need for your application.

Priority Service

The service 'priority' can automatically free resources if a high priority call needs to be established while no resources are available. The service 'priority' can have multiple ports. You can assign a priority level for each port. This priority level defines the priority level of each call, which is received through the port. If a call with higher priority fails to be established, the service tries dropping lower priority calls to free resources for the higher priority call. Subsequently it tries to establish the higher priority call again. Figure 81 on page 510 is a typical application for this service in which non-.emergency calls are dropped to free resources for emergency calls.

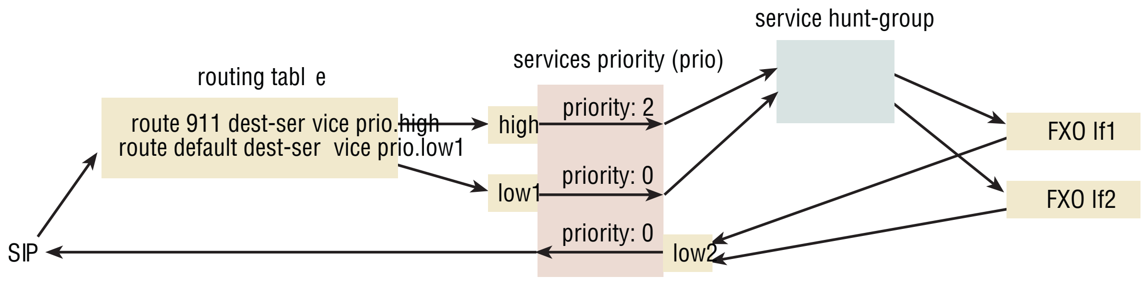


Figure 81. Priority service diagram

By default, the service drops any lower priority calls if a higher priority call fails. However, you have the option to limit the number of lower priority calls to be dropped if a higher priority call fails. It also blocks new lower priority calls for a configurable time after a higher priority call failed.

The following procedure demonstrates how to configure a priority service.

Mode: context cs

Step	Command	Purpose
1	***node***(ctx-cs)[ctx-***name***]# service priority <svc-name>	Create a priority service.
2	***node***(svc-prio)[svc-name]# port <port-name>	Create a port within the service.
3	***node***(port)[port-***name***]# route dest-....	Define the routing destination for calls received on this port.
4 (Optional)	***node***(port)[port-***name***]# priority	Define the priority for calls received through this port. (The default priority level is 0)
5	***node***(port)[port-***name***]# exit	Leave the port configuration mode
6		Repeat steps 2 to 5 for any additional calls you need to create.
7 (Optional)	***node***(svc-prio)[svc-name]# max-calls-to-drop <calls>	Define the maximum number of lower priority calls to be droppedif a higher priority call fails. (Default is to drop any lower priority calls.)
8 (Optional)	***node***(svc-prio)[svc-name]# quiesce-time <seconds>	Define the time for which lower priority calls are blocked after a higher priority call failed. (Default is 15 seconds.)
9 (Optional)	***node***(svc-prio)[svc-name]# retry-timeout <seconds>	Define the time for which the service waits after a higher priority call failed until it tries to establish it for a second time. This allows resources used by dropped lower priority calls to get available again. (Default is 3 seconds.)

Note Although this service improves the probability that higher priority calls can be established successfully, there is no guarantee that a higher priority task can be established successfully at any time.

CS Bridge Service—'VoIP Leased Line'

The circuit switch (CS) bridge service provides the functional ability to create a leased line between two FXS ports, with the FXS ports on different Patton devices. The call is point-to-point in an *always connected* state, also known as *nailed up*. This Call Control service is called *Bridge services*."

The application for this feature is when a constantly connected VoIP link is required between two FXS ports. It is as if you connected a regular telephone that is always off-hook to the FXS port. However it is not identical since the other end of the VoIP leased-line connection does not ring like in a PLAR application. There is no end-to-end call setup so no call-progress tones are required nor needed.

Here is another way to describe the application. A user has an FXO port in two remote locations and wants to connect these two locations over an IP network. Clearly the FXO port at a location must connect to an FXS port. But since the network between the two sites is IP, there needs to be a mapping of the information on the FXS–FXO link to a method of transporting the information over an IP network. Additionally, you do not want any ringing to occur when the connection is made; you simply want it to be connected, so the Patton devices and IP network operate transparently. See figure 82 to visualize the VoIP leased-line connection.

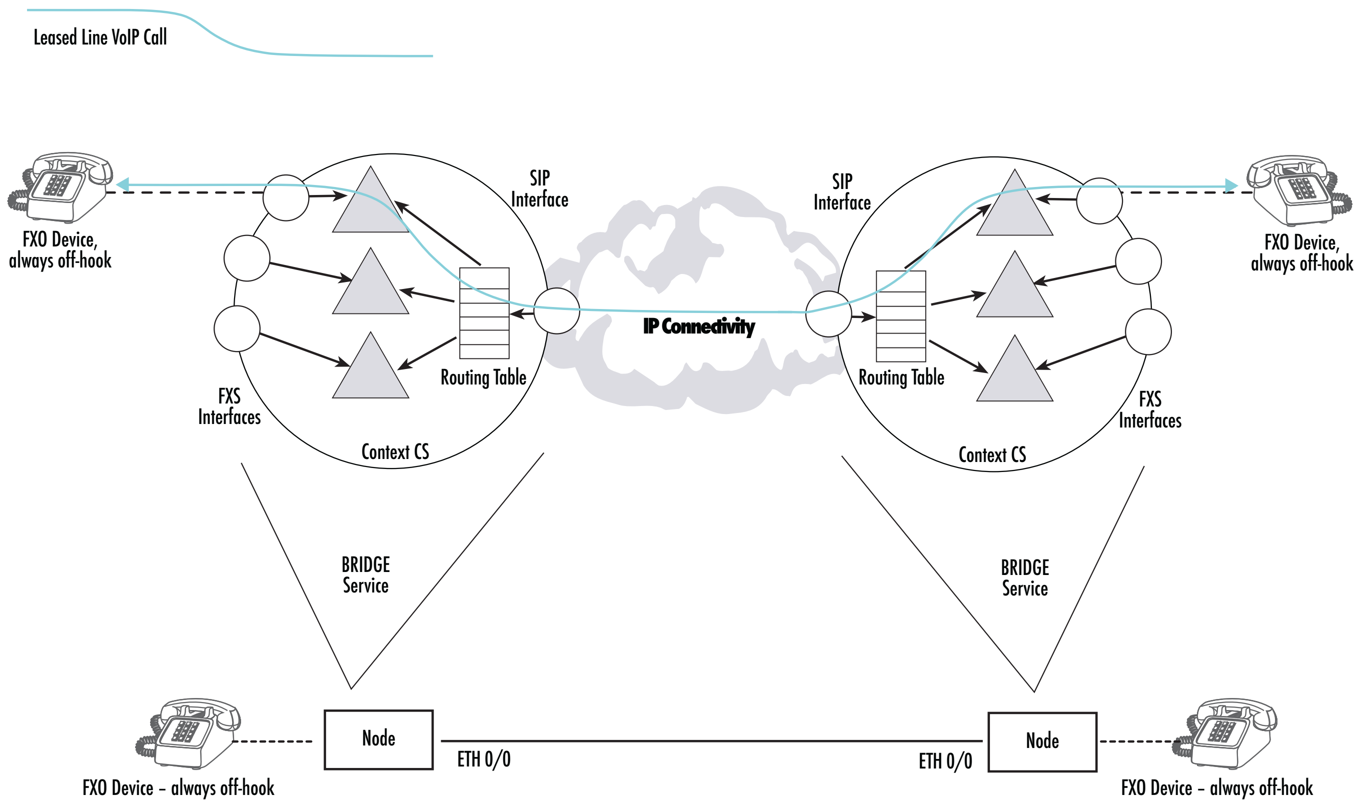


Figure 82. CS Bridge service—'VoIP Leased Line' diagram

Now we will describe the technical details and logical structure to implement this application. From the perspective of just one Patton device, the device makes two independent calls. These two calls are made from a logical structure, called a *Bridge Service*. The Bridge Service has two interfaces, the *listener* port and the *dialer* port (see figure 83 on page 513). Each of these interfaces is responsible for one of the two independent calls. The listener port terminates the “FXS call” and the dialer port terminates the “RTP call.”

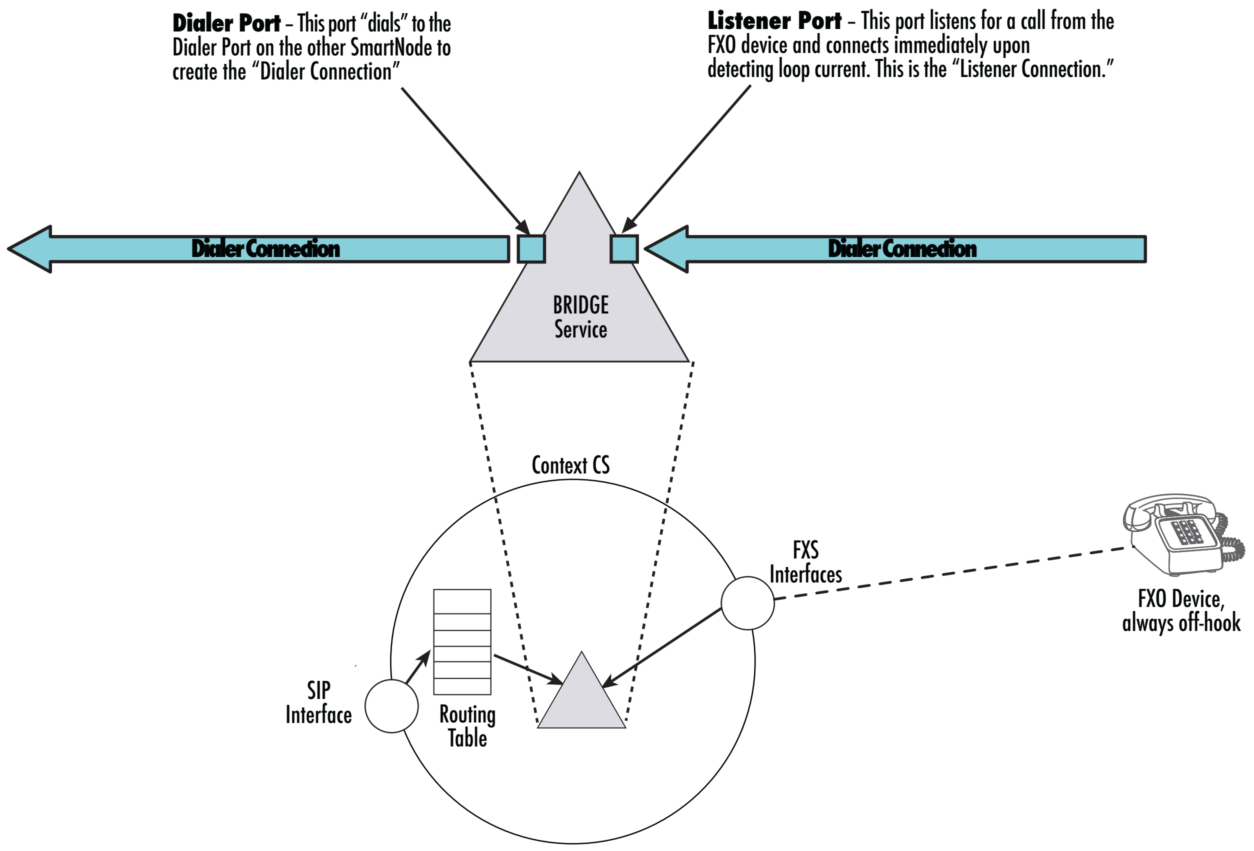

Dialer Port – This port “dials” to the Dialer Port on the other SmartNode to create the “Dialer Connection”
Listener Port – This port listens for a call from the FXO device and connects immediately upon detecting loop current. This is the “Listener Connection.”
Dialer Connection
Dialer Connection
BRIDGE Service
Context CS
FXS Interfaces
FXO Device, always off-hook
SIP Interface
Routing Table

Figure 83. Bridge services diagram

The listener port interface listens to the FXS interface of the device for an active FXS–FXO connection. It recognizes an active connection by detecting current in the FXS–FXO loop, and the *FXS call* is established. The dialer port interface attempts to make and keep an RTP connection session or *call* with a dialer port in a remote device. It is called an *RTP call*. This connection is over the IP network. The listener port and the dialer port both try to keep their individual calls up and operating at all times. However if the listener port loses its connection (that is, its call), the dialer port does not disconnect its RTP call but remains connected to the other device’s dialer port. Similarly, if the local dialer port loses its connection with the remote device’s dialer port, the device’s listener port does not disconnect its FXS call but remains connected to the FXO device. Though the calls operate independently, they operate over a single data path from end-to-end.

The dialer port is bound to the Routing Table which is subsequently bound to either a SIP interface. The routing table makes the connection to the proper FXS interface.

Mode: Context CS

Step	Command	Purpose
1	[*name*] **(ctx-cs)[switch]#[no] service bridge BRIDGE1**	Enters the bridge service configuration mode / deletes a bridge service.
2	[*name*] **(svc-bridg)[BRIDGE1]#port DIALER**	Creates a port on the service that can accept or spawn calls (the max number of ports is currently limited to two)
3	[*name*] **(port)[DIALER]# dial persistent 123 dest-interface REMOTE**	Configures the port to actively dial the called-party “123” to the destination REMOTE. This connection is kept open until the service is shut down, or a “no dial” command is issued. If the remote side terminates the connection, the port tries to reconnect. As soon as the connection is established, the call is connected to all other calls present on the service.
4	[*name*] **(svc-bridg)[BRIDGE1]#port LISTENER**	Creates a port on the service that can accept or spawn calls (the max number of ports is currently limited to two)
5	[*name*] **(port)[DIALER]# no dial**	Without entering a “dial” command, or by specifying “no dial”, this port does not dial, but listen for incoming calls. If a call comes in, it is automatically connected to all other calls present on the service.

Configuration Example:

```
context cs switch

  routing-table called-e164 DISPATCH
    route 1 dest-service BRIDGE1.dialer
    …

  service bridge BRIDGE1
    port listener
      mode listening
      no shutdown
    port dialer
      mode dial-persistent 1 dest-interface REMOTE
      no shutdown

interface sip REMOTE
    bind gateway sip
    route call dest-table DISPATCH
    remote 172.16.32.37

  interface fxs PHONE1
    route call dest-service BRIDGE1.listener
```

Configuring the Service Second-dialtone

If a call is routed into the service second-dialtone, the service establishes the call and plays a dial tone to the caller.When a call is entering the service it is first routed to the configured destination. In case the call cannot be placed, the service plays the second dial-tone or the configured announcement to the caller until the caller enters the first DTMF digit.

Caveat: For calls from the IP side, the service only works if the G711.alaw codec is chosen.

Mode: Context CS

Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]#**service second-dialtone *name***	Enters the service second-dialtone mode.
2	*node*(svc-2dt)[*name*]#**route call dest-interface|dest-service|dest-table *name***	Defines the call destination. If the call cannot be placed the second-dialtone is played.
3	*node*(svc-2dt)[*name*]#**route announcement dest-interface|dest-service|dest-table *name***	Instead of playing a dialtone locally, the service sets up a call to the configured message provider.
4	*node*(svc-2dt)[*name*]#**use profile tone-set *name***	Apply to use a alternative tone-set profile. If not configured the profile tone-set default will be used.

Deleting Call Services

To remove individual call services you can use the **no** form of the **service** command.

Procedure: To delete a call service

Mode: Context CS

Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]#**no service** *service-name*	Delete the service *service-name*. **Note:** You do not have to enter the type of the service when just deleting it. The type must only be specified when creating a service.

Example: Remove an entire mapping table

```
node(cfg)#context cs
node(ctx-cs)[switch]#no service HUNT-BRI
```

Activate the Call Router Configuration

Prior to activate the call router configuration you can show the whole context CS configuration and the entire call routing tables.

The call router configuration is activated as soon as the CS context comes out of shutdown (e.g. at boot time or by manually entering command **no shutdown**). You can modify the configuration at runtime; changes will be active after 3 seconds. Trinity offers a number of possibilities to monitor and debug the CS context and call router configurations. For more information refer to chapter 59, "Debug and Monitoring" on page 661.

Note It is not necessary to shutdown the CS context prior to making any configuration changes.

Procedure: To show and activate the call router configuration

Mode: Context CS

Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]#**show call-router config**	Show the actual call router configuration. This displays all routing and mapping tables in the current context CS. When you are inside a routing or mapping table configuration mode, only the current table is displayed.
2	*node*(ctx-cs)[*switch*]#**show running-config**	Show the whole running config includes the call routing tables
3	*node*(ctx-cs)[*switch*]#**debug call-router detail** *level*	Enable the call router debug monitor. Use level 1 for get informed about errors and increase the level up to 5 to track calls during route lookups.
4	*node*(ctx-cs)[*switch*]#**no shutdown**	Activate the whole CS context configuration including the call router configuration.
5	*node*(ctx-cs)[*switch*]#**show call-router status**	Show the actual call router status. This command can be used to examine whether or not the call router accepted all routing entries as entered in the configuration.

Note You must explicitly enter the **no shutdown** command to activate the call router.

Test the Call Router Configuration

After activating the call router configuration you can test the call router by simulating a route lookup as if a call is routed to a table. You have to execute the test call-router command and specify all necessary call properties together with the routing table you want to test.

Note You must activate the call router using the **no shutdown** command first.

Procedure: To test the call router configuration

Mode: Context CS

Step	Command	Purpose
1	*node*(ctx-cs)[*switch*]#**debug cr**	Enables the call router debug monitor. Chose level 5 to trace route lookups in detail.
2	*node*(ctx-cs)[*switch*]#**test call-router** *table-name* **[***property-type property-value***]**	Tests the routing or mapping table *table-name* with the specified call property. You can repeat the optional section multiple times and thus enter as many call properties as you want.

Example: Create and test a routing table

```
node(cfg)#context cs
node(cts-cs)[switch]#routing-table called-e164 TEST
node(rt-tab)[TEST]#route 1 dest-interface IF1
node(rt-tab)[TEST]#route 1[0-4] dest-interface IF2
node(rt-tab)[TEST]#route 11 dest-interface IF3
node(rt-tab)[TEST]#route 111T dest-interface IF4
node(rt-tab)[TEST]#route default dest-interface IF5
node(rt-tab)[TEST]#exit
node(ctx-cs)[switch]#no shutdown
node(ctx-cs)[switch]#debug cr
node(ctx-cs)[switch]#test call-router TEST called-e164 123
Parameters
==========

 Time:                                  2004-03-02T16:55:33<-- Time of the lookup
 Result:                                route-found-place-call<-- Lookup result
 Destination:                           IF2<-- Dest. Interface
 Timeout:                               0<-- Digit-Coll. TO

 Property Containers
 -------------------

  Properties

  Properties
   E164-Number:                         123 (String)<-- CdPN after lookup/change

  Properties

16:55:33  CR    > [switch] Routing-Lookup:
16:55:33  CR    >   Find best-matching called-element in table test
16:55:33  CR    >     01: Prefix Timeout Expression: E164-Number of 123 completely
(no timeout) matches 1
16:55:33  CR    >     02: Prefix Timeout Expression: E164-Number of 123 completely
(no timeout) matches 1[0-4]
16:55:33  CR    >     03: Prefix Timeout Expression: E164-Number of 123 does not
match 11
16:55:33  CR    >     04: Prefix Timeout Expression: E164-Number of 123 does not
match 111
16:55:33  CR    >     Selecting entry 2
16:55:33  CR    >   Execute all elements in table IF2
16:55:33  CR    >   Execute all elements in table route-found-place-call
16:55:33  CR    >   Lookup result: Route found; place call (timeout=0)
```

Example: Enterprise network with local breakout and IP carrier access

Consider the following Enterprise Network.

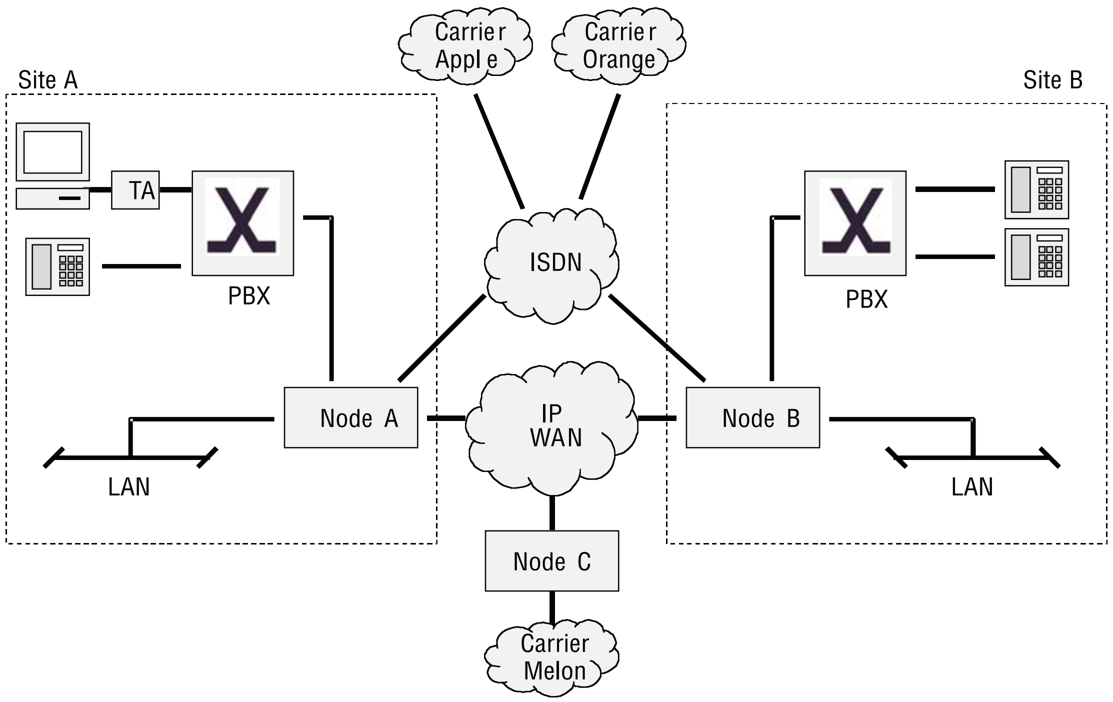


Figure 84. Call routing example network

Note The Patton devices in this Network may be owned and operated by the Company or by a Service Provider.

The goal of this scenario is to connect the two PBX of sites *A* and *B*. The two sites are connected to a broadband IP provider. The IP network is used to exchange data and voice calls between the two sites. On the IP network there is also a PSTN gateway (Node *C*) to an alternative voice carrier *Melon* that shall be used for most call destinations.

Sites A and B also have connections to the local ISDN network. This is called the local breakout connection. The local breakout is to be used as a fallback for ISDN data connections.

We assume the following:

- The number block for site A is 022 782 55 00 to 99
- The number block for site B is 033 665 2 000 to 999
- The Carrier Access Code (CAC) for *Apple* is 1055
- The Carrier Access Code (CAC) for *Orange* is 1066
- Carriers Apple, Orange and Melon do not support ISDN data calls (PC with ISDN Terminal Adapter behind PBX A)

- When calling through carrier *Melon* the CLI (calling party number) must not use the public number blocks of Site A and B
- Carrier *Orange* is to be used for national calls
- Carrier *Apple* is to be used for calls to mobile

The requirements for the call router can be summarized as:

1. Route ISDN data calls to the local breakout.
2. Route inter-site calls to the opposite Patton device (node A to node B and vice versa).
3. Route international calls to carrier *Melon.*
4. Provide a fallback for all VoIP calls on the local breakout.
5. Route local calls to the local breakout.
6. Route national calls to carrier *Orange.*
7. Route mobile calls to carrier *Apple.*
8. Calls from the PSTN, nodes B and C are forwarded directly to the PBX.

The remainder of this example will focus on the configuration for Node A. The configuration for Node B can be built accordingly. Node C has an even simpler configuration.

It is a good idea to specify the required call router elements and names before starting the configuration. A sketch may be helpful:

- Bearer capability table named *TAB-ISDN-SERVICE*, needed for requirement 1.
- Called party number table named *TAB-DEST-A*, needed for requirements 2, 3, 6 and 7
- CAC insertion for *Apple MAP-CAC-APPLE*, needed to add a carrier access code for *Apple*
- CAC insertion for *Orange MAP-CAC-ORANGE*, needed to add carrier access code for *Orange*
- CLI replacement for *Melon MAP-CLI-MELON*, needed to add carrier access code for *Melon*
- PSTN interfaces *IF-PBX-A* and *IF-LOCAL-BREAKOUT*, needed for requirements 4, 5 and 8.
- SIP interface *IF-NODE-C*, needed for requirement 3.

Figure 85 shows the corresponding CS Context and call router elements in node A:

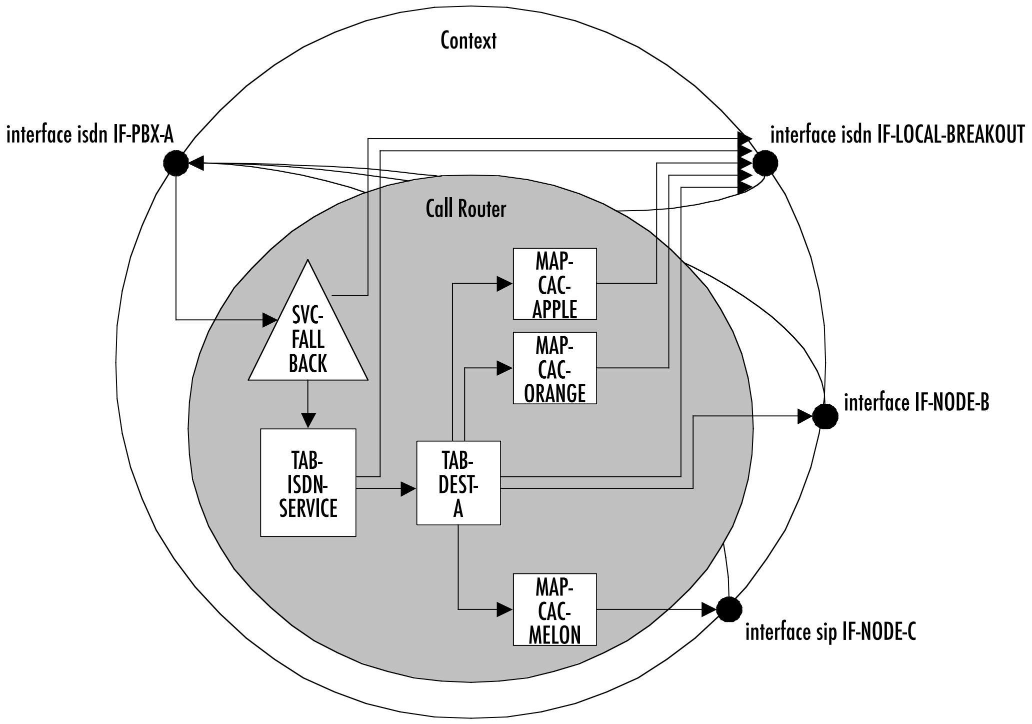


Figure 85. CS context and call router elements

We assume that the CS interfaces have already been created and configured. So we can start directly with the call router elements.

Since the command sequence is quite long it is useful to create the configuration offline and download it using TFTP.

Note In the following lines the prompt is omitted as in a configuration file and for better readability.

```
#--------------------------------------------------------------
# Call Router Config File
#--------------------------------------------------------------

context cs switch

#
# Hunt-group service "SVC-FALLBACK" to catch VoIP network errors
#

service hunt-group SVC-FALLBACK
  no cyclic
  timeout 6
  route call 1 dest-table ISDN-SERVICE
  route call 2 dest-interface LOCAL-BREAKOUT
```

```
#
# Bearer capability routing table “TAB-ISDN-SERVICE”
#

routing-table itc TAB-ISDN-SERVICE
  route unrestricted-digital dest-interface IF-LOCAL-BREAKOUT
  route default dest-table TAB-DEST-A

#
# Called party number routing table “TAB-DEST-A”
#

routing-table called-e164 TAB-DEST-A
  route 0 dest-interface IF-LOCAL-BREAKOUT MAP-CAC-ORANGE
  route 00 dest-interface IF-NODE-C MAP-CLI-MELON
  route 07[4-6] dest-interface IF-LOCAL-BREAKOUT MAP-CAC-APPLE
  route 0336652... dest-interface IF-NODE-B
  route default dest-interface IF-LOCAL-BREAKOUT

#
# Number manipulation “CAC-APPLE”; add prefix 1055
#

mapping-table called-e164 to called-e164 MAP-CAC-APPLE
  map (.%) to 1055\1

#
# Number manipulation “CAC-ORANGE”; add prefix 1066
#

mapping-table called-e164 to called-e164 MAP-CAC-ORANGE
  map (.%) to 1066\1

#
# Number manipulation “CLI-MELON”
# Truncate CLI to last 2 digits and add 08004455 prefix in front
#

mapping-table calling-e164 to calling-e164 MAP-CLI-MELON
  map .%(..) to 08004455\1
```

Prior to downloading this file you should make sure there are no other tables and functions in the call router.

```
node(ctx-cs)[switch]#copy tftp://172.16.36.20/configs/SRconf.cfg running-config
Download...100%
```

Now we have to enable advanced call routing for outgoing calls and basic interface routing for incoming calls: Calls arriving on the interface from the PBX are routed to the SVC-FALLBACK service while incoming calls from the other interfaces are routed directly to the IF-PBX-A interface.

```
node(ctx-cs)[switch]#interface isdn IF-PBX-A
node(if-pstn)[IF-PBX-A]#route dest-service SVC-FALLBACK
node(if-pstn)[IF-PBX-A]#exit
node(ctx-cs)[switch]#interface isdn IF-LOCAL-BREAKOUT
node(if-isdn)[IF-LOCA~]#route dest-interface IF-PBX-A
node(if-isdn)[IF-LOCA~]#exit
node(ctx-cs)[switch]#interface sip IF-NODE-C
node(IF-NODE-C)[IF-NODE-C]#route dest-interface IF-PBX-A
node(IF-NODE-C)[IF-NODE-C]#exit
```

The configuration is now complete. Prior to activating the configuration enable the call router debug monitor to check the loading of the call router elements.

```
node(cfg)#debug call-router
node(cfg)#context cs
node(ctx-cs)[switch]#no shutdown
02:14:30  CR    > Updating tables in 3 seconds...
02:14:33  CR    > [switch] Reloading tables now
02:14:33  CR    > [switch] Flushing all tables
02:14:33  CR    > [switch] Loading table 'TAB-ISDN-SERVICE'
02:14:33  CR    > [switch] Loading table 'TAB-DEST-A'
02:14:33  CR    > [switch] Loading table 'CAC-APPLE'
02:14:33  CR    > [switch] Loading table 'CAC-ORANGE'
02:14:33  CR    > [switch] Loading table 'CLI-MELON'
02:14:33  CR    > [switch] Loading table 'MAP-CAC-APPLE'
02:14:33  CR    > [switch] Loading table 'MAP-CAC-ORANGE'
02:14:33  CR    > [switch] Loading table 'MAP-CLI-MELON'
02:14:33  CR    > [switch] Loading table 'IF-LOCAL-BREAKOUT-precall-service'
02:14:33  CR    > [switch] Loading table 'IF-PBX-A-precall-service'
02:14:33  CR    > [switch] Loading table 'IF-NODE-B-precall-service'
02:14:33  CR    > [switch] Loading table 'IF-NODE-C-precall-service'
node(ctx-cs)[switch]#
```

Configuring Partial Rerouting

To save bandwidth and B-Channels, Trinity supports partial rerouting. Trinity accepts rerouting requests from PBX, which want to push-back a diverted or forwarded call with a Rerouting Request. Trinity can generate such a reroute request to push-back a looped call. Both, acceptation and emission of rerouting requests can be configured separately.

To prevent rerouting when a particular service is invoked in a call, services can be configured to allow or to forbid rerouting of calls of the service. In order that a call can be rerouted, all services participating in this call must allow rerouting. In addition rerouting emit must be configured on the interface where this call comes in and is routed out.

To accept or emit rerouting requests, see "Call reroute" on page 522.

To allow push-back, see "Allow push-back" on page 523.

Call reroute

The **call-reroute** commands enable acceptation and emission of rerouting requests.

Enable rerouting requests on ISDN. If a reroute is accepted, the participant who sends the reroute request is disconnected and the call is established from the Patton device to the new destination.

Mode: context cs/interface isdn

Step	Command	Purpose
1	[*name*]**(if-isdn)**[*interface*]**#[no] call-reroute accept**	Enables acceptation of rerouting requests from ISDN. Default is disabled.

Enable emission of rerouting requests on ISDN. To reroute a call, you must enter and leave the device through the same ISDN interface and every service invoked must allow push-back.

Mode: context cs/interface isdn

Step	Command	Purpose
1	[*name*]**(if-isdn)**[*interface*]**#[no] call-reroute emit**	Enables emission of rerouting requests from ISDN. Default is disabled.

Enable sending of "302 moved temporary" message on SIP. To reroute a call, you must enter and leave the device through the same SIP gateway and every service invoked must allow push-back.

Mode: context cs/interface sip

Step	Command	Purpose
1	[*name*]**(if-sip)**[*interface*]**#[no] call-reroute emit**	Enables emission of "302 moved temporarily" message on SIP. Default is disabled.

Allow push-back

The **push-back** command allows or forbids rerouting of a call, if the service is invoked.

Enable push-back – aaa service.

Mode: context cs/service aaa

Step	Command	Purpose
1	[*name*]**(svc-aaa)**[*service*]**#[no] allows-push-back**	Enables push-back of a call of this service. Default is disabled.

Enable push-back – bridge service.

Mode: context cs/service bridge

Step	Command	Purpose
1	[*name*]**(svc-brdg)**[*service*]**#[no] allows-push-back**	Enables push-back of a call of this service. Default is disabled.

Enable push-back – distribution-group service.

Mode: context cs/service distribution-group

Step	Command	Purpose
1	[*name*]**(svc-dist)**[*service*]**#[no] allows-push-back**	Enables push-back of a call of this service. Default is disabled.

Enable push-back – hunt group service.

Mode: context cs/service hunt-group

Step	Command	Purpose
1	[*name*]**(svc-hunt)**[*service*]**#[no] allows-push-back**	Enables push-back of a call of this service. Default is disabled.

Enable push-back – limiter service.

Mode: context cs/service limiter

Step	Command	Purpose
1	[*name*]**(svc-lim)**[*service*]**#[no] allows-push-back**	Enables push-back of a call of this service. Default is enabled.

Enable push-back – priority service.

Mode: context cs/service priority

Step	Command	Purpose
1	[*name*]**(svc-prio)**[*service*]**#[no] allows-push-back**	Enables push-back of a call of this service. Default is enabled.

Chapter 46 Global SIP Configuration

Chapter contents

Introduction

This chapter describes Trinity commands available under the global SIP configuration mode.

Enabling SIP Client ‘rport’ Support

Trinity supports the ‘rport’ parameter for SIP UAC according to RFC3581. This parameter forces a server that receives a request over UDP to send the response back to the IP-Address and Port from where it received the request and not to the Address/Port parameters available in the top most VIA header. This feature is useful for NAT-Traversal if the Patton device is located behind an external NAT.

This feature can only be enabled on a global base. All outgoing requests will contain the ‘rport’ parameter in the VIA header. The belonging command is available in the global sip configuration mode.

Mode: cfg

Step	Command	Purpose
1	[node]**(cfg)#sip**	Enters the global SIP configuration mode
2	[node]**(sip)#[no] rport**	Enables/disables the rport parameter in the VIA header of outgoing SIP requests. Default: disabled

Configuring SIP Peer Flood Blocking

Trinity's SIP stack can be configured to prevent SIP packet flooding. This is useful to defend against SIP attacks of any sort where many SIP packets are being received from the same peer within a short period of time. Disabled by default, this feature can be enabled using a simple “switch”-style command, and fine-tuned with a separate configuration command.

The SIP Peer Flood Blocking feature works on a per-peer basis, and fixes a limit on the number of packets that the stack will receive from any given peer. That limit is based on a time window combined with a packet count limit and a timeout. If, for any given peer, the number of packets received during the defined time window exceeds the limit, all further packets from that peer will be disregarded for the duration of the defined timeout.

The default parameters are dependent on the Patton device model, mostly on the CPU model and clock speed.

Mode: sip

Step	Command	Purpose
1	[node]**(sip)# [no] peer-flood-blocking**	Enables/disables the Peer Flood Blocking feature

Configuration:

Step	Command	Purpose
1	node(sip)#config-peer-flood-blocking [<window-size> <packet-count-limit> <timeout>]	Configures the Peer Flood Block feature, or resets the parameters to the (product-specific) default values. This occurs when the parameters are omitted; however, if the parameters are used, all three parameters must be specified. **Note** The window-size and timeout values are in milliseconds.

Limit Packets to Prevent SIP Overload Condition

It is possible to limit the maximum amount of incoming SIP packets, which are stored to be handled and processed later. This guarantees a responsive system even in an overload condition. It handles and parses still as many requests as possible, but discards the excess packets.

When more SIP requests are arriving than Trinity can handle, the overall system performance decreases because the internal resources are occupied for requests which cannot be handled in time. This causes re-transmissions of requests and adds additional overhead to the system. Therefore, it is better to drop packets early in the processing queue in order to avoid a condition where no successful request processing is possible.

The best overall performance in processing the highest number of successful call attampts with a continuously overloaded system was reached by setting this queue limit to a value of 8.

Mode: sip

Step	Command	Purpose
1	[node](sip)#[no] max-queued-packets <number of packets>	Limits the internal SIP queue to discard any packets when the configured number of packets is reached. Suggested: 8

Locking DNS Records for SIP Requests

DNS A and SRV records need to be selected according to RFC3263 and RFC2782, therefore allowing load balancing. However, the defined algorithm is not desirable in all situations, e.g.:

- A SIP request (INVITE, REGISTER, SUBSCRIBE) is sent to a SIP server which requires authentication. The next SIP request containing authentication credentials is potentially sent to another server based the new DNS record list.
- A re-INVITE can also be sent to another server than the first INVITE.

In situations where this behavior is undesired, Trinity provides a global configuration option to lock the DNS record list. This option is enabled by default and affects all relevant SIP requests: INVITE, REGISTER and SUBSCRIBE.

Mode: sip

Step	Command	Purpose
1	[node]**(sip)[0/0]#[no] lock-dns-record**	Enable locking of DNS records Default: enabled

SIP Request URI Length Limitation

This feature allows blocking incoming SIP request whose URI length exceeds a user specified value. Limiting the request URI length has device wide validity. The configured number of characters is applied for all incoming SIP requests on the whole device. Therefore the max-request-uri-length is located in the global sip configuration mode. By default the request URI length is unlimited.

Mode: sip

Step	Command	Purpose
1	node**(sip)#[no] max-request-uri-length <length>**	Enables/Disables request URI length limitation. The given length specifies the maximum number of characters including all URI parameters. Default: disabled

Chapter 47 SIP Overload Configuration

Chapter contents

Introduction

This chapter describes how to configure the device's behavior in case of an overload situation, in particular, how SIP calls shall be treated if the device is operated outside its nominal specification. In short, if the CPU load rises above a critical margin, new SIP calls are rejected. The same happens if the available RAM drops below a critical baseline.

Note The device automatically blocks new SIP calls in an overload situation by default and is pre-configured with optimal margins that define when the device is overloaded. Only experts should change this configuration. If you are unsure about whether or how to configure the SIP overload feature, please contact Patton's support team.

System Overview

As depicted in figure 86, the behavior of SIP in an overload condition depends on a state profile, which itself is based on system variables and custom event expressions (see Chapter 11, "Programmable System-Event Configuration" on page 111).

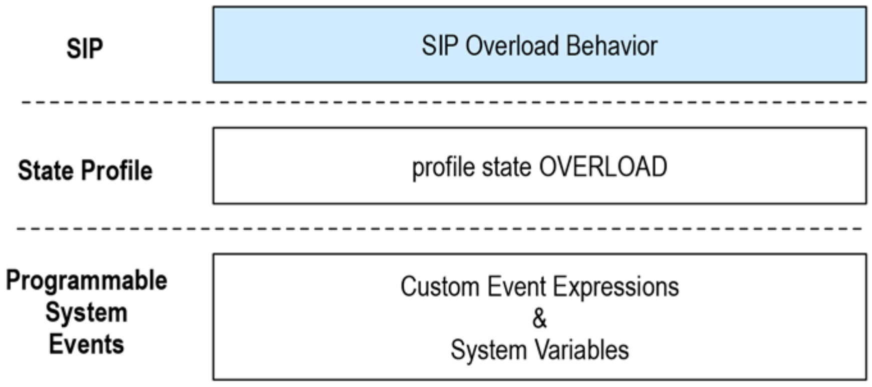

SIP
SIP Overload Behavior
State Profile
profile state OVERLOAD
Programmable System Events
Custom Event Expressions & System Variables

Figure 86. The SIP overload behavior uses the OVERLOAD state profile

From bottom to top these three blocks offer the following services:

- **Programmable System Events**: Trinity exposes many subsystem states as system variables (e.g. the link state of IP interfaces (**ip.ctx:***ctx-name***.if:***if-name***.up**), whether the time has been synchronized over NTP (**ntp.sync**), whether the system is up (**sys.up**), the available RAM (**sys.ram.avail**), etc.). The user may create custom expressions that combine system variables and other expressions with the *expression* command, e.g. *expression UPNSYNC "sys.up && ntp.sync"*. Whenever a system variable or custom expression changes its value a notification event is sent to all interesting applications, e.g. to the state profiles.
- **State Profile**: A state profile builds a simple state machine from system variables and custom expressions. Change-notification events from those variables are used to describe the transitions between its states. The device contains a built-in state profile called *OVERLOAD*. This overload state machine is in state *NORMAL* if the device operates according its nominal specification. The state machine transitions to the *WARNING* state if the CPU utilization was pretty high for a minute. The state machine further transitions to the *CRITICAL* state if the CPU utilization was almost 100% for a minute.

- **SIP Overload Behavior**: On top of these two components the SIP overload behavior links to a state profile (to the *OVERLOAD* profile by default) and defines the behavior of the SIP user agent in each state (*NORMAL*, *WARNING*, and *CRITICAL*). By default new SIP calls are rejected in the *WARNING* state whereas most SIP requests are rejected in the *CRITICAL* state. Only requests to close a call are passed in the latter state. See table 33 for more details.

This chapter explains how to configure the SIP overload behavior based on an existing state profile. See Chapter 47, "SIP Overload Configuration" on page 529 for information on how to change the conditions for the state transitions or how to create a new state profile.

Default Behavior

The following table lists the default behavior of the SIP user agent in any of the overload states:

Table 33. Default SIP overload behavior

State	Behavior of the SIP user agent
NORMAL	The Patton device accepts all incoming SIP messages and places all outgoing SIP calls.
WARNING	The Patton device rejects all incoming **INVITE, OPTIONS, PING, REGISTER**, and **SUBSCRIBE** requests by sending back a 503 Server Out Of Resources response. The Patton device also rejects all incoming responses to previous **INVITE, OPTIONS, PING, REGISTER**, and **SUBSCRIBE** requests sent by the Patton device. The Patton device does not place outgoing SIP calls anymore.
CRITICAL	The Patton device rejects all incoming SIP requests. Only requests required to drop a call are passed: **ACK, BYE, CANCEL**. The Patton device rejects all incoming SIP responses. Only incoming responses to the following requests are still passed to the user agent application: **ACK, BYE, CANCEL**. As in the WARNING state the Patton device does not place outgoing SIP calls anymore.

If the default configuration does not suit your needs please follow the procedure outlined in the next section to change the overload behavior.

Rejecting SIP messages might look strange. In an overload situation however, if the device is operated outside its nominal specification, rejecting SIP messages makes sure that the following properties are retained:

To reject calls if the CPU load is too high allows the operator to still re-configure and monitor the device if the entire CPU capacity is used for other tasks (e.g. handling SIP calls). Each SIP message requires some CPU power to be processes. If the rate of processed SIP request becomes too high, the SIP user agent would consume all CPU power and the user interfaces (CLI, WebUI) would be slowed down drastically. By rejecting SIP calls if the CPU utilization is too high the device stays responsive to management operations.

Mode of operation

There are two hooks in the SIP user agent that filter all incoming messages and outgoing calls (see figure 87 on page 532). All incoming SIP messages have to pass the *inbound SIP message filter* before they reach the SIP gateway (user agent). All outbound SIP calls routed by the call-router to a SIP interface have to pass the *outbound SIP call filter*. Whether or not those two filters permit or deny the message/call to pass is configured in the sip /

overload CLI configuration mode. If you deny INVITE requests, incoming INVITE messages as well as outgoing calls are rejected. If you deny SIP requests other than INVITE, only those received requests are affected (no outbound message filter is in place).

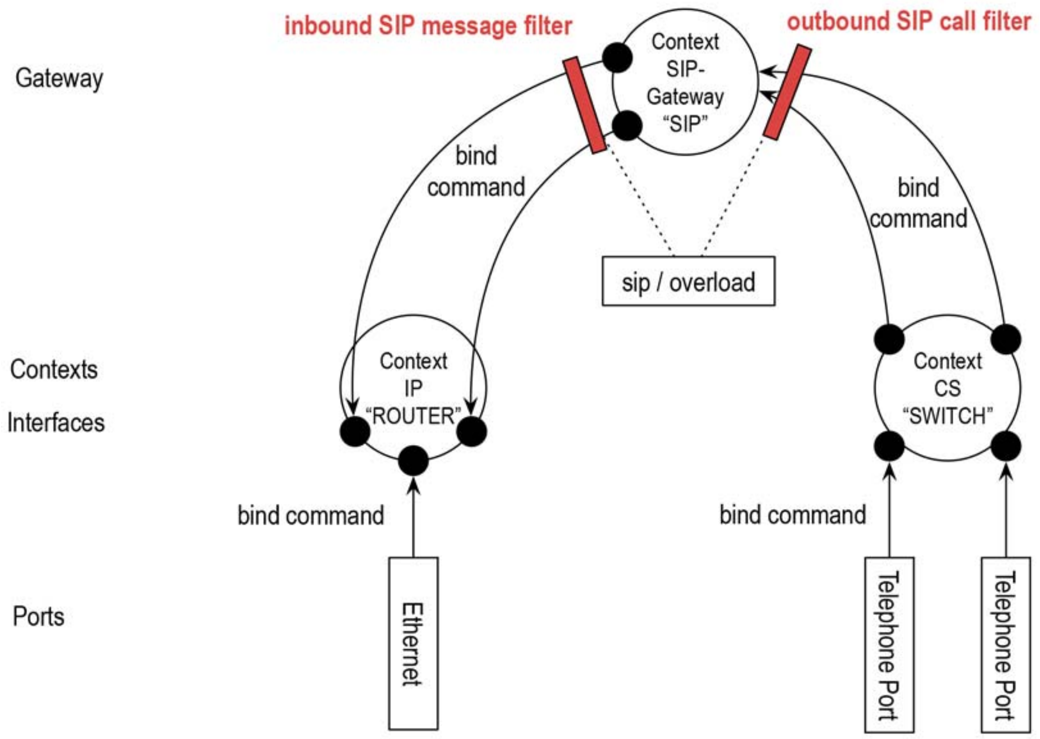


Figure 87. SIP Overload configuration and message flow diagram

Figure 88 on page 533 depicts in detail what happens to an incoming SIP request or an outgoing SIP call when it reaches the inbound or outbound SIP message or call filter, respectively. The SIP overload configuration consists of a set of Access Control Lists (ACL) for SIP methods (INVITE, BYE, etc.). There is a separate SIP ACL for every state of the used state profile. Thus if you are using the default *OVERLOAD* state profile there are separate SIP ACLs for the three states *NORMAL, WARNING*, and *CRITICAL*. A SIP ACL consists of a set of rules each describing whether a set of SIP methods shall be permitted (passed) or denied (rejected).

A SIP message is handed over to the ACL that corresponds to the current state of the used state profile. For example if the device is in *WARNING* state, the first rule rejects the message if it is an INVITE, OPTION, PING, REGISTER, or SUBSCRIBE request or a response to such a request. Since this is the only rule in the *WARNING*-ACL the message is permitted (passed to the user agent). Actually, all message not explicitly mentioned in the SIP ACL are permitted. If you want to deny all un-mentioned messages in a state you have to explicitly permit some messages and append a deny-all rule at the end (see the SIP ACL for the *CRITICAL* state).

Note The SIP ACL works in a similar way than the Access Control List for data packets (see Chapter 33, "Access Control List Configuration" on page 331) but solely operates on incoming SIP packets after they have passed the data ACL.

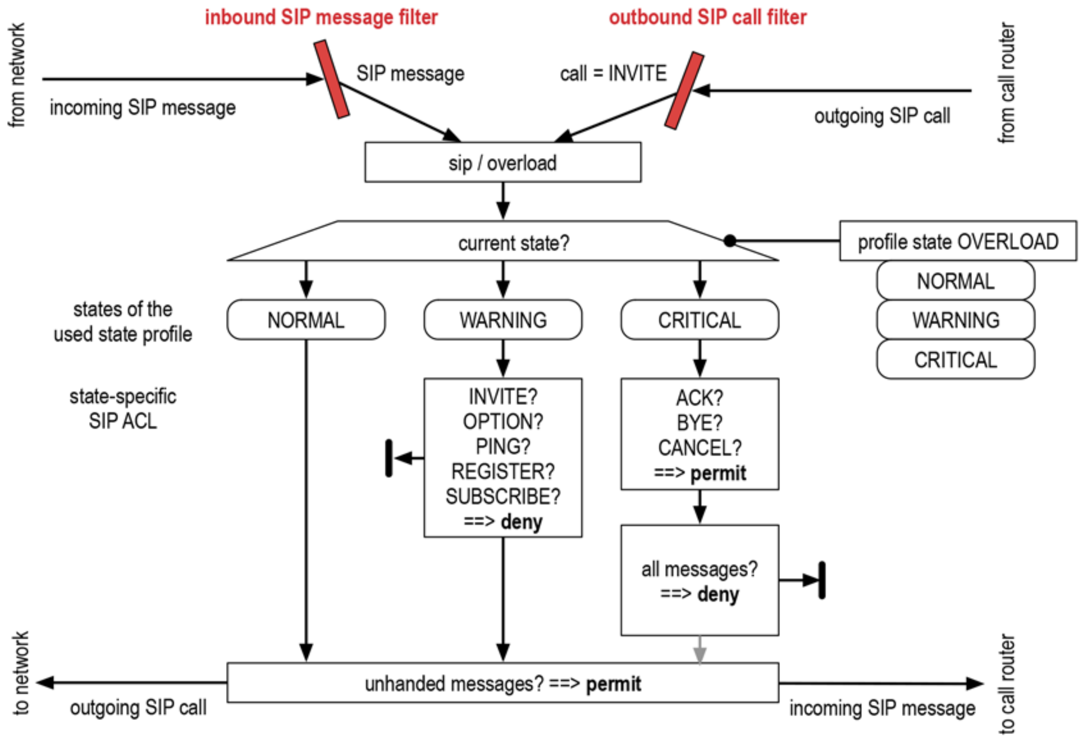


Figure 88. State-Based SIP Access Control Lists (ACL)

SIP overload configuration task list

To change the default behavior of the SIP user agent in an overload condition, perform one of the tasks in the following sections.

- "Disable SIP message rejection"
- "Change the list of SIP messages rejected in an overload situation" on page 534
- "Reset the SIP overload behavior to the system defaults" on page 534
- "Display the behavior of SIP in an overload situation" on page 535
- "Debug the behavior of SIP in an overload situation" on page 537

Disable SIP message rejection

As stated above rejecting SIP messages is a useful approach to make sure that the device is still responsive if operated outside its nominal specification. If you want to disable SIP message blocking nevertheless please follow the procedure below:

Mode: Configure

Step	Command	Purpose
1	***node*(cfg)#sip**	Enter the global SIP configuration mode.
2	***node*(sip)#overload**	Enter the SIP overload configuration mode.

Step	Command	Purpose
3	***node*(overload)#no use profile state**	Enter the SIP overload configuration mode.

The use *[no] profile state* command enables or disables the SIP overload feature; there is no explicit shutdown command.

Change the list of SIP messages rejected in an overload situation

The following procedure shows how to modify the SIP ACL that is passed by SIP messages in a specific state. Steps 4 and 5 may be repeated to add other states or rules in those states.

Mode: Configure

Step	Command	Purpose
1	***node*(cfg)#sip**	Enter the global SIP configuration mode.
2	***node*(sip)#overload**	Enter the SIP overload configuration mode.
3	***node*(overload)#use profile state *name***	Link the SIP user agent to a state profile that defines the overload condition. From now on all incoming SIP messages are filtered by the ACL configured below. By default, the *OVERLOAD* state profile is used.
4	***node*(overload)#state *name***	Enter the ACL mode of a specific state. The *name* parameter must match the name of a state defined by the state profile linked in step 3.
5	***node*(state)[*name*]#(permit|deny) (all|*sip-method**)**	Append a SIP ACL rule for the current state, which is to permit or deny packets of the listed *sip-methods* or of all messages.

Example: Drop all but INVINTE messages in the CRITICAL state.

```
node>enable
node#configure
node(cfg)#sip
node(sip)#overload
node(overload)#use profile state OVERLOAD
node(overload)#state CRITICAL
node(overload)#no deny 2   # remove existing rule
node(overload)#no permit 1 # remove existing rule
node(overload)#permit INVITE
node(overload)#deny all
```

Reset the SIP overload behavior to the system defaults

If you have experimented with either changing the conditions of the OVERLOAD state profile or changed the SIP ACLs in the sip/overload configuration mode, the following procedure re-establishes the system default behavior.

Mode: Configure

Step	Command	Purpose
1	***node*(cfg)#reset profile state OVERLOAD**	Reset the *OVERLOAD* state profile to its default states and conditions.
2	***node*(cfg)#sip**	Enter the global SIP configuration mode.
3	***node*(sip)#reset overload**	Reset the SIP overload behavior (linked profile and SIP ACLs for all states) to its default configuration.

Example: Reset both the default OVERLOAD state profile and the SIP overload behavior to the system-default configuration.

```
node>enable
node#configure
node(cfg)#reset profile state OVERLOAD
node(cfg)#sip
node(sip)#reset overload
```

Display the behavior of SIP in an overload situation

The following CLI command displays the configuration and statistics of the SIP overload feature:

Mode: Operator exec

Step	Command	Purpose
1	***node*>show sip overload**	Displays the linked state profile, the current state and an action table for each SIP message, whether the message will be permitted or denied in each state.

Example: Display information about the current SIP overload behavior

```
node>show sip overload
SIP Overload Behavior
=====================

  State Profile:                           OVERLOAD
  Current State:                           NORMAL
  Status:                                  enabled

  Behavior
  --------

   State: CRITICAL
   ---------------

     Configured Rules
     ----------------
       permit: ACK, BYE, CANCEL
       deny:   ACK, BYE, CANCEL, INFO, INVITE, MESSAGE, NOTIFY,
               OPTIONS, PING, PRACK, PUBLISH, REFER, REGISTER,
               SERVICE, SUBSCRIBE, UPDATE

     Compiled Action Table
```

```
    ---------------------
      ACK:                              permit
      BYE:                              permit
      CANCEL:                           permit
      INFO:                             deny
      INVITE:                           deny
      MESSAGE:                          deny
      NOTIFY:                           deny
      OPTIONS:                          deny
      PING:                             deny
      PRACK:                            deny
      PUBLISH:                          deny
      REFER:                            deny
      REGISTER:                         deny
      SERVICE:                          deny
      SUBSCRIBE:                        deny
      UPDATE:                           deny

  State: WARNING
  --------------

    Configured Rules
    ----------------
      deny:  INVITE, OPTIONS, PING, REGISTER, SUBSCRIBE

    Compiled Action Table
    ---------------------
      ACK:                              permit
      BYE:                              permit
      CANCEL:                           permit
      INFO:                             permit
      INVITE:                           deny
      MESSAGE:                          permit
      NOTIFY:                           permit
      OPTIONS:                          deny
      PING:                             deny
      PRACK:                            permit
      PUBLISH:                          permit
      REFER:                            permit
      REGISTER:                         deny
      SERVICE:                          permit
      SUBSCRIBE:                        deny
      UPDATE:                           permit

 Statistics
 ----------

   Incoming SIP Requests
   ---------------------
     ACK:    Rx:  35 | Permitted:  35 | Denied: .
     CANCEL: Rx:  36 | Permitted:  36 | Denied: .
     INVITE: Rx:  66 | Permitted:  66 | Denied: .

   Incoming SIP Responses
   ----------------------
```

```
    CANCEL: Rx:  35 | Permitted:  35 | Denied: .
    INVITE: Rx: 160 | Permitted: 160 | Denied: .

  Outgoing SIP Requests
  ---------------------
    INVITE: Rx: 64 | Permitted:   64 | Denied: .
```

The first section shows the linked *State Profile* and the *Current State* of the linked state machine.

The second section lists all states mentioned in the SIP overload configuration. For each state the configured SIP ACL rules are printed. In addition the show command displays the compiled message table that assigns each SIP request an action (permit or deny).

Finally, the third section displays statistics about how many messages have been received and how many of those have been permitted or denied. *Incoming SIP Requests* refer to request messages filtered at the inbound SIP message filter hook (see figure 88 on page 533) whereas *Incoming SIP Responses* are responses received at the same hook. *Outgoing SIP Requests* refer to outbound SIP calls received from the call-router that are handled on the outbound SIP call filter.

Note If the SIP overload feature is disabled (*no profile state* command has been entered) the SIP packets are not counted anymore.

Debug the behavior of SIP in an overload situation

Each SIP message that is analyzed by one of the SIP message filters is logged to the sip-signaling monitor (detail level 3).

Mode: Operator exec

Step	Command	Purpose
1	***node*>debug call-signaling [detail *level* | full-detail]**	Enables the real-time debugger that prints information about SIP call-signaling events. The SIP message filters will log to this monitor when permitting or denying a SIP message.

Example: Observe how an incoming SIP call is rejected in an overload situation (state CRITICAL).

```
node>debug call-signaling detail 3
18:47:00.000  SIP-SIG  # [Overload] Received INVITE request in state CRITICAL
18:47:00.000  SIP-SIG  # [Overload]   deny SIP packet
18:47:00.000  SIP-SIG  # [Overload] Received ACK request in state CRITICAL
18:47:00.000  SIP-SIG  # [Overload]   permit SIP packet
18:47:00.000  SIP-SIG  # [STACK] UNMATCHED ACK REQUEST
```

Note The *UNMATCHED ACK REQUEST* message indicates that the SIP stack received an ACK message without receiving an INVITE message before. This is because we denied INVITE requests but permitted ACK requests, a behavior required for properly terminating calls in an overload situation.

Chapter 48 SIP Interface Configuration

Chapter contents

Introduction

This chapter provides an overview of SIP interfaces used by context SIP gateways and describes the specific tasks involved in their configuration. This chapter does not explain the basic configuration steps required for all CS interfaces. See Chapter 41, "CS Interface Configuration" on page 416 for information about configuring basic CS interfaces.

Within the CS context, a SIP interface is a special type of CS interface providing call routing for incoming and outgoing calls to and from the context SIP gateway (see figure 89).

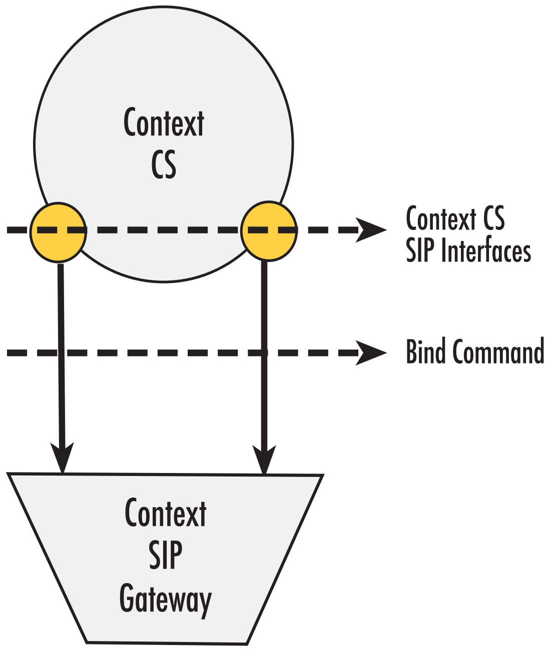


Figure 89. SIP interfaces on the CS context

A SIP interface is a CS interface type that is responsible for the address translation between the SIP environment and the call control. It provides configurable translation rules that define which parameter of which SIP header has to be translated to call control properties and vice versa. In addition, it offers VoIP settings and the possibility to configure SIP supplementary services like Call Transfer, Call Reroute or Session Timer. All SIP interfaces must be explicitly bound to a context SIP gateway. Calls that are routed from the Context CS to one of the SIP interfaces will be forwarded for call establishment to the context SIP gateway to which the SIP interface is bound. All of the parameters configured in the SIP interface will be applied to the forwarded call.

SIP Interface Configuration Task List

This section describes the configuration tasks for SIP interfaces listed below. You must at least perform the tasks that are not marked as optional to define a working SIP interface. The optional tasks are only required in advanced configurations. Before you can start with the SIP interface specific configuration tasks, you need to create the SIP interface and define the routing for it as defined in Chapter 41, "CS Interface Configuration" on page 416.

- Binding the interface to a context SIP gateway (see page 540)
- Configure a remote host (see page 541)
- Configure a local host (Optional) (see page 541)
- Using an alternate VoIP profile (Optional) (see page 542)
- Using an alternate SIP profile (Optional) (see page 542)
- Using an alternate Tone-Set profile (Optional) (see page 543)
- Configuring early call connect /disconnect (Optional) (see page 543)
- Early Media Behavior (see page 544)
- Configuring address translation (Optional) (see page 544)
- SIP REFER Transmission (& ISDN Explicit Call Transfer support) (Optional) (see page 548)
- AOC over SIP (Optional) (see page 549)
- Enable the session timer (Optional) (see page 550)
- Enable the SIP penalty-box feature (Optional) (see page 550)
- Initiating a new SIP session for redirected SIP calls (Optional) (see page 551)
- Configure the SIP hold method (Optional) (see page 551)
- Configuring PRACK for Reliable Provisional Responses (optional)
- Configuring History-Info in SIP (see page 554)
- Enabling NAT Traversal for SIP INVITE Messages (see page 554)
- Accepting Re-invite After Reboot (see page 554)

Binding the Interface to a SIP Gateway

Every SIP interface must be explicitly bound to a context SIP gateway. It defines the transport interfaces to be used to send requests and also from where it expects to receive incoming calls.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)[*if-name*]# [no] bind context sip-gateway** *gw-name*	Binds the interface to a context SIP gateway. The no form of the command removes an existing binding. Default: none

Configuring a Remote Host

The remote host parameter is used to build the host part of the To-Header-URI and the Request-URI. All calls forwarded to the context SIP gateway through this SIP interface will be sent to that host unless an outbound proxy has been configured for the outgoing Request-URI. In this case, all requests will be sent to the specified proxy independent of the Request-URI. For details about proxy configuration, see Chapter 44, "Location Service" on page 434.

The remote host parameter may contain a Domain Name, a Full Qualified Domain Name or an IP Address. If domain names are used, do not forget to configure the DNS client for address resolving. See Chapter 27, "DNS Configuration" on page 279 for more information.

An optional port number can be entered on this command. It specifies the destination port of the requests to be sent. If no port has been defined, the default SIP signaling port 5060 will be taken.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)**[if-name]# **[no] remote** *<hostname>* **[port]**	Specifies the remote host name and port. The no form of the command removes a configured host name. Default Hostname: none Default Port: 5060

Managing trusted hosts

A list of trusted remote peers can be configured on SIP interfaces. If configured, only connections with peers in that list will be accepted. The list may contain IP-addresses or FQDNs or the keyword remote. If no trusted peers are configured all connections are accepted.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)**[if-name]# **[no] trust { remote | <ip-address> | <domain-name> }**	Adds/removes a host to/from the trusted host list. Enter either an ip-address, a fully qualified domain name or the keyword *remote.*The *remote* keyword will add whatever is configured in the SIP interface's remote command.

Configuring a Local Host (Optional)

The local host parameter is used to build the host part of the From-Header-URI. Optionally, a port can be entered. If no port has been specified, it will not appear in the From-Header-URI. This command is optional in a SIP network setup where no DNS names are involved and only IP addresses applied. If no local host name has been specified, the IP address of the outgoing IP interface will be taken as host part of the From-Header-URI.

There are some exceptions where a local host name must be entered. One exception is if the bound gateway has two transport interfaces and the From-Header-URI call outbound properties have been configured. Such a call outbound property is the outbound proxy. To find the right identity in the location service, the From-Header-URI must exist. But, if there is more than one transport interface, it is not clear which one will be used to send the request because the proxy influences the routing. In such a case, it is recommended that the user configures the local host name and knows which IP interface the request should be sent over.

The SIP server expects its own IP address in the From-Header-URI because of security or routing reasons. This can be solved by manually entering a local host name.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)**[*if-name*]**# [no] local** <*hostname*>**[port]**	Specifies the local host name and port. The no form of the command removes a configured host name. Default Hostname: none Default Port: none

Using an Alternate VoIP Profile (Optional)

A VoIP profile is a container for all data path-related settings on VoIP connections. The predefined default profile exists persistently in the system and it is preconfigured with proper default parameters. It will always be taken if no other profile has been specified. This command allows the user to specify an alternate VoIP profile to use for all calls routed over this interface. For details about VoIP profile configuration see Chapter 38, "VoIP Profile Configuration" on page 365.

When a profile has been specified on the interface, it is possible to provide a special one for specific identities or group of identities. It can be configured in location service identities for call inbound and call outbound. See Chapter 44, "Location Service" on page 434 for details about identities and VoIP profile configurations. The identity lookup to find a possible configured VoIP profile will always be applied to the remote SIP URI. For outgoing calls, the SIP Request-URI will be taken. For incoming calls, the SIP From-URI will be taken.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)**[if-name]**#use profile voip** <*profile-name*>	Defines an alternate VoIP profile to be used for this SIP interface. Default: default

Using an Alternate SIP Profile (Optional)

The SIP profile contains a cause and reason mapping from SIP notation to the call control notation and vice versa. The predefined default profile exists persistently in the system and it is preconfigured with proper default mappings. It will always be taken if no other profile has been specified. This command allows the user to specify an alternate SIP profile to use for all calls routed over this interface. For details about SIP profile configuration see Chapter 37, "SIP Profile Configuration" on page 359.

When a profile has been specified on the interface, it is possible to provide a special one for specific identities or group of identities. It can be configured in location service identities for call inbound and call outbound. See Chapter 44, "Location Service" on page 434 for details about identities and SIP profile configurations. The identity lookup to find a possible configured SIP profile will always be applied to the remote SIP URI. For outgoing calls, the SIP Request-URI will be taken. For incoming calls, the SIP From-URI will be taken.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)**[if-name]**#use profile sip** *<profile-name>*	Defines an alternate SIP profile to be used for this SIP interface. Default: default

Using an Alternate Tone-Set Profile (Optional)

A Tone-Set profile contains a mapping of call-progress tones to defined tone sequences. The predefined default profile exists persistently in the system and it is preconfigured with the Swiss standard call-progress tone mapping. It will always be taken if no other profile has been specified. This command allows the user to specify an alternate Tone-Set profile to be use for all calls routed over this interface. For details about Tone-Set profile configuration see Chapter 42, "Tone Configuration" on page 424.

When a profile has been specified on the interface, it is possible to provide a special one for specific identities or group of identities. It can be configured in location service identities for call inbound and call outbound. See Chapter 44, "Location Service" on page 434 for details about identities and Tone-Set profile configurations. The identity lookup to find a possible configured Tone-Set profile will always be applied to the remote SIP URI. For outgoing calls, the SIP Request-URI will be taken. For incoming calls, the SIP From-URI will be taken.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)**[if-name]**#use profile tone-set** *<profile-name>*	Defines an alternate Tone-Set profile to be used for this SIP interface. Default: default

Configuring Early Call Connect/Disconnect (Optional)

Normally, SIP calls are fully connected by sending a 200 OK response to the INVITE request, if the called party answers the call. Any call progress tones or announcements are transmitted in the early SIP dialog. However, there are several SIP user agents that do not support media to be transmitted or received in an early dialog. To solve this problem, it is possible to connect the SIP call using a 200 OK response to the initial INVITE request as soon as inbound information is available. This will allow any SIP user agent to receive precall inbound information.

Early call disconnect suppresses busy tones (e.g. disturbing a telephone conference) and post-call announcements by sending a BYE message to the remote SIP user agent when the connected terminal hangs up (ISDN: when Disconnect message is received; analog line: when busy tone is detected, loop current is interrupted, or battery voltage is reversed).

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)**[if-name]**#[no] early-connect**	Enables/Disables early call connect Default: disabled

Step	Command	Purpose
2	**[*node*](if-sip)**[if-name]**#[no] early-disconnect**	Enables/Disables early call disconnect Default: disabled

Enable/Disable Early-proceeding on SIP Interface

In SBC scenarios, it is possible that multiple provisional responses (1xx) are not correctly forwarded. This can be corrected by *disabling* the early-proceeding parameter on SIP interfaces. In case of PSTN gateway scenarios, it is preferable to leave the early-proceeding parameter set to *enabled* on SIP interfaces.

Mode: Interface SIP

Step	Command	Purpose
1	**[node](if-sip)**[if-name]**# [no] early-proceeding**	Switches to proceeding state without receiving any provisional response. Default: enabled

Early Media Behavior

This command allows controlling SDP announcements in provisional SIP responses. It is possible to prevent any announcement (no early-media emit) or to instruct the system to do it automatically. In auto mode, SDP announcement is done based on call properties.

Mode: Interface SIP

Step	Command	Purpose
1	**device(if-sip)[name]#[no] early-media emit {auto}**	Defines SDP announcement procedure for provisional responses. Default: auto

Configuring Address Translation (Optional)

Mapping call-control properties in SIP headers

This functionality specifies rules for building the SIP headers for outgoing SIP requests. The user can configure which call-control property should take place as the user-part, the host-part or as additional parameter in a SIP header's URI.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)[if-name]#[no] address-translation outgoing-call** *{header} {parameter} {source} {property}*	Specifies an outgoing address translation rule.

Mapping SIP headers to call-control properties

This functionality specifies rules that describe which SIP header should take place as a specific call-control property like the called-e164 or the calling-e164 number.

Mode: Interface SIP

Step	Command	Purpose
1	**[node](if-sip)[if-name]#[no] address-translation incoming-call {property} {header}**	Specifies rule to extract call-control *property* from a *header* in the incoming call.
2	**[node](if-sip)[if-name]#[no] address-translation incoming-call called-e164** { fix | request-uri | to-header | p-called-party-id **}**	Specifies rule to extract *called-e164* number from one of the headers: fix, request-uri, to-header or p-called-party-id (Default header: request-uri) in the incoming call.
3	**[node](if-sip)[if-name]#[no] address-translation incoming-call called-uri { fix | local | local-ip | request-uri | to-header | p-called-party-id }**	Specifies rule to extract *called-uri* from one of the headers: fix, local, local-ip, request-uri, to-header or p-called-party-id (Default header: request-uri) in the incoming call.
4	**[node](if-sip)[if-name]#[no] address-translation incoming-call called-name { fix | to-header | p-called-party-id }**	Specifies rule to extract *called-name* property from one of the headers: fix, to-header or p-called-party-id (Default header: to-header) in the incoming call.

Configuring ISDN redirecting number tunneling over SIP

A SIP interface can be configured to tunnel the ISDN Redirecting E.164 Number and Redirecting Reason. This is implemented per the IETF draft standard method according to draft-jennings-sip-voicemail-uri-05.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)[if-name]# address-translation outgoing-call**	Enables Redirecting Party Number Tunneling ISDN ‡ SIP: Enables transmission of the target and cause parameters in the Request-URI for outgoing SIP calls. Whenever the (incoming ISDN) call has a redirecting party number information element, the Request-URI is extended by the target and cause parameters. Default: disabled
2	**[*node*](if-sip)[if-name]# address-translation incoming-call**	Enables Redirecting Party Number Tunneling SIP ‡ ISDN: Enables reception of the target and cause parameters in the Request-URI for incoming SIP calls. Sets the redirecting party number information element for (outgoing ISDN) calls where the target parameter is present in the Request-URI. Default: disabled

Enabling SIP RFC privacy, asserted-identity, & preferred-identity headers (RFC 3323/3325)

The following command sequence enables support for the SIP Privacy and Asserted-Identity headers, sending and receiving of the Privacy as well as the Asserted-Identity headers of RFCs 3323 & 3325. This provides the required identity to SIP entities. The privacy header suppresses the forwarding of the identity to the final station. Handling of the Asserted-Identity header can be configured in the same way as for any other SIP header using the address-translation command in the SIP interface configuration mode. The privacy header is mapped to the call-control's presentation indicator property field.

The type of header sent depends on the screening-indicator property of the call-control. When receiving one of these privacy headers the screening-indicator is also set depending on the received header type.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)[if-name]#[no] privacy**	Enables/Disables privacy. Default: disabled

Updating caller address parameters

A SIP User Agent Client (UAC) can use the sip update method for modifying caller-name and caller-number. These parameters must be included in the P-Asserted-Identity header of the sip update message. This feature is often required if sip has to interwork with an analog telephone network where the Caller-Id cannot be delivered together with the call offering.

It is possible to configure the sip interface to wait for the caller-name, caller-number or both parameters before routing the call. If the caller-address parameters are sent with sip update and the call will be forwarded to a network where these parameters must be present at call setup time (ISDN), this wait-service must be enabled.

Note The SIP interface of the call-router must be configured to take the caller addresses parameters for incoming calls from the P-Asserted-Identity header. See "Enabling SIP RFC privacy, asserted-identity, & preferred-identity headers (RFC 3323/3325)" on page 546 for more information.

Mode: Interface SIP

Step	Command	Purpose
1	[*node*]**(if-sip)**[*interface*]**#[no] update accept address {wait-for-name | wait-for-number | wait-for-name-and-number} [[proceeding-timeout** <*timeout*>**][early-alerting timeout** <*timeout*>**]]**	**Enables or disables wait service:** **Wait Options:** • **wait-for-name** The call will be routed as soon as a valid caller name is present. • **wait-for-number** The call will be routed as soon as a valid caller number is present. • **wait-for-name-and-number** The call will be routed as soon as a valid caller name and a valid caller number are present. **Wait Timeouts:** • **proceeding-timeout** After this time, the call will be routed and the specified caller parameters are not present. *Default:* 4000ms • **early-alerting-timeout** If the specified caller parameters are not present after this time, a 180 RINGING will be sent to UAC. *Default:* 0ms (immediately)

SIP diversion header

Trinity supports the SIP Diversion Header for transmitting redirecting information over SIP according to draft-levy-sip-diversion-08. Sending and receiving of the header can be configured independent of each other. Even though the Diversion Header standard would allow appending a header for each diversion occurred in the network, Trinity only records the last and the first diversion. If only one Diversion Header is attached to the INVITE request, then it represents the last diversion.

Transmit Direction. For enabling sending of the Diversion Header, an outgoing address translation expression must be configured on the sip interface. This expression specifies how to create the Diversion URI of the header. As the User Part of the URI, the Calling Redirecting number will always be taken. The user must configure the Host Part that is set per default to none. Setting the Host Part to none disables transmission of the Diversion Header.

Mode: Interface SIP

Step	Command	Purpose
1	**[name] (if-sip)**[*interface*]**#address-translation outgoing-call diversion-header host-part** {call | local | remote | fix | interface | none}	Enables or disables sending of the Diversion Header and specifies the Host Part of the URI.

Receive Direction. For receiving of the Diversion Header, an incoming address translation expression must be configured on the sip interface. Because several methods for transmitting redirecting information are available, this expression specifies that they must be taken from the Diversion Header when providing them to the call control.

Mode: Interface SIP

Step	Command	Purpose
1	**[name] (if-sip)[if-name]#address-translation incoming-call calling-redir diversion-header**	Enables or disables extracting of the redirection information from the Diversion Header.

SIP REFER Transmission (& ISDN Explicit Call Transfer support)

Additional call transfer support, a push-back mechanism, is enabled by default for SIP interfaces by sending REFER messages.

Trinity detects calls that are looped internally, i.e. calls that leave the device over the same SIP interface over which they enter the device. If an internal loop is detected for a SIP interface, Trinity sends a REFER message to push back the call to the connected network as soon as the call is connected.

ISDN interfaces react similarly to internally looped calls. The push-back mechanism is enabled by default for ISDN interfaces (BRI ports) by accepting or rejecting explicit call-transfer (ECT) invocations. An ISDN phone that is connected to a BRI port and that has two active calls can send an ECT invocation to connect the two calls inside the device. An ISDN interface can be configured to accept or reject ECT invocations.

Trinity detects calls that are looped internally, i.e. calls that leave the device over the same ISDN interface over which they enter the device. If an internal loop is detected for an ISDN interface bound by an ISDN user port, Trinity sends an explicit call-transfer (ECT) to push back the call to the connected network as soon as the call is connected. An ISDN interface can be configured to emit ECT invocations.

Figure 101 on page 638 shows an example scenario where a SIP network connects two devices to give a home office access to a PBX in the central office.

The phone in the home office has two active calls to other subscribers of the PBX in the central office. The user wants to connect the other two participants and (a) sends an explicit call-transfer invocation to the device HO. The device HO internally connects the two calls and sends a DISCONNECT message to the phone for both calls. In a second step (b) the firmware on HO detects an internal loop. Both call legs are connected to the same network. In this example, both call legs are handled by the same SIP gateway. The firmware on device HO sends a REFER message to device CO, which connects the two call legs internally and sends a BYE message to the device HO. (c) Again the firmware of CO detect an internal loop. This time the call legs are handled by the same SIP interface, connected to the PBX. Since the ISDN port is a user port it sends an explicit call-transfer invocation to the PBX (d), which connects the call and sends the device CO a DISCONNECT message for both calls. During all these push back operations the data path of the two participants keeps connected.

The push back mechanism over ISDN (using ECT) and SIP (using REFER) works independently of the protocol that invoked the call-transfer.

The push-back mechanism can be configured on each interface separately. Per default push-back is enabled for ISDN and SIP interfaces. You only have to change the configuration if you don't want internally looped calls to be pushed back to the network. The configuration command **[no] call-transfer accept** configures if an incoming call-transfer request (e.g. ECT or REFER) shall be accepted. The configuration command **[no] call-**

transfer emit configures if a call reaching the device over this interface and leaving the device over this interface shall be pushed back to the network, i.e. if a call-transfer request (ECT or REFER) shall be sent.

The following procedure disables the push-back mechanism on the ISDN interface connected to the PBX. No ECT invocation is sent when a call is detected that is looped internally.

Mode: Interface ISDN

Step	Command	Purpose
1	**[*node*](if-isdn)[if-name]#no call-transfer emit**	Disables the ISDN push back mechanism.

The following procedure disables the push-back mechanism on a SIP interface. No REFER message is sent when a call is detected that is looped internally.

Mode: Interface SIP

Step	Command	Purpose
1	**device(if-sip)[if-name]#no call-transfer emit**	Disables the SIP push back mechanism.
2	**device(if-sip)[SWITCH.IF_SIP0]#call-transfer expires <seconds>**	Defines the expires timeout for a call transfer using SIP refer. If a call transfer will last longer than the expires time, the call(s) will be closed by the Patton Device.

AOC Over SIP (Optional)

This enhancement sends AOC (Advice of Charge) information transparently from ISDN to SIP and vice versa. AOC-D elements are hex-encoded and sent as application/QSIG content in SIP INFO messages during a session.

Mode: Interface SIP

Step	Command	Purpose
1	**[name] (if-sip)[*interface*]#[no] aoc-d accept**	Enables or disables the reception of SIP Info messages containing AOC-D elements and propagate charging information to adjacent peer.
2	**[name] (if-sip)[*interface*]#[no] aoc-d emit**	Enables or disables the sending of SIP Info messages with AOC-D elements containing charging information from adjacent peer. If no charging information is available, no message is sent.

The following commands have to be used to receive and send Advice Of Charge in ASN1 or XML format. Please note that ASN1 format is only supported during a call.

Mode: Interface SIP

Step	Command	Purpose
1	**[node](if-sip)[IF_SIP]#[no] aoc-s** { accept | emit }	Enable/disable the SIP interface AOC /emit feature at the beginning of a call. Default: disabled
2	**[node](if-sip)[IF_SIP]#[no] aoc-d** { accept | emit }	Enable/disable the SIP interface AOC accpt/emit feature during a call. Default: disabled
3	**[node](if-sip)[IF_SIP]#[no] aoc-e** { accept | emit }	Enable/disable the SIP interface AOC accpt/emit feature at the end of a call. Default: disabled
4	**[node](if-sip)[IF_SIP]#[no] aoc-forma**t { asn1 | xml }	Sets the SIP interface AOC format. Default: asn1

Enabling the Session Timer (Optional)

The gateway implements the SIP session timer feature, which is currently only defined in SIP draft standards. The session timer feature allows a gateway to check periodically during a call, if the remote gateway is still alive and if the call is still connected on the remote gateway. You can enable this feature using the command shown below. If the session timer feature detects that the call is no longer connected on the remote side, it will also drop the call locally.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)[if-name]#[no] session-timer <seconds>**	Enables/Disables the session timer with a refresh period of the specified number of seconds. Default: disabled

Enabling the SIP Penalty-box Feature (Optional)

Normally a SIP call is started by sending an INVITE request to the destination. If that destination is unavailable for any kind of reason, it could take some time to detect that this destination is failing. During that time the party who initiated the call is waiting for the ring-back tone as feedback. To try an alternative path to reach the called destination it is practically too late because the caller gave up during that time. It is now possible to determine the reachability of destinations ahead of actual calls. For a failing destination an alternative routing path to the called end user could be issued immediately without having a timeout.
This is done by periodically sending SIP OPTION requests to the configured remote address. If we receive an answer for that OPTIONS request the destination is still alive and calls are forwarded toward it. For the reachability the content of the answer does not matter, neither if the answer is a failing or successful one. If there is no answer during the configured timeout, the remote destination is added to the penalty-box. For a certain time such destinations are considered not reachable, no requests are sent to them and calls can be redirected immediately without timeout. This could happen for example with a hunt-group service in the call-control.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)[if-name]#[no] penalty-box**	Enables/Disables the SIP penalty box feature. Default: disabled

Initiating a New SIP Session for Redirected SIP Calls (Optional)

Normally, if a SIP call is redirected to a different location by receiving a SIP 3xx response, only the request header is replaced in the original INVITE message and the INVITE is sent to the new destination. Using the following procedure, the SIP gateway can be forced to create a completely new session for forwarded calls.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)[if-name]#[no] new-session-after-redirect**	Enables/disables creation of a new session after a redirect response. Default: disabled

Rerouting Calls from SIP (Optional)

The **call-reroute** command allows to redirect calls to other mediums like ISDN through the call-control.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)[if-name]#[no] call-reroute accept**	Enables or disables the rerouting to other networks.

Configuring the SIP Hold Method (Optional)

There are different ways to set a remote SIP subscriber On Hold. This command specifies which method the device uses to indicate this call state. In receive direction, all of them will be accepted.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)[if-name]#hold-method {direction-attribute | zero-ip}**	Configures the method to be used to signal the hold state. Default: zero-ip
2	**[*node*](if-sip)[if-name]#hold-method direction-attribute { sendonly | inactive }**	Configures the *direction-attribute* hold method. Default: sendonly

Configuring SIP Overlap Dialing (Optional)

When the sending feature of overlap-dialing is enabled, the user part of the to-header and request URI is taken from the called party number. Trinity sends for each update of the called party number a new INVITE message with an updated to-header and request URI, but uses the same call-id and from-tag as the original invite. As soon as one of the INVITE requests gets a response establishing an early-dialog the overlap dialing is finished. All other INVITE requests which are still pending are canceled at this time and additional digits are transported through the established session. If all INVITE requests get a negative answer and the sip interface receives no updates of the called party number for the digit-collection timeout time, the overlap dialing fails and the call is dropped.

When overlap dialing acceptance is enabled, an incoming INVITE request is answered with a “100 Trying” message and the call is forwarded to the called destination. When an INVITE is received with the same call-id and from-tag as an already pending INVITE request, the pending request is always released with a “484 address incomplete” message. Then, the new request is answered with a “100 Trying” message and the additional digits received with the new INVITE are forwarded to the called destination. When the called destination signals to establish an early-dialog the overlap acceptance period ends and the last received INVITE request is answered with the according answer. When the called destination drops the call, the overlap dialing session ends with a final negative response. An additional INVITE, coming in after overlap dialing has ended, is treated as an independent request and creates a new call.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)[name]# [no] overlap-dialing new-transaction accept**	Configures the sip interface to accept overlap dialing.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)[name]# [no] overlap-dialing new-transaction emit**	Configures the sip interface to emit overlap dialing.

Configuring PRACK for Reliable Provisional Responses (optional)

Reliable Provisional Responses are supported according to RFC3262. Reliability is achieved by issuing a PRACK message upon reception of a provisional response.

Note the following conditions:

- It is the UAC that issues an INVITE message, which will also issue a PRACK message upon reception of a reliable provisional response.
- It is the UAS that receives an INVITE request, which will also receive a PRACK request upon issuing a reliable provisional response.

Two configuration commands in the SIP interface of 'context cs' allow specifying the behavior on both client and server sides. The UAC indicates its behavior with a 100rel tag in the Supported or Required header field of the INVITE message.

UAC Side:

Step	Command	Purpose
1	**[node](if-sip)[name]# [no] prack emit {supported | required}**	Enables PRACK for outgoing calls when acting as UAC Default: disabled

- **supported:** The peer may send provisional responses reliably.
- **required:** The peer must send provisional responses reliably. The call fails if the peer is not willing or fails to do so.

UAS Side:

Step	Command	Purpose
1	**[node](if-sip)[name]# [no] prack accept {supported | preferred | required}**	Enables PRACK for incoming calls when acting as UAS Default: disabled

- **supported:** Provisional responses are sent reliably if it is required by the peer.
- **preferred:** Provisional responses are sent reliably if it is supported or required by the peer.
- **required:** Provisional responses are sent reliably. The peer must support reliable provisional responses. The call fails if the peer does not support reliability for provisional responses.

The following table summarizes the UAS' behavior:

		Remote Peer		
		no reliability	supported	required
Local configuration	no prack accept	non-reliable	non-reliable	failure
	prack accept supported	non-reliable	non-reliable	reliable
	prack accept preferred	non-reliable	reliable	reliable
	prack accept required	failure	reliable	reliable

Limitations:

The following scenarios are not supported:

- Reliable Provisional Responses and PRACK are not supported for any re-INVITE scenarios.
- If an INVITE request is sent to a stateless proxy who forks the request to several UAS, then the SIP protocol on the device cannot correctly handle all Reliable Provisional Responses it receives from all the UAS. Hence, such a scenario will not work.

Configuring History-Info in SIP

For SIP to SIP calls the History-Info header is now supported according to RFC 7044. This includes reception and forwarding, the generation of new headers for changed request uri and generation of new headers for redirected calls. The History-Info header is only treated in INVITE messages and its responses. The support of the History-Info header is announced with the "histinfo" tag in the Supported header of the initial INVITE.

Mode: context cs / interface sip

Step	Command	Purpose
1	**node(if-sip)[<name>]#[no] history-info emit**	Enables or disables the sending of History-Info headers in INVITE messages and responses to INVITE messages. Default is disabled.

Enabling NAT Traversal for SIP INVITE Messages

The following command enables Nortel-type NAT traversal for SIP INVITE messages. This command sends SIP PING requests every 55 seconds after a successful registration to keep NAT open.

Mode: Interface SIP

Step	Command	Purpose
1	**[*node*](if-sip)[<name>]#[no] nat-traversal nortel**	Enables Nortel-type NAT traversal for SIP INVITE messages.

Accepting Re-invite After Reboot

To distinguish the re-invite from an initial invite, Trinity examines if a header-parameter ‘tag’ is present in the To header. If the To-tag is present and the re-invite does not match any ongoing call, it is rejected with a ‘481 Call/Transaction Does Not Exist’ answer. This behavior can be changed to accept an initial invite with a To-tag, which goes against RFC3261 and should therefore be avoided.

Mode: SIP

Step	Command	Purpose
1	**[*node*](if-sip)#[no] check-to-tag**	Specifies if initial invites with To-tag should be rejected. Default: *enabled*

Chapter 49 Secure SIP Applications

Chapter contents

Introduction

With Transport Layer Security (TLS) and Secure Real-Time Protocol (SRTP) Trinity devices are capable of securing both signaling connections and voice streams of SIP calls. Both protocols have to operate in harmony and work in a couple of different scenarios. Trinity, the software running on Patton devices offers a vast number of configuration options, depending on whether you just want to use TLS/SRTP to encrypt your calls or whether you would like to set up a proper trust relationship with certificates to be able to authenticate the SIP peer.

This chapter discusses different scenarios and provides the configuration snippets to set up the Patton device for them. In particular, this chapter covers the following scenarios:

- TLS/SRTP encryption without TLS authentication
- TLS/SRTP encryption with mutual TLS authentication (MTLS) based on mutually exchanged, locally-generated, self-signed certificates
- TLS/SRTP encryption with mutual TLS authentication (MTLS) in a Microsoft Lync environment.

TLS/SRTP encryption without TLS authentication

TLS connections always perform two tasks: symmetric packet encryption and endpoint authentication through certificates. You cannot have one without the other. Setting up a trust relationship with certificates is a complex task, described in the next scenarios. This scenario covers the case where you just want to encrypt SIP signaling traffic (and secure the RTP streams) without caring about TLS endpoint authentication. In this case it is sufficient to use a self-signed certificate. In a web-of-trust certificate scheme there is no central CA, and so identity certificates for each user can be self-signed. As the name tells it is an identity certificate that is signed by the same entity whose identity it certifies.

Patton devices come with an automatically and randomly generated private key and with a self signed certificate (with a modulus length of 512 bits). If you don't change the default TLS profile, this self-signed certificate will be used automatically for all TLS connections. Note that in this scenario, peer SIP devices will not be able to authenticate the Patton device over TLS, because the self-signed certificate is only stored locally and no trust relationship between the Patton device and the peer device exists. However, TLS still encrypts the signaling traffic, which may be sufficient in some basic scenarios.

Configuration without TLS/SRTP

In the following, you will learn how you can secure the SIP and RTP communication starting from an existing configuration that does not use TLS or SRTP yet. Figure 1 below depicts a LAN segment where two Patton SIP-to-SIP gateways are connected back-to-back. They are to secure a SIP connection between two SIP soft-phones that are not capable of doing TLS/SRTP.

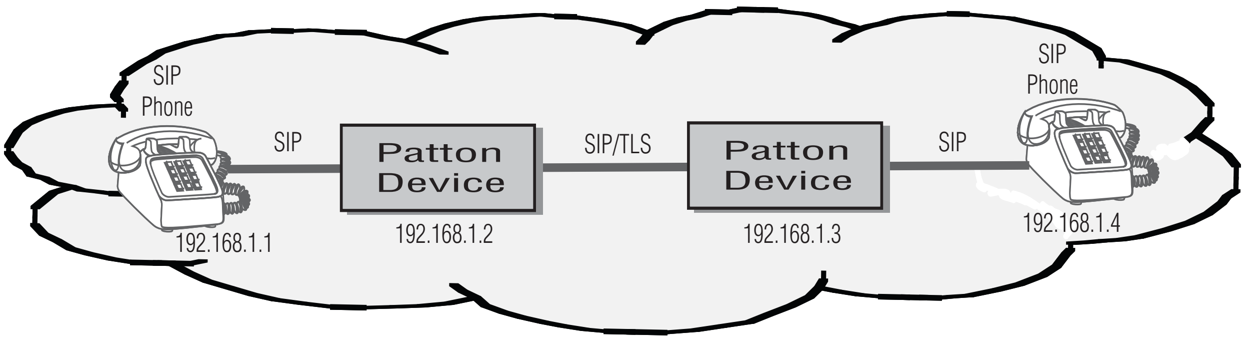


Figure 90. Patton device secures a SIP connection between two SIP soft-phones.

The devices encapsulate the plain SIP messages into a TLS connection and at the same time secure the voice streams using SRTP.

Let us assume that the two gateways are already configured to communicate via a non-secure SIP and RTP connection. Please find the configuration of the Patton device 192.168.1.2 before enabling TLS/SRTP below. Note that on first boot up the Patton device automatically created the DEFAULT private key, the DEFAULT own-certificate, and the DEFAULT TLS profile. Refer to the section “Generated default files” in Chapter 30, "Public-Key Infrastructure (PKI)" on page 301 for more information.

```
profile tls DEFAULT
   no authentication incoming
   no authentication outgoing
   private-key pki:private-key/DEFAULT
   own-certificate 1 pki:own-certificate/DEFAULT

profile voip DEFAULT
   codec 1 g711alaw64k rx-length 20 tx-length 20
   codec 2 g711ulaw64k rx-length 20 tx-length 20

context ip ROUTER
   interface LAN
      ipaddress LAN 192.168.1.2/24

context cs SWITCH
   no shutdown

interface sip IF_SIP_PHONE
   bind context sip-gateway GW_SIP
   route call dest-interface IF_SIP_TLS
   remote 192.168.1.1
   no call-transfer accept
   no call-transfer emit

interface sip IF_SIP_TLS
   bind context sip-gateway GW_SIP
   route call dest-interface IF_SIP_PHONE
   remote 192.168.1.3
```

```
      no call-transfer accept
      no call-transfer emit

   context sip-gateway GW_SIP
      use profile tls DEFAULT

      interface GW_SIP_IF
      transport-protocol udp+tcp 5060
      no transport-protocol tls
      bind ipaddress ROUTER LAN LAN

   context sip-gateway GW_SIP
      no shutdown

   port ethernet 0 0
      bind interface ROUTER LAN
      no shutdown
```

Tasks to Configure TLS/SRTP

The following steps list all commands you will have to enter to configure the Patton device 192.168.1.2 in order to secure the connection to the peer device 192.168.1.3 by using TLS for signaling and SRTP for voice-data traffic.

Configure URI-scheme

First, use the sips URI-scheme towards the peer Patton device. This makes sure that outbound requests are sent with a sips URI, which forces the gateway to use a secure connection such as TLS:

```
node>enable
node#configure
node(cfg)#context cs SWITCH
node(ctx-cs)[SWITCH]#interface sip IF_SIP_TLS
node(if-sip)[SWITCH.IF_SIP_TLS]#uri-scheme sips
```

Make sure that the URI scheme towards the connected SIP soft-phone is still set to sip, because we don't want to secure the connection to the soft-phone:

```
node(cfg)#context cs SWITCH
node(ctx-cs)[SWITCH]#interface sip IF_SIP_PHONE
node(if-sip)[SWITCH.IF_SIP_TLS]#uri-scheme sip
```

Enable TLS on the SIP gateway

Next, configure the transport protocol TLS on the interface of the SIP gateway.

```
node(cfg)#context sip-gateway GW_SIP
node(sip-gw)[GW_SIP]#interface GW_SIP_IF
node(sip-if)[GW_SIP.GW_SIP_IF]#transport-protocol tls
```

Enable SRTP on the VoIP profile

Finally, secure the stream of voice-data packets by configuring SRTP in the VoIP profile.

```
node(cfg)#profile voip DEFAULT
node(pf-voip)[DEFAULT]#rtp security preferred
```

The setting preferred means that we only use SRTP if the peer device supports it as well. This allows us to use the same VoIP profile for the *IF_SIP_TLS* as well as for the *IF_SIP_PHONE* interface, assuming that the soft phone just ignores the crypto attribute in the SDP part of the SIP message. If your soft-phone does not support the crypto attribute at all, you will have to create a separate VoIP profile and use in on the interface *IF_SIP_PHONE.*

Note You have to perform the same tasks on the other Patton device 192.168.1.3.

Configuration with TLS/SRTP

The text below includes all TLS/SRTP configuration tasks (highlighted in bold).

```
profile tls DEFAULT
   no authentication incoming
   no authentication outgoing
   private-key pki:private-key/DEFAULT
   own-certificate 1 pki:own-certificate/DEFAULT

profile voip DEFAULT
   codec 1 g711alaw64k rx-length 20 tx-length 20
   codec 2 g711ulaw64k rx-length 20 tx-length 20
   rtp security preferred

context ip ROUTER
   interface LAN
   ipaddress LAN 192.168.1.2/24

context cs SWITCH
   no shutdown

   interface sip IF_SIP_PHONE
      bind context sip-gateway GW_SIP
      route call dest-interface IF_SIP_TLS
      remote 192.168.1.1
      no call-transfer accept
      no call-transfer emit
      uri-scheme sip

   interface sip IF_SIP_TLS
      bind context sip-gateway GW_SIP
      route call dest-interface IF_SIP_PHONE
      remote 192.168.1.3
      no call-transfer accept
      no call-transfer emit
      uri-scheme sips

context sip-gateway GW_SIP
   use profile tls DEFAULT

   interface GW_SIP_IF
      transport-protocol udp+tcp 5060
      transport-protocol tls 5061
      bind ipaddress ROUTER LAN LAN


context sip-gateway GW_SIP
```

```
      no shutdown

   port ethernet 0 0
      bind interface ROUTER LAN
      no shutdown
```

TLS/SRTP encryption with mutual TLS authentication (MTLS) based on exchanged, locally-generated, self-signed certificates

This scenario extends the previous one by mutual TLS authentication. Instead of using certificates that have been signed by a proper Certification Authority (CA), we still use self-signed certificates here. For this purpose, we let the two Patton devices exchange their self-signed, locally-generated certificates via a TFTP server (see Figure 2). This is helpful if you want to avoid the complexity of a proper certification chain but still want to make sure that the two Patton devices authenticate themselves over TLS

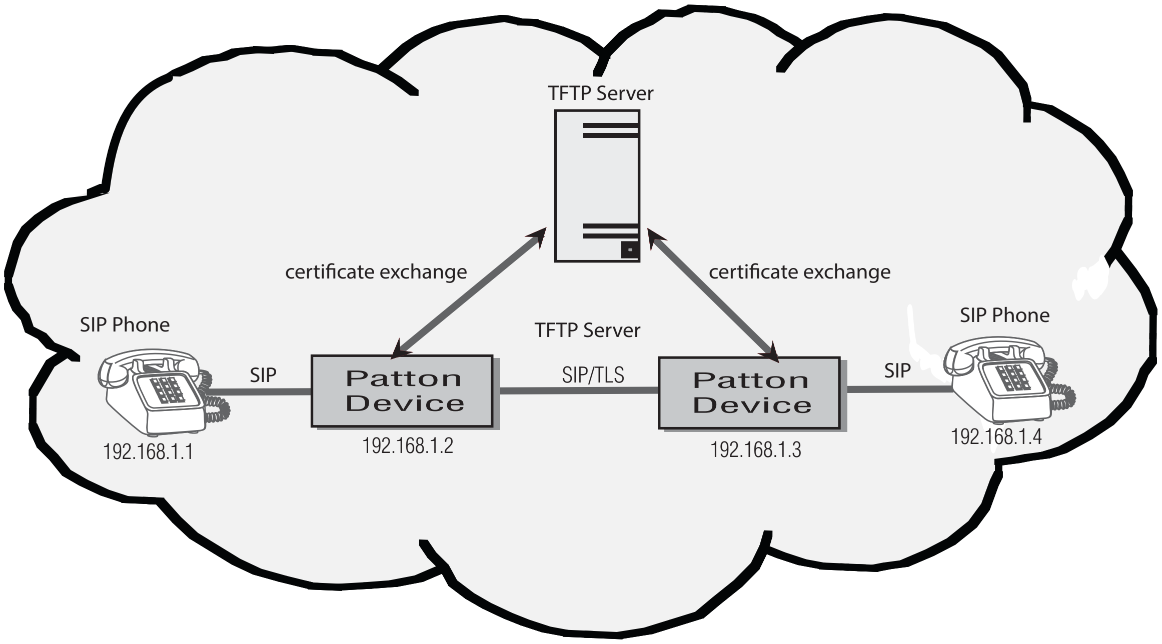


Figure 91. Patton devices secures a SIP connection between two SIP soft-phones.

The Patton devices encapsulate the plain SIP messages into a TLS connection and at the same time secure the voice streams using SRTP. The locally-generated, self-signed certificates are exchanged via a TFTP server prior to making calls.

Tasks to create and exchange self-signed certificates

The following steps start with the TLS configuration completed in the previous section.

Generate private key

We continue to configure the Patton device 192.168.1.1 and first create a new private key with longer modules:

```
node(cfg)#generate pki:private-key/SECRET1 key-length 2048
Generating RSA private key, 2048 bit long modulus
..+++
.................+++
e is 65537 (0x10001)
writing RSA key
```

Generate certificate request

As a next step, generate a certificate request derived from the created private key. The certificate request contains the same information as in the final certificate, but the request is not signed yet.

Note Note: The common-name at the end of the command must match the IP address of the Patton device.

```
node(cfg)#generate pki:certificate-request/REQUEST1 private-key
pki:private-key/SECRET1 country CH state Bern locality Bern organization
Patton organization-unit RND common-name 192.168.1.2
```

Self-sign the certificate request

You can now sign the certificate request locally with the same private key that we used to create it (self-signed). We set the validity period to one year (356 days). Note: the printed CN field must match the IP address of the Patton device:

```
node(cfg)#generate pki:own-certificate/CERT1 pki:certificaterequest/
REQUEST1 private-key pki:private-key/SECRET1 validity-period 356
Signature ok
subject=/C=CH/ST=Bern/L=Bern/O=Patton/OU=RND/CN=192.168.1.2
Getting Private key
```

Exchange certificates

The next step is to deploy the certificate to the peer Patton device. Export the created owncertificate to your TFTP server (e.g. at 192.168.1.10) and import it on the peer device as trusted certificate:

```
node(cfg)#copy pki:own-certificate/CERT1 tftp://192.168.1.10/
CERT1_FROM_192.168.1.2
```

If all the steps above were executed for the device 192.168.1.3 as well, you now are able to import the remote certificate (stored as CERT1_FROM_192.168.1.3) from the TFTP server as trusted certificate:

```
node(cfg)#copy tftp://192.168.1.10/CERT1_FROM_192.168.1.3 pki:trustedcertificate/
CERT1_FROM_192.168.1.3
```

Configure the TLS profile

Create a new TLS profile and configure it with the new private key and the own-certificate of the device:

```
node(cfg)#profile tls TLS_AUTH
node(pf-tls)[TLS_AUTH]#private-key pki:private-key/SECRET1
node(pf-tls)[TLS_AUTH]#own-certificate 1 pki:own-certificate/CERT1
```

Configure the imported trusted certificate in the TLS profile.

```
node(pf-tls)[TLS_AUTH]#trusted-certificate pki:trustedcertificate/
CERT1_FROM_192.168.1.3
```

Enable mutual TLS authentication:

```
node(pf-tls)[TLS_AUTH]#authentication incoming
node(pf-tls)[TLS_AUTH]#authentication outgoing
```

Finally, use the new TLS profile on the sip-gateway.

```
node(cfg)#context sip-gateway GW_SIP
node(sip-gw)[GW_SIP]#use profile tls TLS_AUTH
```

Configuration with TLS/SRTP and mutual TLS authentication

The text below includes all configuration task discussed in this section (highlighted).

```
profile tls DEFAULT
   no authentication incoming
   no authentication outgoing
   private-key pki:private-key/DEFAULT
   own-certificate 1 pki:own-certificate/DEFAULT

profile tls TLS_AUTH
   authentication incoming
   authentication outgoing
   private-key pki:private-key/SECRET1
   own-certificate 1 pki:own-certificate/CERT1
   trusted-certificate pki:trusted-certificate/CERT1_FROM_192.168.1.3

profile voip DEFAULT
   codec 1 g711alaw64k rx-length 20 tx-length 20
   codec 2 g711ulaw64k rx-length 20 tx-length 20
   rtp security preferred

context ip ROUTER
   interface LAN
     ipaddress LAN 192.168.1.2/24

context cs SWITCH
   no shutdown

   interface sip IF_SIP_PHONE
     bind context sip-gateway GW_SIP
     route call dest-interface IF_SIP_TLS
     remote 192.168.1.1
     no call-transfer accept
     no call-transfer emit
     uri-scheme sip

   interface sip IF_SIP_TLS
     bind context sip-gateway GW_SIP
     route call dest-interface IF_SIP_PHONE
     remote 192.168.1.3
     no call-transfer accept
     no call-transfer emit
     uri-scheme sips

context sip-gateway GW_SIP
```

```
      use profile tls TLS_AUTH

      interface GW_SIP_IF
        transport-protocol udp+tcp 5060
        transport-protocol tls 5061
        bind ipaddress ROUTER LAN WAN

   context sip-gateway GW_SIP
      no shutdown


   port ethernet 0 0
       bind interface ROUTER WAN
       no shutdown
```

TLS/SRTP encryption with mutual TLS authentication (MTLS) in a Microsoft Lync environment

This scenario makes use of proper certificates, signed by an official Certification Authority (CA). The goal is to connect a Patton device as an Enterprise Session Boarder Router to a Microsoft Lync network. For this purpose, please read Chapter 49, "Secure SIP Applications" on page 555. It contains a detailed description of how to setup both the Lync server as well as the Patton device, including certificate handling.

Chapter 50 SIP Security

Chapter contents

Introduction

This chapter provides overviews and describes the tasks involved in configuring the following;

- Trinity's Transport Layer Security (TLS) capabilities for SIP (see section "Transport Layer Security (TLS)").
- Trinity's Secure Real-Time Prtotocol (SRTP) capabilities (see section "Secure Real-Time Protocol (SRTP) Configuration" on page 590).

Transport Layer Security (TLS)

TLS provides communication security over insecure networks such as the Internet. TLS allows transport connections such as TCP to be safe from eavesdropping and tampering. For this purpose, TLS uses a Public Key Infrastructure (PKI) for authentication and confidentiality of the key exchange. Payload confidentiality is ensured with symmetric encryption using the exchanged keys. Message authentication and integrity is achieved by adding authentication codes to each message.

Using TLS for SIP signaling ensures that the content of the SIP messages is not subject to eavesdropping. This is especially important for SRTP master keys, which are exchanged over SIP/SDP signaling messages. TLS guarantees that the next hop destination is authenticated and that the messages are not altered by a third party.

TLS configuration task list

To configure TLS, perform the tasks in the following sections:

- Install license
- Configure URI scheme
- Configure Time
- Configure Public-Key Infrastructure (PKI)
- About TLS profiles
- Configure TLS profile
- Configure TLS-profile usage
- Configure TLS transport
- Configure transport-protocol enforcement or fallback
- Configure non-default TLS port usage
- Configure SRTP
- About security issue: connection re-use
- Troubleshooting

Install license

Voice security is a licensed option for Patton devices. You need to have a "sip-tls-srtp" license installed on your device before you are able to configure and use SRTP and TLS.

Configure URI scheme

The next step for enabling SIP over TLS is to configure the URI scheme for all SIP services that require a secure transmission. The URI scheme for SIP calls is to be configured on the SIP interface in context CS, whereas the URI scheme for SIP registrations is configured on the outbound registration mode in the context SIP gateway. This separation gives you the flexibility to decide individually whether or not call or registration connections will be encrypted.

URI Scheme for Calls

For calls there are three possible settings for the URI scheme.

- SIP
- SIPS
- transparent

SIP. The SIP URI scheme defines an unsecure SIP URI, i.e., "sip:200@patton.com". Calls destined to a request-URI with the SIP URI scheme usually are signaled over non-secure transports as UDP and TCP. However, they may be signaled over secure transports such as TLS as well. The transports depend on the availability and preference of transport protocols for reaching the given URI. For example, a DNS lookup for an URI according to the SIP URI scheme may return a list of transport protocols, including TLS. Our protocol preference selection configurable in the call-outbound configuration mode of the location service – then defines which of the transports is preferred.

SIPS. The SIPS URI scheme defines a secure SIP URI, i.e., "sips:200@patton.com". Calls destined to a request-URI with the SIPS URI scheme must be signaled over the secure transports TLS. (Although there are alternative secure transports protocols, Patton devices only support TLS at the moment).

Transparent. Selects the URI scheme on a per-call basis. The actual URI scheme for an outgoing call is defined by the call-control peer, i.e., over the protocol where the call entered the Patton device. For all protocols other than SIP, when there isn't any security information – we always use the SIP URI scheme. However, when an incoming SIP call is routed over SIP again, the URI scheme is tunneled transparently through our call-control application. If the requested URI of the incoming call is SIPS and the transport protocol is TLS, then the outgoing call will also be placed using the SIPS URI scheme. However, when the incoming request-URI is SIP or when it arrives over a non-secure transport the resulting URI scheme is SIP.

Mode: Interface SIP

Step	Command	Purpose
1	**nodeif-sip)[name]# uri-scheme { sip | sips | transparent }**	Configures the URI scheme for outgoing calls. Default is transparent.

Note If you are using SRTP it is strongly recommended to signal calls over a secure TLS connection by using the SIPs URI scheme. This is because the SRTP session's master key is exchanged in the SDP portion of SIP signaling messages. If an insecure transport is used for SIP signaling, an eavesdropper may read the SRTP master key and listen to the exchanged RTP voice packets. See "Configure SRTP" below for more information.

URI Scheme for Registration

The URI scheme for outbound registrations can be configured separately from the URI scheme for outbound calls.

Note For registration, the transparent option is not available, because there isn't any incoming source for registrations.

- SIP
- SIPS

SIP. The SIP URI scheme defines an insecure SIP URI, for example "sip:200@patton.com". The registration messages and incoming calls to the registered contact *may* travel over any secure or non-secure transport.

SIPS. The SIPS URI scheme defines a secure SIP URI, for example "sips:200@patton.com". The registration messages and incoming calls to the registered contact *must* use the secure transport protocol TLS.

Mode: Registration outbound

Step	Command	Purpose
1	**node(regout)# uri-scheme { sip | sips }**	Configures the URI scheme for outbound registrations and the corresponding calls toward the registered contact. Default is sip.

URI Scheme for Re-routed Calls

It is possible to specify the behavior regarding URI scheme for rerouted calls. This is especially valuable when the call is rerouted to a URI with a different URI scheme than the original URI of the call.

- Rerouted
- Original

Rerouted. The URI scheme is taken from the reroute operation. This means it could be degradation or an enhancement of security of the call. Calls with a SIP URI could be degraded to a SIP URI or calls with a SIP URI could be changed to a SIP URI. This is the default behavior.

Original. The URI scheme for the rerouted call remains the same as for the original call. This means the rerouting operation cannot have an impact on the security of the call. Neither degradation or an enhancement. There is the possibility that the target destination is not compatible to the original URI scheme and would need the URI scheme from the reroute operation.

Mode: Interface SIP

Step	Command	Purpose
1	**node(if-sip)[name]# call-reroute uri-scheme { rerouted |original }**	Configures the URI scheme for outgoing calls which are rerouted. Default is rerouted.

URI Scheme for Transferred Calls

For calls it is possible to specify the behavior regarding URI scheme in the case of a transfer of a call. This is especially valuable when the call is transferred to a URI with a different URI scheme than the original URI of the call.

- Referred
- Original

Referred The URI scheme is taken from the REFER request. This means it could be a degradation or an enhancement of security of the call. Calls with SIPs URI could be degraded to SIP URI or calls with SIP URI could be changed to SIPs URI. This is the default behavior.

Original The URI scheme for the transferred call remains the same as for the original call. This means the transfer operation cannot have an impact on the security of the call. Neither degradation or an enhancement. There is the possibility that the target destination is not compatible to the original URI scheme and would need the URI scheme from the transfer operation.

Mode: Interface SIP

Step	Command	Purpose
1	**node(if-sip)[name]# call-transfer uri-scheme { referred |original }**	Configures the URI scheme for calls which are transferred. Default is referred.

Configure Time

The system time must be configured before you can work with certificates, and enabling the PKI and TLS. It is also very important to set the correct offset of the local time with respect to the UTC time, due to the validity of certificates expressed in UTC time. The configuration option that works best is to update the UTC time from a configured SNTP server and to derive the local time by configuring the static offset of the local time zone with respect to UTC. If a SNTP server is not available, you should first configure the offset of the local time zone and afterwards set the local time with the **et clock** command.

Note It is important to also correctly configure the time of remote devices, which are having TLS connections to our device.

Configure Public-Key Infrastructure (PKI)

The commands that are needed to set up the public-key infrastructure are described in detail in The PKI is responsible for creating, importing and exporting keys and certificates, and storing them persistently in flash memory.

A TLS profile (described in the next section) groups security and other relevant configuration required to open a TLS connection. For this purpose, the TLS profile may refer to a private, an own certificate and many trusted certificates, In order to configure and use a TLS profile, you need to create and/or import keys and certificates. Refer to "Public-Key Infrastructure (PKI)" on page 301 for more information.

About TLS Profiles

A TLS profile is a collection of TLS configuration parameters, which are used by a SIP gateway to establish a secure transport connection to a remote device. Therefore each SIP gateway is bound to one TLS profile. If your device has to use different PKI keys or certificates to talk to different SIP servers, you have to configure multiple SIP gateway instances, each bound to its own TLS profile.

Basic Operation Principle of TLS

The server side of a TLS connection sends a certificate to the client that serves two purposes: encryption and authentication. First, the certificate contains a public key required to securely negotiate a symmetric session key to encrypt SIP signaling messages. Second, the certificate provides message authentication and integrity by identifying the server. The client is able to authenticate the server by verifying that the server's certificate is derived from one of the locally configured trusted certificates. At the same time, the client is able to verify that the TLS connection is actually terminated by the IP address or domain name listed in the *common name* field of the certificate. The server may request an optional certificate from the client, which is also called mutual authentication.

Encryption usually works with the DEFAULT TLS profile, i.e. without configuring any further options. However, for the authentication and integrity check to work, you have to configure the TLS profile to reflect the trust relationship with the servers/domains you want to communicate with, and you have to manually enable authentication for inbound and outbound connections.

For Trinity, you are able to establish different levels of security; you can set up a quick solution that works with almost zero configuration effort, or you can set up a secure trust relationship in the TLS profile. The following scenarios elaborate more on these security options.

TLS without authentication (default behavior)

This scenario requires no effort to install certificates and only a minimal effort to configure the device, but it also provides the lowest level of security. Each TLS-licensed Patton devices comes with a private key and a default self-signed certificate already installed. The **DEFAULT** TLS profile is configured to use this **DEFAULT** private key and **DEFAULT** own certificate to encrypt SIP signaling messages.

In terms of authentication and integrity check, the security provided by the unaltered TLS profile **DEFAULT** is pretty poor, because authentication for incoming and outgoing TLS connections is switched off. As a consequence, when we accept an incoming TLS connection, Trinity accepts any server certificate without checking a trust relationship and without checking whether the remote IP address matches the c*ommon name* field in the server's certificate. If Trinity acts as a TLS server, the **DEFAULT** TLS profile does not force mutual authentication and thus does not check the identity of the connected client.

Thus, the unaltered TLS profile **DEFAULT** may be helpful for a quick start, but for a serious installation, you should configure a proper trust relationship as described below.

TLS authentication with self-signed certificates

n this scenario we increase the level of security provided by TLS compared to the previous zero-configuration scenario. This is achieved by creating a new self-signed certificate that is only valid for the actual IP address or domain name of the Patton device. See "Public-Key Infrastructure (PKI)" on page 301 to learn how to perform the steps described below:

1. Create a new certificate request, which contains as *common name* field with our public IP address or domain name.
2. Assign a **DEFAULT** private key to the request (or, optionally, assign a newly generated private key).
3. Configure the TLS profile to use the new self-signed certificate and the private key.

Note If such a TLS profile is bound to a SIP gateway, we send this self-signed certificate to identify ourselves. The remote TLS endpoints then are able to verify whether this device's IP address or domain name matches the sent certificate. In order for that to be secure, you also have to export the self-signed certificate and import it to all remote devices as a trusted certificate.

If the device authenticates remote devices, it needs to import the certificates of all remote devices and list them as *trusted certificates* in its TLS profile. In addition, it will switch-on incoming and outgoing authentication to verify whether the remote certificate is derived from one of the trusted certificates listed in the TLS profile. The outgoing authentication process verifies that the remote server endpoint's IP address or domain name matches the common name in the received certificate.

TLS Authentication with 3rd Party Signed Certificates

A generic certification chain consists of the certificate C itself, zero to many certificates of intermediate certification authorities CA1, CA2,.., CAn, and the certificate of the root certification authority CAroot.

In such a case, store certificate C and all certificates of the intermediate certification authorities CA1, CA2,.., CAn on the device that needs to be authenticated. The certificate of the root certification authority CAroot, on the other hand, must be stored as trusted certificate on the device that wants to authenticate the other device(s).

TLS Profile Default

When booting a TLS-licensed Patton device for the first time, a TLS profile with the name DEFAULT is created automatically; it is linked to an automatically-created private key DEFAULT and a self-signed certificate DEFAULT. It is configured not to authenticate and can be used to do authentication-free TLS. For enabling authentication and integrity checks, alter the DEFAULT profile or create a new TLS profile (recommended) and to enable inbound and/or outbound authentication. If a TLS profile is not explicitly bound to a SIP gateway, the DEFAULT TLS profile is used.

Configure TLS Profile

This section describes how to configure TLS profiles, which are those instances in your configuration that collect all TLS-related configuration parameters, and which are used by SIP gateways to establish secure transport connections to remote devices.

Creating a TLS profile and enter configuration mode

This procedure describes how to create a TLS profile and enter the TLS profile configuration mode.

Mode: Administrator exec

Step	Command	Purpose
1	**node(cfg)#profile tls** *name*	Creates the TLS profile *name* and enters its configuration mode.

The parameter *name* is the identifier by which the profile will be known. Entering this command puts you into the TLS profile configuration mode where you can enter the individual TLS configuration parameters.

Use the **no** form of this command to delete a TLS profile. You cannot delete a TLS profile if it is currently bound to a SIP gateway.

To modify a TLS profile that is currently being used by one or more active SIP gateways, the gateways are restarted gracefully; ongoing calls over a certain gateway still use the old TLS configuration until all calls over that gateway have terminated. At that point in time the gateway restarts and all future calls will use the new TLS configuration.

Example: Create a TLS profile

In the following example the TLS profile named *Y_TLS* has been created.

```
node>enable
node#configure
node(cfg)#profile tls MY_TLS
node(pf-tls)[MY_TLS]#
```

Private key

A private key file actually contains a private/public key pair. TLS requires them to secure its key exchange. Thus, it is important to link a TLS profile to a valid private key, which is done by the following command:

Mode: Profile TLS

Step	Command	Purpose
1	**node(pf-tls)[name]# [no] private-key pki:private-key/key**	Configures the private (and public) key to be used for the TLS key-exchange encryption. This configuration is mandatory!

Note If your TLS profile has a configured chain of *own-certificates* it is required that the first certificate in that chain has been created from a certification request, which contains the corresponding public key.

Example: Create a new private key and link it to a new TLS profile.

In the following example, we let Trinity generate a new private key. We then create a new TLS profile, called *Y_TLS* and link it to the generated private key.

```
node>enable
node#generate pki:private-key/MY_PRIVATE_KEY
node#configurenode(cfg)#profile tls MY_TLS
node(pf-tls)[MY_TLS]#private-key pki:private-key/MY_PRIVATE_KEY
```

Own-certificate chain

Our own certificate is used to identify our device to others when making or accepting TLS connections. The chain of own certificates consists of an order list of one of more links to stored certificates. The first certificate in the list (index 1) is used to communicate our public key to remote TLS devices. It is required that the private key linked by the TLS profile corresponds to the public key in this certificate. Therefore it is mandatory to configure a link to a private key on the TLS profile.

If you want to use a self-signed certificate, the chain of own certificates only contains one entry – a link to that certificate. Otherwise, if your certificate is signed by one or more levels of certification authorities, those intermediate certificates have to follow in the chain. That is, the next entry with index 2 links to the certificate of the certification authority that signed our own certificate at index 1. Then, the entry with index 3 links to the certificate of the next higher-level certification authority, and so on. The certificate of the root certification authority is not required and may be omitted or may be configured as the final entry in the list.

Mode: Profile tls

Step	Command	Purpose
1	**node(pf-tls)[name]# [no] own-certificate [index] pki: own-certificate /certificate**	Configures the chain of own certificates. Inserts the certificate at the specified index into the chain or appends it at the end if no index is specified. The sequence of own certificates is of importance! This configuration is mandatory.

Example: Configure a chain of own certificates.

In the following example, we assume that the own-certificate as well as all intermediate certificates that signed our own certificate already have been imported (e.g. with the **own** command). We configure our TLS profile with the certificate chain that represents our own certificate.

```
node(cfg)#profile tls MY_TLS
node(pf-tls)[MY_TLS]#own-certificate 1 pki:own-certificate/MY_CERTn
ode(pf-tls)[MY_TLS]#own-certificate 2 pki:own-certificate/CERT_OF_MY_CA
node(pf-tls)[MY_TLS]#own-certificate 3 pki:own-certificate/CERT_OF_MY_CAS_CA
```

Trusted Certificates

A set of trusted certificates is required to express the trust relationship to other devices. When our device wants to authenticate other devices our TLS profile must be configured with the certificates of all corresponding trusted root certification authorities. In this case a set of one or more trusted certificates is needed. However, when our device doesn't need to authenticate other devices no trusted certificates are needed.

As a default behavior all imported trusted certificates are considered trusted when not configured otherwise in the TLS profile. That is, trusted root certificates usually only have to be imported (e.g. with the **own** command) but must not be linked explicitly in the TLS profile.

Note Certificates stored as own certificates are not automatically considered as trusted certificates. If you want to use an own certificate as a trusted certificate as well, you have to copy it to the trusted-certificate section.

Mode: Profile tls

Step	Command	Purpose
1	**node(pf-tls)[name]# trusted-certificate all-stored**	Configures to use all imported trusted certificates for the current TLS profile. This is the default setting.

In the case that you want to create different TLS profile where each of them links to a different set of trusted certificates, you have to configure those links explicitly.

Mode: Profile tls

Step	Command	Purpose
1	**node(pf-tls)[name]# [no] trusted-certificate pki:trusted-certificate/certificate**	Adds or removes a certificate to or from the set of trusted certificates of the current PKI profile. Removing the last link sets the TLS-profile configuration back to the default setting, which is to use all trusted certificates in PKI.

Example: Configures a set of trusted profiles.

In the following example, we want to limit the set of trusted devices to those that have their certificate derived from one of the following root certificates.

```
node(cfg)#profile tls MY_TLS
node(pf-tls)[MY_TLS]#trusted-certificate pki:trusted-certificate/ROOT_CA_COMPANY_A
node(pf-tls)[MY_TLS]#trusted-certificate pki:trusted-certificate/ROOT_CA_COMPANY_B
node(pf-tls)[MY_TLS]#trusted-certificate pki:trusted-certificate/ROOT_CA_OUR_OWN_-
COMPANY
```

Authentication

TLS-licensed Patton devices can be configured whether or not to authenticate other devices when establishing TLS connections. If TLS authentication is to be enabled, it is required to have at least one trusted certificate installed on the device and to link the TLS profile to that certificate accordingly (see the description of the **trusted-certificate** command above).

Note This device can only authenticate other devices, but it cannot force other devices to authenticate us. Therefore mutual TLS authentication depends on having each device configured to authenticate the other devices.

Although it is technically possible to authenticate only TLS connections where we are the server side or to authenticate only TLS connections where we are the client side it is not recommended to do so, because in a SIP call, the client and server role may change during a call. (For example, we are the client side for an outgoing call, but the server side if the call is terminated by the remote party.)

Furthermore, having a different configuration for incoming and outgoing TLS connections opens the security hole of abusing connection-reuse for getting or receiving calls to or from unauthenticated remote devices.

Mode: Profile TLS

Step	Command	Purpose
1	**node(pf-tls)[name]# [no] authentication incoming**	Configures whether or not we are authenticating the remote client in a new TLS connection. This means that we are acting as TLS server for that connection. If enabled, we verify that the received certificate is signed by one of the trusted certificates listed in this TLS profile. Default is enabled.

Step	Command	Purpose
2	**node(pf-tls)[name][name]# [no] authentication outgoing**	Configures whether or not we are authenticating the remote server in a new TLS connection. This means that we are acting as TLS client for that connection. If enabled, we verify: • The received certificate is signed by one of the trusted certificates listed in this TLS profile • The *common name* field of the received certificate matches the remote IP address or DNS name of the remote TLS endpoint Default is enabled.

Cipher Suites

TLS automatically negotiates to use the crypto algorithms that provide the strongest security, meaning that the remote endpoint may lower the level of security by only providing a weaker crypto algorithm. In high-security environments it may be desirable to restrict the set of offered crypto algorithms. This can be done with the **cipher-suites** command.

Instead of listing each algorithm individually, Trinity offers a selection of pre-configured cipher-suites. Either one or multiple suites can be selected. The suits themselves may overlap with other cipher-suites. The available suites are:

- **ALL** All cipher suites except the eNULL cipher. This is the default setting.
- **DEFAULT** A pre-selected group of ciphers that offer medium to high level of security and at the same time maximum interoperability in most scenarios.
- **HIGH** ciphers with a key-size of 128 bit or more.
- **MEDIUM** ciphers with a key-size of 128 bit.
- **LOW** ciphers with a key-size of 64 or 56 bit.
- **SHA1** ciphers using SHA-1 as digest algorithm.
- **MD5** ciphers using MD5 as digest algorithm.
- **aRSA** ciphers offering RSA authentication.
- **kRSA** ciphers offering RSA key exchange.
- **eNULL** ciphers offering no encryption. Not recommended, use only for debugging.

Mode: Profile tls

Step	Command	Purpose
1	**node(pf-tls)[name]# cipher-suites (ALL | { DEFAULT | HIGH | MEDIUM | LOW | SHA1 | MD5 | aRSA | kRSA | eNULL }*)**	Configures one or multiple groups of cipher-suites to offer or accept. Default is ALL.

Compression

Compression has the benefit of reducing the size of the TLS packets and thus reducing the bandwidth needed for signaling over TLS. But it could have a negative effect on the performance for encrypting and decrypting the packet and therefore the load of SIP signaling traffic which can be handled on the device.

Mode: Profile TLS

Step	Command	Purpose
1	**node(pf-tls)[name]# [no] compression**	Configures if we are offering or accepting compression for TLS messages. Default is disabled.

Protocol

The Patton device supports SSL version 3 and its successor TLS version 1. The offering or acceptance can be configured separately for both of them. At least one of the Protocols need to be enabled for TLS operation.

Mode: Profile TLS

Step	Command	Purpose
1	**node(pf-tls)[name[name]# [no] protocol tls-v1**	Configures if we are offering or accepting the TLS protocol version 1. Default is enabled.
2	**node(pf-tls)[name]# [no] protocol ssl-v3**	Configures if we are offering or accepting the SSL protocol version 3. Default is enabled.

Configure TLS profile usage

To actually use TLS for SIP calls and/or registrations, a TLS profile must be bound to a SIP gateway. This allows to quickly changing between different TLS profiles. The DEFAULT profile may be bound if using the default private key and self-signed certificate is fine. Otherwise, if you want to configure a proper trust relationship or use different TLS profiles on different SIP gateways, you have to create new TLS profiles and use them on the respective SIP gateway with the **se c**ommand:

Mode: Context sip-gateway

Step	Command	Purpose
1	**node(sip-gw)[name]# use profile tls <name>**	Configures which TLS profile to use for connections from and to this SIP gateway. Default is the TLS profile DEFAULT.

Configure TLS transport

Some TLS-related parameters have to be configured outside the TLS profile. In order for the TLS protocol to be used on a SIP gateway, you have to enable it and specify the TCP port over which TLS connections are accepted (port 5061 by default).

It is possible to configure the availability of UDP/TCP, and TLS separately on the transport interface of the SIP gateway. This for example allows a secure configuration where both UDP and TCP are disabled, whereas TLS is enabled as the only transport protocol.

Note In order to use the SIPs URI scheme, TLS must be enabled on the transport interface of the SIP gateway.

Mode: interface sip-gateway

Step	Command	Purpose
1	**node(sip-if)[name]# [no] bind ipaddress [<context>] <interface> <label>**	Binds the transport interface to a specific IP address on an IP interface in the selected context.
2	**node(sip-if)[name]# [no] transport-protocol udp+tcp [<port>]**	Enables the reception and sending of SIP requests over UDP and TCP from and to the specified port. Default is enabled. Default port is 5060.
3	**node(sip-if)[name]# [no] transport-protocol tls [<port>]**	Enables the reception and sending of SIP requests over TLS from and to the specified port. Default is disabled. Default port is 5061.

Configure transport enforcement or fallback

Having multiple transport protocols available to choose from when sending a SIP message makes it desirable to select the order in which they are being used, or even the possibility to force one specific transport protocol. The transport enforcement or fallback configuration only has an effect to outgoing requests to the next hop device. It is not possible to prefer a transport protocol for incoming requests, but TLS can be forced by disabling UDP and TCP on the transport level altogether. Before you are able to force a transport protocol you have to enable the protocol on the transport level (see above). Otherwise requests cannot be sent over the protocol at all.

Note You are able to configure the device to only use SRTP when its session keys can be exchanged over a secure signaling connection. Forcing TLS as transport protocol here is not enough for SRTP to recognize a secure connection. The only way to actually prove SRTP that it is using a secure connection is to configure the SIPs URI scheme. By setting the URI scheme to SIPs, we also force the transport protocol to be TLS, too, but in addition, the transport protocol for SIP signaling messages is forced to be secure beyond the next-hop device. The same applies for having TLS as the preferred transport protocol, where additionally a local fallback to an insecure transport protocol could happen.

The command for selecting a preferred transport protocol allows you to select as a second choice the first fallback and as a third choice the second fallback protocol. As there is a maximum of three protocols the second fallback is implicitly given by choosing the preferred and the first fallback.

Note Not specifying the fallback protocols doesn’t disable them, but the order is derived from the internal default ordering. Specifying the transport protocols here does not enable them if they are disabled on the gateway‘s transport level. The preference of transport has no influence on calls with the URI scheme SIPs. Such calls are always forced to be signaled over TLS. The default ordering of preference is UDP, TCP, and TLS.

Mode: Call Outbound

Step	Command	Purpose
1	**node(callout)# [no] transport-protocol force { udp | tcp | tls }**	Forces outgoing INVITE requests to be sent over UDP, TCP, or TLS. If the according transport is not enabled on the sip-gateway level or the remote device does not support it there is no fallback to another transport.
2	**node(callout)# [no] transport-protocol prefer { udp | tcp | tls } ***	Sets the preferred transport protocol to choose if multiple are available for an INVITE request. Also specifies to which first and second alternative transport protocol we fall back. Default is to give preference to UDP, and to use TCP and then TLS as fallback.

The same protocol selection can be configured for outbound registrations as well by enforcing or preferring the transport protocol for REGISTER request. The same mechanism is applied for outbound registrations (REGISTER requests) as for outbound calls (INVITE requests).

In addition, we are able to control the transport protocol of incoming calls by the outbound registration; the registered contact address is stored on the server and remembers the transport protocol that was used to register the address. The server then uses this transport protocol to make calls, which the device will receive incoming calls for that registration with the transport protocol the registration was made with.

Mode: Registration outbound

Step	Command	Purpose
1	**node(regout)# [no] transport-protocol force { udp | tcp | tls }**	Forces outgoing REGISTER requests to be sent over UDP, TCP, or TLS. If the according transport is not enabled on the sip-gateway level or the remote device does not support it there is no fallback to another transport.
2	**node(regout)# [no] transport-protocol prefer { udp | tcp | tls } ***	Set the preferred transport protocol to choose if multiple are available for a REGISTER request. Also specifies to which first and second alternative transport protocol we fall back. Default is to give preference to UDP, and to use TCP and then TLS as fallback.

Configure Non-Default TLS Port Usage

Several SIP-related commands select the remote endpoint's host name and port number. All of them were extended for TLS and now allow you to optionally configure the remote TLS port next to the UDP/TCP port.

For most of these commands the port can be omitted because the standard SIP UDP and TCP port 5060 is used. For cases where the remote device does not use the standard port, the user must specify in the command. Now with the addition of TLS and the possibility to have a transport protocol fallback from TLS to UDP or TCP and vice versa, we need the ability to specify two ports: one for UDP and TCP and the other for TLS.

For all the following commands the same rule applies; the optional port parameter right after the ip-address parameter specifies the port number for UDP and TCP. If not indicated, the port number defaults to 5060. However, this first port parameter is never used for TLS, even if it is the only port specified explicitly, or if UDP and TCP transports are not available.

An additional parameter optionally configures the TLS port. If not indicated, the port defaults to 5061. This parameter is never used for UDP or TCP, even if it is the only port specified explicitly, or if the TLS transport is not be available.

Mode: Interface SIP

Step	Command	Purpose
1	**node(if-sip)[name]# [no] remote <host> [<port>] [tls-port <tls-port>]**	The existing remote command has the ability to specify a non-default UDP and TCP port and to specify a non-default TLS port. Default UDP/TCP port is 5060. Default TLS port is 5061.
2	**node(if-sip)[name]# [no] address-translation outgoing to-header host-part fix <host> [<port>] [tls-port <tls-port>]**	The existing address translation to-header command has the ability to specify a non-default UDP and TCP port and to specify a non-default TLS port. Default UDP/TCP port is 5060. Default TLS port is 5061.
3	**node(if-sip)[name]# [no]] address-translation outgoing request-uri host-part fix <host> [<port>] [tls-port <tls-port>]**	The existing address translation Request-URI command has the ability to specify a non-default UDP and TCP port and to specify a non-default TLS port. Default UDP/TCP port is 5060. Default TLS port is 5061.

Mode: Service sip-conference

Step	Command	Purpose
1	**node(svc-ls)[name]# [no] conference-server <host> [<port>] [tls-port <tls-port>]**	The existing conference-server command has the ability to specify a non-default UDP and TCP port and to specify a non-default TLS port. Default UDP/TCP port is 5060. Default TLS port is 5061.

Mode: Call Outbound

Step	Command	Purpose
1	**node(callout)# [no] force-destination <host> [<port>] [tls-port <tls-port>]**	The existing force-destination command has the ability to specify a non-default UDP and TCP port and to specify a non-default TLS port. Default UDP/TCP port is 5060. Default TLS port is 5061.
2	**node(callout)# [no] proxy <host> [<port>] [tls-port <tls-port>]**	The existing proxy command has the ability to specify a non-default UDP and TCP port and to specify a non-default TLS port. Default UDP/TCP port is 5060. Default TLS port is 5061.

Mode: Registration Outbound

Step	Command	Purpose
1	**node(regout)# [no] registrar <host> [<port>] [tls-port <tls-port>]**	The existing registrar command has the ability to specify a non-default UDP and TCP port and to specify a non-default TLS port. Default UDP/TCP port is 5060. Default TLS port is 5061.
2	**node(regout)# [no] force-destination <host> [<port>] [tls-port <tls-port>]**	The existing force-destination command has the ability to specify a non-default UDP and TCP port and to specify a non-default TLS port. Default UDP/TCP port is 5060. Default TLS port is 5061.
3	**node(regout)# [no] proxy <host> [<port>] [tls-port <tls-port>]**	The existing proxy command has the ability to specify a non-default UDP and TCP port and to specify a non-default TLS port. Default UDP/TCP port is 5060. Default TLS port is 5061.

Mode: Message Inbound

Step	Command	Purpose
1	**node(msgin)# [no] message-server <host> [<port>] [tls-port <tls-port>]**	The existing message-server command has the ability to specify a non-default UDP and TCP port and to specify a non-default TLS port. Default port is 5060. Default TLS port is 5061.

Configure SRTP

After having configured a secure URI scheme and a secure transport protocols for SIP signaling, you should consider securing the actual media-streams that carry voice or data by using SRTP. For more details, see "Secure Real-Time Protocol (SRTP) Configuration" on page 562.

Security consideration: Why you should use mutual TLS authentication

In the TLS profile, you are able to enable inbound and outbound TLS authentication separately (commands **authentication inbound** and **authentication outbound** respectively). This configuration affects how TLS connections are authenticated, i.e., whether the Patton device verifies TLS server certificates (for outgoing TLS connections) and/or whether it requests and verifies TLS client certificates (for incoming TLS connections). In this section, we will explain why you should enable TLS authentication in both directions for maximum security.

SIP gateway tries to re-use transport connections whenever possible; this is a desired and intended behavior according to the SIP standard. For example, if the Patton device opens ten calls to the exact same SIP server, it would be a waste of resources to establish a new TLS connection for each call. However, since each TLS connection requires CPU resources and network bandwidth to generate keys and negotiate security options, respectively.

The TLS authentication procedure, i.e. the verification of certificates, is only performed when the TLS connection is established. Since TLS connections can be re-used for SIP requests in both directions, not every single SIP call is authenticated. Furthermore, a TLS connection may be used to make calls (or other requests) in the opposite direction, which potentially cancels the intent of your authentication configuration.

IP does not Re-use Connections for Requests in Different Directions

A TLS connection may be opened in either direction during a call: one connection from the calling to the called party, and another one from the called to the calling party. Figure 1 depicts a common registration and call scenario without (left) and with (right) mutual TLS authentication.

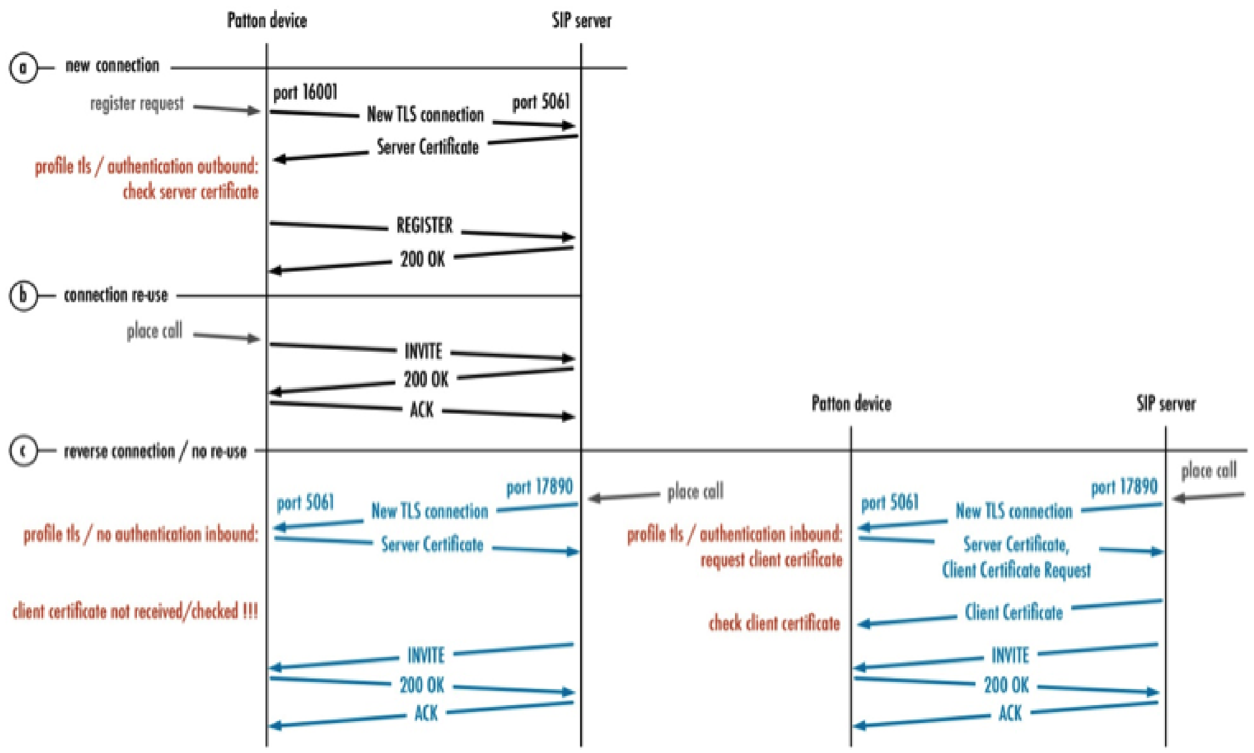


Figure 92. TLS Connection Establishments and Re-use in SIP

A. When the Patton device has to send a request such as a REGISTER, it may open a new TLS connection. Acting as a TLS client in this scenario, the Patton device uses an ephemeral port (e.g. 16001) and makes a connection to the default TLS port 5061 of the server (or any other port configured in the Request URI). The server then sends back its certificate. The authentication outbound command in the TLS profile decides whether or not our device verifies this certificate (i.e., whether or not it (i) compares the IP address or hostname of the server to the *canonical name* field of the certificate and (ii) verifies that the TLS profile lists in the list of trusted certificates the root CA of the received certificate of the server. Once the TLS handshake completed successfully, the REGISTER request and the corresponding response are sent over this established connection – no new TLS connection has to be opened for a response.

B. If the Patton device wants to send another request (such as an INVITE or re-REGISTER request) to the same SIP server, it re-uses the TLS connection and therefore does not verify the server's certificate again.

C. If the SIP server attempts to establish a call to the Patton device at a later point in time. The SIP server cannot re-use the connection, because the Request URI of the INVITE request requires the call to be connected to our local port 5061. Thus the SIP server creates a new TLS connection (blue). The Patton device is the server side for this TLS connection and the authentication inbound command in the TLS profile decides whether or not we request and verify the certificate of the TLS client (which is the SIP server in this case). Note that such a new connection is not only opened for a new call, but also for a new request during an existing call, such as a re-INVITE or a BYE request to renegotiate or drop the call, respectively.

Two Scenarios of TLS Authentication Vulnerability

In the above scenario, assume that **authentication outbound** s enabled but **no authentication inbound** has been configured (left part of figure 92). Let us focus on scenario c where the SIP server places a call towards our device. The SIP server certificate (= TLS client) is not verified since it could be a security issue.

Another, opposite scenario is to have enabled **authentication inbound** but **no authentication outbound** configured. For scenario C, where the SIP server places a call towards our device, we would verify the certificate of the SIP server (=TLS client). The SIP server certificate is no verified again if the server did not act according to the SIP standard and re-used an existing TLS connection we established before. Remember that the command **no authentication outbound** does not check the certificate for outbound TLS connections.

The recommendation is enabling both **authentication outbound** and **authentication inbound** together on the TLS profile whenever possible. The reason for separating the commands was to maximize compatibility in networks where the client does not have its own certificate.

The table below summarizes the vulnerability of the installation due to TLS connections being re-used according to the SIP standard:

Authentication Incoming	Authentication Outgoing	Vulnerability
Enable	Enable	None
Disable	Enable	Connection re-use outgoing
Enable	Disable	Connection re-use oncoming
Disable	Disable	Intentional no authentication

Conditions for the Patton Device to Re-use Connections

In order to be precise and complete, this sub-section defines the exact behavior of our SIP application in terms of re-using TLS connections:

- A new outgoing TLS outgoing is opened in the following case:
- SIP Request of a new SIP transaction (e.g. a re-INVITE or BYE request) needs to be sent over TLS, and
- There isn't an opened TLS connection that matches the following parameters:
  - The remote IP address of the connection is equal to the destination IP address of the request,
  - The remote port of the connection is equal to the destination port of the request, and
  - The local IP address of the connection is equal to the bound IP address of the gateway where the request is to be sent.

An existing TLS connection is re-used in the following case:

- The device sends an answer to a incoming request over TLS, or
- The device sends a request over TLS that belongs to an incoming or outgoing transaction, or
- The device sends a SIP Request for a new SIP transaction (e.g. a re-INVITE or BYE request) if there already exists an open TLS connection that matches the following parameters:
  - The remote IP address of the connection is equal to the destination IP address of the request,
  - The remote port of the connection is equal to the destination port of the request, and

- The local IP address of the connection is equal to the bound IP address of the gateway where the request is to be sent.

Caveat for TLS Profile

The TLS profiles have a limitation in distinctive usage. In certain circumstances, a wrong TLS profile is used for an outgoing connection.

The condition in which this issue was observed is:

- Having multiple SIP gateways with different TLS profiles
- The SIP gateways are bound to the same IP address
- A transaction-forming request needs to be sent over one of these gateways

In all other cases there is no issue at all, especially when one of these conditions is met:

- Having only one gateway
- Using the same TLS profile on all gateways
- Having all gateways bound to different IP addresses
- The TLS connection is opened in incoming direction

We plan to remove this limitation in a future software version.

Showing TLS profile

For debugging purposes, there are two commands that can be entered to take a closer look at the TLS profile internals.

Mode: Administrator executable

Step	Command	Purpose
1	**node# show profile tls**	Shows a list of all TLS profiles.
2	**node# show profile tls <name>**	Shows the specific TLS Profiles ready to be used in SIP.
3	**node# show profile sip-tls**	Shows a list of all TLS profiles ready to be used in SIP.
4	**node# show profile sip-tls <name>**	Shows the specific TLS profiles ready to be used in SIP, separated into client side and server side.

Troubleshooting

Listed below are common problems that will require a troubleshooting procedure:

- check license
- check time
- check certificates
- check TLS profile status

- check sip-gateway status
- check server name

Check License

In order to use TLS or SRTP for SIP a license has to be installed on the Patton device: “sip-tls-srtp”. This can be verified by showing all currently installed licenses by executing **show system licenses.** Verify that “IP TLS and SRTP” line is displayed.

```
node# show system licenses
Software Licenses
===============================================
                      SIP TLS and SRTP
```

Check time

The time settings can be verified by executing the command **show clock** It is important to have set the correct time for using TLS. The interesting entry here is “UTC Time”, and the entry “Default Offset” must be set correctly.

```
node# show clock
System Time Information
===============================================
DST Enabled          : false
Default Offset       : +02:00
Local Time           : Monday September 09 2013 08:30:02
UTC Time             : Monday September 09 2013 06:30:02
Clock                : 2013-09-09T08:30:02
```

An example of how to set the correct time:

```
node (cfg)# clock local default-offset +02:00
node (cfg)# clock set 2013-09-09T08:30:00
node (cfg)#
node (cfg)# show clock
System Time Information
===============================================
DST Enabled          : false
Default Offset       : +02:00
Local Time           : Monday September 09 2013 08:30:02
UTC Time             : Monday September 09 2013 06:30:02
Clock                : 2013-09-09T08:30:02
```

Check Certificates

he validity period of certificates can be examined by **show pki:own-certificates/<name> or how pki:trusted-certificates/<name>**

```
node# show pki:own-certificate/MY_CERT
Certificate:
   Data:
       Version: 1 (0x0)
       Serial Number:
           f8:9e:12:de:f9:a9:07:5d
       Signature Algorithm: sha1WithRSAEncryption
       Issuer: L=Bern, CN=192.168.10.10
       Validity
           Not Before: Jul 12 20:48:49 2013 GMT
           Not After : Jul  3 20:48:49 2014 GMT
       Subject: L=Bern, CN=192.168. 10.10
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
               Public-Key: (512 bit)
               Modulus:
                   00:e8:f8:07:3c:77:b4:c3:37:d6:58:11:66:fb:3d:
                   82:52:bf:5c:b3:d9:53:e8:eb:9b:3e:8a:e6:23:ab:
                   c7:b3:f0:6c:12:e3:97:63:42:ab:86:f2:13:f2:f1:
                   d7:24:04:af:a1:0e:3d:22:03:a9:60:73:aa:d4:cd:
                   70:82:50:08:55
               Exponent: 65537 (0x10001)
   Signature Algorithm: sha1WithRSAEncryption
       7e:ab:68:fb:d1:50:ce:5e:69:80:85:1f:73:84:ba:03:6a:76:
       6e:a9:ad:e3:27:08:67:cd:4d:7b:fe:51:56:4b:50:9b:37:90:
       25:f0:a2:78:73:33:84:31:09:08:75:b0:18:ad:7e:bc:8d:f5:
       8d:6b:60:20:32:61:ac:c6:10:9d
```

Check TLS Profile Status

The loading status of the TLS profile as it is seen by our SIP application can be examined by executing the command **show profile sip-tls <name>**. It is important here to check that the “Client status” is shown as “Client successful configured” and the “Server status” reported as “Server successful configured”.

If that is the case the following internal checks have been passed successfully:

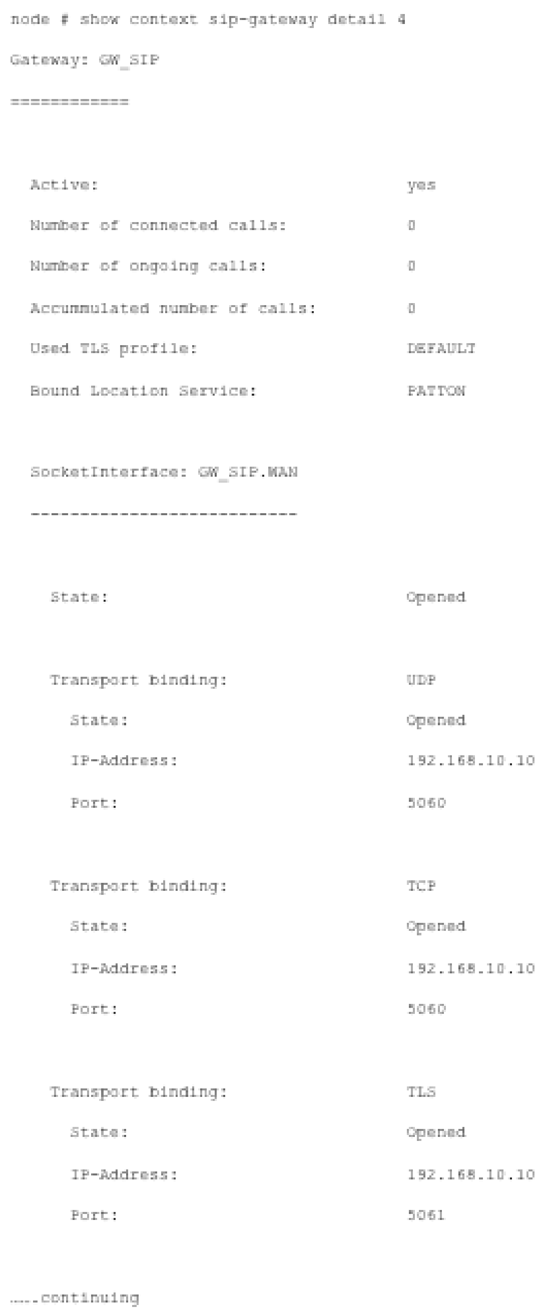
- The loaded certificates are syntactically valid
- The dependency of the own-certificate chain is correct
- The own-certificate has really been derived from the configured private key

In addition there is a line for each certificate with the status of the validity period, which should be shown as “OK”. The system time is within the validity period of that certificates at showing time.

```
node# show profile sip-tls DEFAULT

SIP-TLS Profile: DEFAULT

========================


  Used:                                   by 1 module(s)

  Client:                                 enabled

  Client status:                          Client successful configured

  Server:                                 enabled

  Server status:                          Server successful configured


  Client Context:

  ---------------


    TLS V1 Protocol:                      enabled

    SSL V3 Protocol:                      enabled

    Compression:                          enabled

    Peer Authentication:                  disabled

    Ciphers:                              ALL

    Validity of Own Certificates:         1

      Own Certificate 1:                  OK

    Validity of Trusted Certificates:     2

      Trusted Certificate 1:              OK

      Trusted Certificate 2:              OK


  Server Context:

  ---------------


    TLS V1 Protocol:                      enabled

    SSL V3 Protocol:                      enabled

    Compression:                          enabled

    Peer Authentication:                  disabled

    Ciphers:                              ALL

    Validity of Own Certificates:         1

      Own Certificate 1:                  OK

    Validity of Trusted Certificates:     2
```

Check SIP Gateway Status

The status of the SIP gateway can be examined by executing the command s**how context sip-gateway detail 4**. Note the “Used TLS profile” for TLS and the status of the transport binding TLS.



```
node # show context sip-gateway detail 4

Gateway: GW_SIP

============


  Active:                             yes

  Number of connected calls:          0

  Number of ongoing calls:            0

  Accummulated number of calls:       0

  Used TLS profile:                   DEFAULT

  Bound Location Service:             PATTON


  SocketInterface: GW_SIP.WAN

  ---------------------------


    State:                            Opened


    Transport binding:                UDP

      State:                          Opened

      IP-Address:                     192.168.10.10

      Port:                           5060


    Transport binding:                TCP

      State:                          Opened

      IP-Address:                     192.168.10.10

      Port:                           5060


    Transport binding:                TLS

      State:                          Opened

      IP-Address:                     192.168.10.10

      Port:                           5061


…….continuing
```

Check Server Name

When authenticating outbound while the TLS connection is opened from this device, the common name in the server certificate must match the host name of the request URI. To verify if this check fails or is successful for the TLS connection creation, the user will need to trace.

```
node # debug all-sip detail 5
```

Execute a test-call to trace the sip-signaling and the server-name check during the TLS handshake. Keywords to look for in the trace are:

- TLS handshaking request arrived
- TLS handshake approval for client connection

```
 [DBG] [STACK] TLS handshaking request arrived
# [DBG] [STACK]   bIsClient: yes
# [DBG] [STACK]   bPeerAuth: yes
# [DBG] [STACK]   Number of certificate(s) in the chain: 1
# [DBG] [STACK]   Certificate 0
# [DBG] [STACK]     Version: EVERSION_V3
# [DBG] [STACK]     Serial:
# [DBG] [STACK]     B7 0A E7 84 40 91 2A 2C
# [DBG] [STACK]     Not Before: 2000/6/2 23:40:15
# [DBG] [STACK]     Not After: 2999/10/3 23:40:15
# [DBG] [STACK]   rstrPeerHostname: 172.16.40.7
# [DBG] [STACK] No SubjectAlternateName Extensions in certificate
# [DBG] [STACK] No CommonName in certificate
# [DBG] [STACK] TLS handshake approval for client connection: NOT
APPROVED, PeerHostname does not match CommonName or SubjectAltName
```

There are three possible outcomes:

1. **A handshaking request visible and not approve.** If the TLS handshake request was not approved there wasn't a SubjectAlternateName or CommonName in the server certificate received which matched the PeerHostName.

- Certificate contains no SubjectAlternateName or CommonName: The servers certificate needs to be created new with specifying at least one of these parameters.
- Certificate contains a SubjectAlternateName or CommonName which does not match the PeerHostName: change address-translation in SIP call-control interface to match the resulting request URI of the INVITE to the SubjectAlternateName or CommonName.

2. **A handshaking request visible and approved.** If the TLS handshaking request was approved, continue troubleshooting by capturing Wireshark traces of the TLS connection establishment after getting a TLS handshake request. Probably the TLS connection establishment was successful and the failure happens later in normal SIP signaling

3. **No handshaking request visible:** If the section is not visible, continue troubleshooting by capturing Wireshark traces of the TLS connection establishment before getting a TLS handshake request.

Note If the SubjectAlternateName or CommonName in the certificate is a host name that needs to be resolved through DNS, the PeerHostName needs to be a DNS name too. The same applies for IP-addresses; if the SubjectAlternateName or CommonName in the certificate is an ip-address, the PeerHostName needs to be an ip-address as well. IP-addresses do not match the DNS names and vice versa.

Secure Real-Time Protocol (SRTP) Configuration

SRTP provides security, confidentiality, message authentication and replay protection for RTP and RTCP packets. This is achieved by generating a master key for each RTP stream, which is used to encrypt and decrypt the RTP packets. The keys are exchanged in the SDP content of the SIP signaling messages.

To protect the integrity of the SRTP master keys, securing the SIP signaling traffic is required too. This can be achieved by using TLS. Although not recommended it is still possible to use SRTP without TLS.

SRTP configuration task list

To configure the SRTP, perform the tasks in the following sections.

- License
- Information about using DSP channels
- Configure secure call signaling
- Configure RTP security
- Configure optional protection against key compromising through forking
- Configure optional protection against key compromising through call forward

License

The usage of a Patton device for doing SRTP, URI scheme sips and TLS are licensed. It is a precondition to have the license “sip-tls-srtp” installed on the Patton device before being able to perform all configuration steps.

Information about using DSP channels

A call encrypting its media-stream with SRTP always needs a DSP channel to perform the encryption of the outgoing SRTP stream and the decryption of the incoming SRTP stream. Therefore SRTP cannot be used on devices without DSP. The total number of DSP channels needed for one call depends on the involved protocols.

- For calls SIP to ISDN, SIP to FXS, SIP to FXO and vice versa there is always **one DSP channel** needed to do SRTP. That is the same number as when doing RTP without encryption.
- For calls SIP to SIP there are always **two DSP channels** needed to do SRTP. That is the same number as when doing transcoding from one codec to another codec.

When doing SRTP there is not the same possibility to save DSP channels in a SIP to SIP call as in a call without SRTP. The Patton device terminates always the SRTP streams.

Following an overview of using DSP channels in SIP to SIP calls:

RTP Security on SIP session toward A	RTP Security on SIP session toward B	Same or different codes on sessions	Used SDSP Channels
Unsecure RTP	Unsecured RTP	same	0
Unsecure RTP	Unsecure RTP	different	2
Unsecure RTP	SRTP	same	2
Unsecure RTP	SRTP	different	2

RTP Security on SIP session toward A	RTP Security on SIP session toward B	Same or different codes on sessions	Used SDSP Channels
SRTP	SRTP	same	2
SRTP	SRTP	different	2

Configure secure call signaling

The overall security of the SRTP master keys depends on having the sip signaling secured. Therefore, using the sips URI scheme is encouraged because it does force the sip signaling to be secured with TLS until reaching the destination domain. Using the sips URI scheme requires having TLS transport configured for SIP. TLS can be configured and used without URI scheme sips. But consider: TLS secures signaling only until the next hop and if the next hop is a proxy it could forward SIP signaling messages in an unsecured way. See chapter 50, "SIP Security" on page 564 for details about TLS and URI scheme.

Some of the RTP security commands modes are depending on the security of the call signaling. A call is considered as secure signaled if and only if the following two conditions are met.

- The URI scheme of the request URI is equal to "sips"
- The transport used for signaling is TLS

The reasoning why it is not enough to have TLS as transport is twofold. First TLS guarantees secure signaling only until the next hop and in case of a proxy the SRTP keys could be forwarded in an insecure manner toward the device terminating the SRTP stream. The second reason is the sequence of SDP offer construction and the selection of the transport protocol for signaling. The construction of the SDP offer happens first, for which we need to know if we want to offer SRTP. At this time the used URI scheme is known, but the finally selected transport could potentially change in the case of the URI scheme sip. It could be that a message is meant to be sent over an unsecured TCP transport, fails to be sent over TCP, and is finally sent over the secure transport TLS as a fallback.

Configure RTP security

There are 5 different modes how to support or force usage of SRTP:

- Disabled
- Preferred
- Enforced
- Preferred only for secured call signaling
- Enforced only for secured call signaling

Disabled: We don't offer SRTP and force the remote device to do unprotected RTP. If we get an offer with only SRTP it is rejected with a "488 not acceptable here" message.

Mode: Profile voip

Step	Command	Purpose
1	**node(pf-voip)[name]# no rtp security**	Disables the usage of SRTP at all and enforces to do unprotected RTP. This is the Default setting.

Preferred: We offer both, SRTP and unprotected RTP in the same offering and let the remote device decide on what to use. SRTP is the first in the ordering to indicate our preference for it. Incoming we accept any offer we receive: Only with SRTP, Only with unprotected RTP or offers witch both. In the last case we select SRTP in our answer to the remote device.

Mode: Profile voip

Step	Command	Purpose
1	**node(pf-voip)[name]# rtp security preferred**	Enables SRTP and unprotected RTP. If both are available SRTP is selected.

Enforced: We don't offer unprotected RTP and force the remote device to do SRTP. If we get an offer with only unprotected RTP it is rejected with a "488 not acceptable here" message.

Mode: Profile voip

Step	Command	Purpose
1	**node(pf-voip)[name] # rtp security forced**	Disables the usage of unprotected RTP at all and forces to do SRTP.

Preferred only for secured call signaling. The behavior depends on the security of the call signaling. Intention is to offer or accept SRTP only when the keys are protected through secure signaling. For secure calls the same rules apply as when the option Preferred would be selected. For non-secure signaled calls the same rules apply as when the option Disabled would be selected.

Mode: Profile voip

Step	Command	Purpose
1	**node(pf-voip)[name] # rtp security prefered sips**	Enabled the usage of SRTP only for secured calls. Do unprotected RTP for non-secure calls.

Enforced only for secured call signaling: The behavior depends on the security of the call signaling. Intention is to enforce secure media when the keys are protected through secure signaling. For secure calls the same rules apply as when the option Enforced would be selected. For non-secure signaled calls the same rules apply as when the option Preferred would be selected.

Mode: Profile voip

Step	Command	Purpose
1	**node(pf-voip)[name] # rtp security forced sips**	Enforce the usage of SRTP only for secured calls. Do offer and accept unprotected RTP and SRTP for non-secure calls.

Protection against key compromising through forking

In a forking scenario there is the possibility of a key being compromised. We are party A sending an INVITE containing the SRTP master key A to a proxy. The proxy forks the INVITE to party B and C and thus B and C getting to know key A, which is used to encrypt the SRTP stream toward party A. Now B picks up the call and C is dropped off the call. This results in a call where C knows the key A and can listen to the SRTP stream from B toward A.

To avoid this we as party A can create a new key A2 and send it upon connecting with a re-INVITE message to party B. This guarantees that as soon as both parties are connected and talking C cannot listen anymore to B's speech. The disadvantage is that because we as party A have no possibility to detect all forking cases, we need to generate a new key and re-INVITE upon connecting always. Therefore this behavior can be enabled or disabled.

Mode: Interface sip

Step	Command	Purpose
1	**node(if-sip)[name] # renew-srtp-keys-on-connect**	Configures to always generate new SRTP master keys upon connect and sending with a re-INVITE. Default is disabled.

Protection against key compromising through call forward

n a call forward scenario there is the possibility of a key being compromised. We are party A sending an INVITE containing the SRTP master key A to party B. Party B forwards the call toward party C by sending a "302 temporarily moved" answer toward us. We send in some cases now the same INVITE request with the same key A to party C. This results in a call where B knows the key A and can listen to the SRTP stream from C toward A. In other cases we generate a new INVITE request with a new key A2 to be sent to party C, which results in a call where party B does not know any key.

The behavior of generating a new key or sending the same key twice depends on the configuration of call reroute and call redirect. We send the same key only if the commands "new-session-after-redirect" and "call-reroute accept" are both disabled, which is the default setting. To ensure the generation of a new key enable either "new-session-after-redirect", "call-reroute accept" or both. Be aware that these commands existed already before and have other effects apart from influencing SRTP key generation. If you are in doubt enable only "new-session-after-redirect".

Mode: Interface sip

Step	Command	Purpose
1	**node(if-sip)[name] # new-session-after-redirect**	Configures to always generate a new INVITE request from scratch, when being forwarded. With the effect that a new SRTP key is generated too. Default is disabled.

Mode: Interface sip

Step	Command	Purpose
1	**node(if-sip)[name] # call-reroute accept**	Configures to forward the call toward call-control, when being forwarded. The call could be pushed back to another port or protocol. If the forwarded call is going to SIP again all rules the sip interface which is reached are applied. This could mean no SRTP is done at all. But if SRTP is offered again it is guaranteed to generate a new SRTP master key. Default is disabled.

Chapter 51 SIP Call-router Services

Chapter contents

Introduction

This chapter contains the description of all SIP specific call router services, which are only available if the software includes the SIP component.

SIP Conference-service

RFC4240 describes how to address different services on a media server without additional SIP headers or header parameters. This mechanism makes use of the fact media servers do not manage users, they manage media services. For that reason RFC4240 defines how to use the user part of the Request-URI as a service indicator. The conference-service provides the functionality described in RFC4240 as 'Conference Service'. It builds the Request-URI to address a conference room instance on a media server according to this standard. The user part of the Request URI will start with the service indicator 'conf=' followed by a unique conference room identifier. All calls with the same conference room identifier will attend the same session. The host part of the Request-URI that represents the media server host name has to be configured by the user. The conference room identifier and the called party number will be taken concatenated with the serial number of the device.

This looks as follows: conf=<called-nbr><serial-nbr>@<media-server>

SIP Conference-service Configuration Task List

The following section describes how to create a new conference-service and how to enter the configuration mode of an existing one. In addition it describes all commands and sub commands of the conference-service configuration mode. All configuration tasks for a conference-service are listed below.

- Enter conference-service configuration mode (see page 596)
- Configure the call routing destination (see page 596)
- Configure the conference server (see page 597)

Entering conference-service configuration mode

The **service sip-conference** command enters the configuration mode of an existing conference-service or creates a new one with a specified name. It also destroys an existing service by using the no form of the command.

Mode: Context CS

Step	Command	Purpose
1	**[name] (ctx-cs)[switch]#[no] service sip-conference** <*service-name*>	Creates/Destroys a SIP conference-service or enters the configuration mode of an existing one.

Configuring the call routing destination

The call routing destination specifies the next interface, table or service to which the current call has to be forwarded. Detailed information about call routing of a call control service can be found in chapter 45, "Call Router Configuration" on page 455. Because this is a SIP specific service, the user should be aware that the final destination of the configured call routing must be a SIP interface. This SIP interface will then set up the call to the configured conference server. Be aware that the elements in the routing path do not modify the Request-URI because it has been built by this service. Therefore the final SIP interface must not have configured a remote host.

Mode: Service SIP conference

Step	Command	Purpose
1	**[name] (svc-sip-conf)[name]#[no] route call dest-interface** *<name>* OR **dest-table** *<name>* OR **dest-service** *<name>*	Specifies the next call routing destination for an incoming call. The no form of the command deletes the current routing entry.

Configuring the conference server

As described in the introduction, one of the responsibilities of this service is to build the conference server Request-URI according to RFC4240. The user part will be built from the called-e164 property and the serial number whereas the host part is specified by this command.

Mode: Service SIP conference

Step	Command	Purpose
1	**[name] (svc-ls)[name]#[no] service sip-conference [name] conference-server** *<host-name>* **[***<port>***]**	Specifies the conference server host name and the port. The no form of the command removes the current entry.

SIP Location-service

Note To avoid possible name conflicts, two expressions will be declared here.
Location Service: Domain based identity data base.
Service Location-Service: A call-route service that accesses the Location Service declared above.

This service is the main consumer of the address bindings (mapping of the users identity to a contact address) deposited in the location service data base (see chapter 44, "Location Service" on page 434). Address bindings have been entered to location service data base either by inbound registration or manually by the user. An address binding is an entry that describes on which host a user is currently reachable. If a call is routed to this service, it performs a lookup with the requested URI to find the right contact information.

If the call is originated by the PSTN network, then this URI does not yet exist and must be built first with a mapping table. If this is not done, the lookup uses the called-e164 number and checks if a registered identity matches this number. For more information about this process, see section, "B2B User Agent with Registered Clients" on page 614.

It is also possible a user is registered with more than one contact. This can be if the user registers with its office SIP phone and also with its SIP soft client. In this case, the location-service's behavior can be configured. On the location-service, no routing command is available. The call will automatically be routed to that SIP interface on which the register request has been received, therefore manually entered address bindings must be provided with a SIP interface as routing destination.

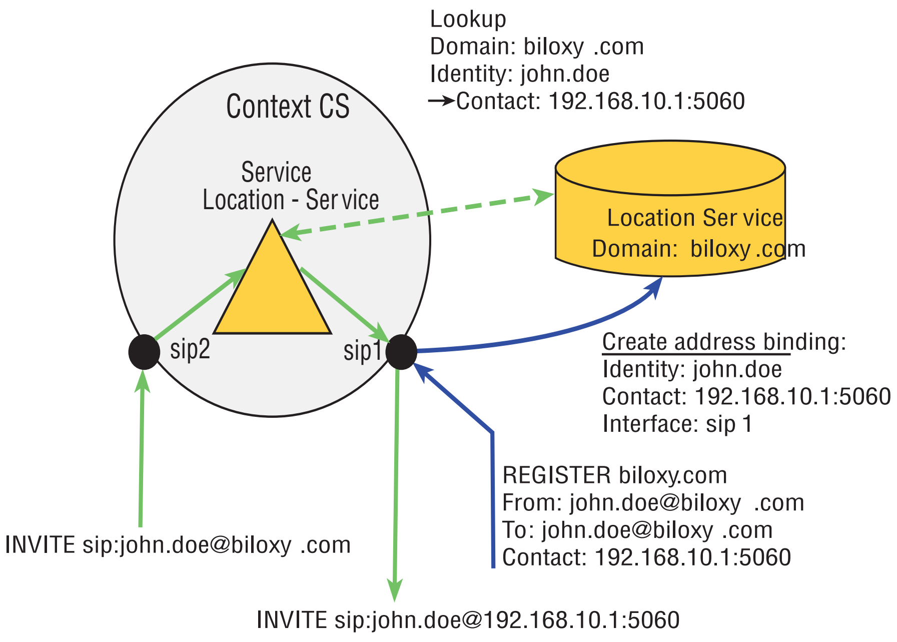


Figure 93. Registration and Lookup

SIP Location-service Configuration Task List

The following section describes how to create a new location-service and how to enter the configuration mode of an existing one. In addition, it describes all commands and sub commands of the location-service configuration mode. All configuration tasks for a location-service are listed below.

- Enter SIP location-service configuration mode (see page 598)
- Bind a location service (see page 599)
- Configure multi contact behavior (see page 599)
- Configure the hunt timeout (see page 599)

Entering SIP location-service configuration mode

The **service sip-location-service** command enters the configuration mode of an existing location-service or creates a new one with a specified name. It also destroys an existing service by using the no form of the command.

Mode: Context CS

Step	Command	Purpose
1	**[name] (ctx-cs)[switch]#[no] service sip-location-service** *<service-name>*	Creates/Destroys a SIP location-service or enters the configuration mode of an existing one.

Binding a location service

If a call is routed to the location-service it performs a lookup with the requested URI to the bound location service to find the address bindings. This command is optional because if no location service is bound, all existing location services will be considered to find the right contact information.

Mode: Service SIP location-service

Step	Command	Purpose
1	**device(sip-gw)#bind location-service <lsName>**	Binds the service to a location service. The no form of the command removes an existing binding. The [no-registration-outbound] command disables registration outbound.(default: enabled)

Configuring multi-contact behavior

There may be cases when a user has registered with several contacts in the location service database to be reachable on several SIP clients. If a call is routed to this service and the lookup in the location service database results in several contacts, this command specifies the behavior for such a case. One possibility is to contact all clients at the same time (distribute) or to contact the clients one after the other (hunt). At time of registration a priority value can be provided. In hunt mode, this priority defines the sequence of contacting the clients. The following modes are available:

- **Hunt:** Contact one registration after the other, depending on the priority. Highest priority first.
- **Distribute:** Contact all registrations at the same time.
- **Distribute and Hunt**: Registrations with the same priority build a distribution group. Hunt over this groups beginning with the highest priority.

Mode: Service SIP location-service

Step	Command	Purpose
1	**[name] (svc-ls)[name]#mode {hunt | distribute | distribute-and-hunt}**	Specifies the multi contact behavior of the SIP location-service. Default: distribute-and-hunt

Configuring the hunt timeout

The **hunt-timeout** command is used in 'hunt' mode and 'distribute-and-hunt' mode. It specifies the time that the service must hunt to the next contact if one is not responding. The user must configure this value because it is set per default to zero. This means that the hunting process will stay on the first contact and will never proceed with the next registered contact.

Mode: Service SIP location-service

Step	Command	Purpose
1	**[name] (svc-ls)[name]#[no] hunt-timeout** *<seconds>*	Defines the seconds to wait before hunting to the next contact or group of contacts. Default: zero (0)

Chapter 52 Context SIP Gateway Overview

Chapter contents

Introduction

This chapter provides an overview of the context SIP gateway. The main purpose of a context SIP gateway is forwarding and reception of SIP packets according to a RFC 3261 User Agent. In Trinity, the context SIP gateway represents the interface between the Call-Router (Context CS) and the IP Router (Context IP). It is responsible for dispatching incoming initial SIP requests to the right call control SIP interface and for the determination of the right IP interface for outgoing SIP messages. This is also called *SIP Transport Routing*. Figure 94 shows where the context SIP gateway is located in Trinity and how it interacts with other components.

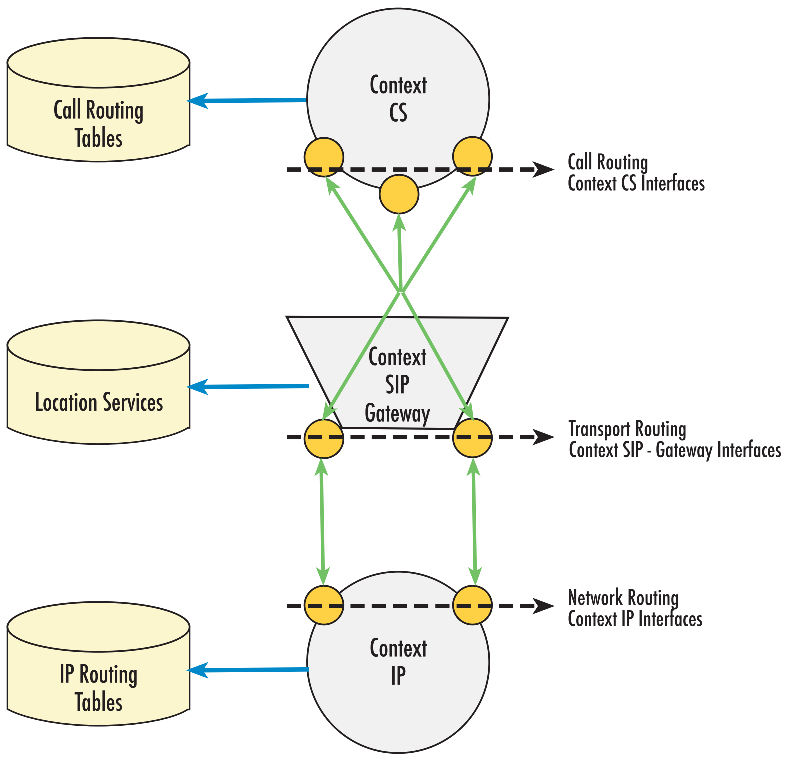

Call Routing Tables
Context CS
Call Routing Context CS Interfaces
Location Services
Context SIP Gateway
Transport Routing Context SIP - Gateway Interfaces
IP Routing Tables
Network Routing Context IP Interfaces
Context IP

Figure 94. Routing Architecture

There is a relationship between the call control SIP interface and the context SIP gateway. Every call control SIP interface must be bound to a context SIP gateway. One of the responsibilities of this interface is to build Request-URI and to determine a possible outbound proxy. If a proxy is available, it represents the next SIP hop and all outgoing messages will be sent to this host. If no proxy has been configured, the messages will be sent to the Request-URI's host. Depending on these two SIP parameters, the context SIP gateway chooses the right outgoing IP interface. The IP address of the outgoing IP interface will be placed in all SIP and SDP headers that need a direct routable contact address (VIA, Contact).

In the other direction, the context SIP gateway dispatches incoming SIP requests according to the Request-URI and the From-URI. The call control SIP interface has configuration parameters called “remote” and “local” to build the Request-URI, the To-URI and the From-URI for outgoing requests. These parameters will also used for identifying the best matching call control SIP interface for incoming requests according to the following rule:

1. **From-URI-Host** *equal* **Remote**
   and
   Request-URI-Host *equal* **Local**
2. **From-URI-Host** *equal* **Remote**
3. **Request-URI-Host** *equal* **Local**
4. No match, the first configured will be taken

For detailed information about call control SIP interface configuration, see chapter 48, "SIP Interface Configuration" on page 538.

Context SIP Gateway Configuration Task List

The following section describes how to create a new context SIP gateway and how to enter the configuration mode of an existing context SIP gateway. Additionally, it describes all commands and sub commands of the context SIP gateway configuration mode. All configuration tasks for a Context SIP Gateway are listed below.

- Create a context SIP gateway (see page 603)
- Create a transport interface (see page 603)
- Configure the IP binding (see page 604)
- Configure a spoofed contact address (see page 604)
- Bind location services (see page 606)
- Setting a Traffic Class
- Configuring Quality of Protection in SIP Authentication
- Enable/Disable (see page 607)

Creating a Context SIP Gateway

The **context sip-gateway** command enters the configuration mode of a context SIP gateway. If the requested one does not exist, a new one will be created. The no form of the command removes an existing context SIP gateway. This command can be entered without specifying a name. In this case, the default name *sip* will be taken.

Mode: Configure

Step	Command	Purpose
1	[*node*]**(cfg)# [no] context sip-gateway [<*name*>]**	Creates/Destroys a context SIP gateway or enter configuration mode.

Creating a Transport Interface

The **interface** command enters the configuration mode of a transport interface. If the requested interface does not exist, a new one will be created. The no form of the command removes an existing transport interface. A maximum of 2 transports interfaces per SIP Gateway can be configured. This limitation is due to the fact, that a Gateway can only be bound to one IP address type (IPv4 or IPv6).

Mode: Context SIP Gateway

Step	Command	Purpose
1	[*node*]**(sip-gw)[name]# [no] interface** *<name>*	Creates/Destroy a transport interface or enter configuration mode.

Configuring the IP Binding

The following commands specify the ip parameters for the transport layer. A listening socket for UDP and TCP will be created according to the entered parameters. If the port parameter has not been specified default SIP port 5060 will be taken.

Mode: Transport Interface

Step	Command	Purpose
1	[*node*]**(sip-if)[name]# [no] bind** OR [*node*]**(sip-if)[name]#bind ipaddress [ip-context]** *<ip-interface> <ip-address-label>* **[port** *<port>* **]**	Binds the transport interface to the IP layer or removes an existing binding.

Configuring a Spoofed Contact Address

The SIP contact-header and via-header can be independently spoofed using their corresponding option.

Mode: Context sip-gateway/interface

Contact-Header

Step	Command	Purpose
1	**device(sip-if)[<name>]# [no] spoofed contact-header (auto | manual <host>) [port <port>]**	Configures the public visible contact-header. Auto specifies that the NAT address shall be detected and then inserted into the field.

Via-Header

Step	Command	Purpose
1	**device(sip-if)[<name>]# [no] spoofed via-header (auto | manual <host>) [port <port>]**	Configures the public visible via-header. Auto specifies that the NAT address shall be detected and then inserted into the field.

Nat-Address

Step	Command	Purpose
1	**device(sip-if)[<name>]# [no] spoofed nat-address (auto | manual <ip>) [port <port>]**	Configures the public nat address used in sip requests. Showing up in via-header, contact-header and SDP content. Auto specifies that the NAT address shall be detected and then inserted into the field.

Note Public NAT addresses are detected by sending http requests to checkip.dyndns.org, port 80.

If the device is located behind a NAT, this command can be used to provide the Contact (SIP/SDP) and VIA headers of outgoing requests with the public ip address. The IP address may either be specified directly or the address of the NAT may be detected.The port parameter in the spoofed commands is optional. If not specified in the spoofed command, the same port as configured in the context SIP gateway interface is taken. If no port is specified in the interface either, then no port is explicitly signaled which should result in the use of the default port.

Note The options contact-header and via-header can be used together, while the option nat-address can only be used alone.

Mode: SIP Interface

Step	Command	Purpose
1	**[node](sip-if)[name]# [no] spoofed { nat-address } { auto | manual } [<ip address>]**	Applies a spoofed address to the via-header, contact-header and sdp-contact fields to the SIP interface. Alternatively, *auto* specifies that the NAT address shall be detected and then inserted into the field.

Note Public NAT addresses are detected by sending http requests to checkip.dyndns.org, port 80.

Binding Location Services

The **bind location-service** command binds predefined location services to the context SIP gateway. The no form of the command removes a bound location service. All bound location services define the domains and identities for the context SIP gateway. Identity features, such as outbound registration, inbound registration, and authentication, may or may not be executed, depending on the specific configuration for the identity. Also, they provide transport properties, like Proxy or Traffic Class, and media configuration parameters, like VoIP Profile, SIP Profile or Tone Profile. For more information about configuration of location services and identities, see chapter 44, "Location Service" on page 434.

Mode: Context SIP Gateway

Step	Command	Purpose
1	[*node*]**(sip-gw)[name]# [no] bind location-service** *locationservice*	Add a location service binding to this context SIP gateway or removes an existing one.

SIP Trusted Host Behavior

In context CS`s SIP interfaces, SIP hosts can be set to trusted. By default the SmartNode responds with a "SIP/2.0 503 Service Unavailable" to SIP requests received from hosts which are not trusted. With the [no] form of the command below, the SmartNode will not send any message to untrusted hosts.
Mode: context sip-gateway

Step	Command	Purpose
1	[*node*]**(sip-gw)[sip]# [no] answer-untrusted-hosts**	Enables "503 Service Unavailable" response for untrusted hosts Default: enabled

Setting a Traffic Class

This feature allows you to directly configure the SIP application with a specific traffic-class. All local generated SIP packets will automatically be marked with the configured traffic-class without the need of applying specific classification rules.

Mode: context sip-gateway

Step	Command	Purpose
1	[*node*]**(sip-gw)[name]#traffic-class <tc>**	Specify specific traffic-classes for the SIP application. Default: local-default

Configuring Quality of Protection in SIP Authentication

For inbound authentication, when the context sip gateway is challenging a request from a user, the quality of protection can be configured. There are four options:

- **none:** This is the default setting and no quality of protection is required. For backward compatibility the request-digest is calculated according RFC 2069.

- **auth:** The quality of protection is set to “Authentication”. The request-digest is calculated according RFC 2617.
- **auth-int:** The quality of protection is set to “Authentication with integrity protection”. The request-digest is calculated according RFC 2617 and the entire body of the message is part of the request-digest calculation.
- **both:** The quality of protection can be either “Authentication” or “Authentication with integrity protection”.

Mode: Context SIP Gateway

Step	Command	Purpose
1	**[*node*](sip-gw)[name]# quality-of-protection (none | auth | auth-int | both)**	Configures the quality of protection for inbound authentication. Default: none

Enabling/Disabling the Context SIP Gateway

The **shutdown** command disables the context SIP gateway. The **no shutdown** command enables the context SIP gateway.

Mode: Context SIP Gateway

Step	Command	Purpose
1	**[*node*](sip-gw)[name]# [no] shutdown**	Disables the Context SIP Gateway. The no form of the command enables the Context SIP Gateway.

SIP notify check-sync event

It is possible to trigger auto-provisioning with a SIP NOTIFY request. Basically the SIP gateway has to be configured to accept the related packet which will trigger an action script event when the proper information arrived. Finally the action will execute the configured provisioning. The notify packet has to contain the following information:

- A required “Event” header which must be “check-sync”
- An optional “reboot” parameter can be set inside the “Event” header. The possible values are: “true” or “false”. If this parameter is missing then true will be assumed.

Mode: context sip-gateway

Step	Command	Purpose
1	**[node](sip–gw)[GW_SIP]#[no] notify check-sync accept**	Configures whether the notify check-sync packet will be accepted or not. Default: disabled

Mode: actions

Step	Command	Purpose
1	**[node](act)# rule RULE**	Defines the rule under which the condition has to be checked.

Step	Command	Purpose
2	**[node](act)[RULE]# condition sip <gateway: gatewayname> { NOTIFY_CHECK_SYNC _NORELOAD | NOTIFY_CHECK_SYNC_ RELOAD}**	Defines the condition under which the action has to be executed.
3	**[node](act)[RULE]# action "provisioning execute <pf-name>"**	Sets what should happen at the end of provisioning.

Mode: profile provisioning

Step	Command	Purpose
1	**[node](pf-prov)[<pf-name>]# [no] activation reload { deferred | graceful | immediate }**	Configures whether the notify check-sync packet will be accepted or not. Default: disabled

Troubleshooting

Show Status Information

Mode: Administrator Execution

Step	Command	Purpose
1	[*node*]**#show context sip-gateway** [*gw-name*] **[detail** <*level*>**]**	Displays status and configuration information about a context sip-gateway and its bound resources like the call-control sip interfaces.

Debug Commands

Mode: Administrator Execution

Step	Command	Purpose
1	[node]**#debug sip-datapath [detail** <*level*>**]**	Logs information related to the media channels
2	[node]**#debug sip-registration [detail** <*level*>**]**	Logs information about user registration activities
3	[node]**#debug sip-signaling [detail** <*level*>**]**	Logs call signaling related information
4	[node]**#debug sip-transport [detail** <*level*>**]**	Logs all SIP messages sent or received over the IP network

Configuration Examples

Example 1

This is the minimal configuration of a working context sip-gateway. It has one transport interface that is bound to the ip address LAN. With this configuration, the context sip-gateway processes all SIP requests addressed to port 5060.

```
context sip-gateway SIP-GW
```

```
    interface lan
      transport-protocol udp+tcp 5060
      bind ipaddress ROUTER LAN LAN
  context sip-gateway SIP-GW
    no shutdown
```

Example 2

The following example shows a context sip-gateway that is explicitly bound to a LAN-Side and a WAN-Side ip interface. This could be a Back-To-Back User Agent application that interconnects a private SIP environment and a public SIP environment.

```
context sip-gateway SIP-GW
  interface lan
    transport-protocol udp+tcp 5060
    bind ipaddress ROUTER LAN LAN
  interface wan
    transport-protocol udp+tcp 5060
    bind ipaddress ROUTER WAN WAN
context sip-gateway SIP-GW
  no shutdown
```

Example 3

If special features like Outbound Registration, Inbound Registration or Authentication is required, one or more location services must be configured that enables theses capabilities in general, for a group of identities or just for a single identity. These location services must be bound to the context sip-gateway that is responsible to manage the location service's domains. For detailed description about location service configuration, see chapter 43, "Authentication Service" on page 431.

```
context sip-gateway SIP-GW
  interface lan
    transport-protocol udp+tcp 5060
    bind ipaddress ROUTER LAN LAN
  interface wan
    transport-protocol udp+tcp 5060
    bind ipaddress ROUTER WAN WAN
context sip-gateway SIP-GW
  bind location-service SIP_LS_1
  no shutdown
```

Applications

Outbound Authentication

The back-to-back user agent can provide credentials for authentication on another sip user agent or proxy. The username and password used for authentication must be configured in an authentication-service. If one or more realms are configured in the authentication-service, the credentials are only provided to challenges containing one of these realms. If no realm is configured, the credentials are provided to any realm.

In an authentication-service, there can be multiple usernames and passwords. An identity which should authenticate can direct the authentication outbound face to a pair of credentials. There can be multiple identities using the same credentials. An identity can also point to multiple credentials, but each of these credentials needs to be in another authentication-service with another realm. It is possible to authenticate to multiple realms with multiple credentials at the same time.

If the gateway has to provide credentials for unknown identities or for any identity which belongs to a certain domain, there can be a DEFAULT identity-group configured. The authentication credentials configured in the identity-group DEFAULT are used for any identity in this location-service that is not explicitly configured.

```
authentication-service AUTH_PATTON
  realm patton.com
  username hermes password Wh6Xbk9G= encrypted
  username john password Fa0Y9e4L= encrypted

authentication-service AUTH_ANY
  username bob password Co7s3bUp= encrypted

location-service PATTON
  domain voice-cloud.com
  domain patton.com

  identity-group default
    authentication outbound
      authenticate 1 authentication-service AUTH_ANY username bob

  identity 400
    authentication outbound
      authenticate 1 authentication-service AUTH_PATTON username hermes
      authenticate 2 authentication-service AUTH_ANY username bob

  identity 500
    authentication outbound
      authenticate 1 authentication-service AUTH_PATTON username hermes

  identity 600
    authentication outbound
      authenticate 1 authentication-service AUTH_PATTON username john
```

If the gateway needs to provide authentication credentials on a sip request, the following procedure takes place:

1. Determine the location-service which provides credentials. The domain of the location service must match the host part of the from-uri and the location-service is bound to the context sip-gateway which sends the request.
2. Determine the identity which provides credentials. The name or the alias of the identity must match the user part of the from-uri. If there is no identity that matches and an identity-group with the name DEFAULT is configured, the identity-group DEFAULT is taken.
3. Determine the authentication-service which provides credentials. The authentication entries of the taken identity or identity-group are searched for an authentication-service that matches exactly the realm requested in the answer to our request. Then this authentication service is taken. If no match was found, an authentication service with no realm configured is taken.

4. Determine the authentication username which provides credentials. If the authentication entry of the identity which configures the taken authentication service has also configured a username this username is taken. If there is no username configured the name of the identity is taken as username.
5. Take the credentials in the authentication service with the according username and provide username and password for re-issuing the request.

If one of these steps has no result and fails, authentication is not possible for that request.

Inbound Authentication

The back-to-back user agent can challenge another sip user agent or proxy for authentication credentials. The username and password used for challenges must be configured in an authentication-service. There must be at least one realm configured in the authentication-service. The first realm configured is used for challenging requests.

In an authentication-service, there can be multiple usernames and passwords. An identity which should be challenged can direct the authentication inbound face to a pair of credentials. There can be multiple identities using exactly the same credentials. An identity can also point to multiple credentials, but only the first entry is used for challenging. If an identity points to multiple credentials, any of these credentials are accepted in the answer as long as it is valid for the challenged realm.

If the gateway has to challenge credentials for unknown identities or for any identity which belongs to a certain domain, there can be a DEFAULT identity-group. The challenging credentials configured in the identity-group DEFAULT are used for any identity in this location-service that is not explicitly configured.

```
authentication-service AUTH_PATTON
  realm patton.com
  username kevin password Wh6Xbk9G= encrypted
  username dirk password Fa0Y9e4L= encrypted
  username boss password Q9Gns6Nd4= encrypted

location-service PATTON
  domain patton.com

  identity-group default
    authentication inbound
      authenticate 1 authentication-service AUTH_PATTON username kevin

  identity 400
    authentication inbound
      authenticate 1 authentication-service AUTH_PATTON username kevin
      authenticate 2 authentication-service AUTH_PATTON username dirk

  identity 555
    authentication inbound
      authenticate 1 authentication-service AUTH_PATTON username boss
```

If the gateway receives an incoming request without credentials, the following procedure takes place:

1. Determine the location-service which challenges credentials. The domain of the location service must match the host part of the request-uri and the location-service is bound to the context sip-gateway which handles the request.

2. Determine the identity which challenges credentials. The name or the alias of the identity must match the user part of the request-uri. If there is no identity that matches and an identity-group with the name DEFAULT is configured, the identity-group DEFAULT is taken.
3. Determine the authentication-service which challenges credentials. The first authentication entry of the taken identity or identity-group where a realm is configured is taken.
4. The first realm in the authentication-service is used for challenging the request. If no realm was found no challenging is possible.

If one of these steps has no result and fails, challenging is not possible for that request and the request is accepted.

If the gateway receives an incoming request with credentials, the following procedure takes place:

1. Determine the location-service which challenges credentials. The domain of the location service must match the host part of the request-uri and the location-service is bound to the context sip-gateway which handles the request.
2. Determine the identity which challenges credentials. The name or the alias of the identity must match the user part of the request-uri. If there is no identity that matches and an identity-group with the name DEFAULT is configured, the identity-group DEFAULT is taken.
3. Check credentials. All authentication entries of the taken identity or identity-group are compared with the provided credentials. If one matches the request is accepted.
4. An authentication entry matches if the username and password matches with the configured credentials and one realm in the authentication-service match exactly the realm challenged or the authentication-service has no realm configured.

If one of these steps has no result and fails, the request is treated as a request without credentials.

Outbound Registration

The back-to-back user agent can register identities on a sip registrar. Therefore, a registration outbound face with the register “auto” command is needed. The address of record is built with the domain of the location-service and the name of the identity. The REGISTER request is sent to the configured registrar or, if missing, to the domain configured in the location-service.

If the registration is successful, the registrar forwards requests that are destined to the address of record to the back-to-back user agent.

```
location-service PATTON
  domain patton.com

  identity-group DEFAULT
    registration outbound
      register auto

  identity 400
    registration outbound
      registrar sip.patton.com
      register auto

  identity 555 inherits DEFAULT
```

```
context sip-gateway GW_SIP

  sip-interface SIP_WAN
    transport-protocol udp+tcp 5060
    bind ipaddress ROUTER IF_WAN IF_WAN

context sip-gateway GW_SIP
  bind location-service PATTON
```

If an identity is added to a location-service which is bound to a context sip-gateway, the following procedure takes place:

1. Determine if identity should be registered.

- The location-service containing the identity must have at least one domain configured.
- The identity must have a registration outbound face configured or inherited.
- The register command must be set to "auto".

2. Build the address of record to register. The name of the identity builds the user-part. The first domain configured in the location-service builds the host-part.
3. Build the address of the registrar. The registrar configured in the registration outbound face is taken as request-uri. If no registrar is configured the first domain configured in the location-service builds the request-uri.
4. Build expire header. If a lifetime is configured in the registration outbound face an expire header with the desired lifetime is added.
5. Build contact address to register. The spoofed-contact parameter of the sip-interface in the context sip-gateway through which the REGISTER request is sent is set as contact address. If no spoofed-contact is configured the ip-address and port of the sip-interface in the context sip-gateway through which the REGISTER request is sent builds the contact address.
6. Send the REGISTER request.

If one of these steps has no result and fails, the registration fails. After a certain timeout (which is configurable in the registration outbound face), the request is re-issued.

Inbound Registration

With the according license, the back-to-back user agent can allow registrations from other user agents. Therefore, identities must be configured with a registration inbound face. Contacts to forward requests for this identity can be configured or added dynamically with REGISTER requests.

If the gateway has to accept register requests for unknown identities or for any identity that belongs to a certain domain, there can be a DEFAULT identity-group configured. The configuration of the identity-group DEFAULT with the registration inbound face allows registration for any identity in this location-service that is not explicitly configured.

```
location-service PATTON
  domain patton.com

  identity-group DEFAULT
    registration inbound
  identity 400
    registration inbound
      lifetime default 4000 min 600 max 36000
      contact 172.16.40.22 switch IF_SIP priority 500

context sip-gateway GW_SIP

  sip-interface SIP_WAN
    transport-protocol udp+tcp 5060
    bind ipaddress ROUTER WAN WAN

context sip-gateway GW_SIP
  bind location-service PATTON
```

If the gateway receives an incoming REGISTER request, the following procedure takes place:

1. Determine to which sip interface in the context cs the request should be forwarded. This happens according the same rules as an incoming INVITE is forwarded. Outgoing calls to the registered contacts will pass through the same sip interface as the incoming REGISTER request.
2. Check request-uri. The host part of the request-uri must match a domain of a location-service which has configured imperative “authoritative” and is bound to the context sip-gateway which received the request.
3. Check to header. The host part of the to-uri must match the host part of the request-uri.
4. Check if registration is allowed. There must be an identity in the location-service that match with name or alias the host part of the to-uri or a identity-group DEFAULT with a registration inbound face configured.
5. Create a dynamic identity if the identity to register does not already exist. This happens when the identity-group DEFAULT is configured with a registration inbound face. The dynamic created identity inherits from the identity-group DEFAULT.
6. Add all contacts with expires¦0 to the requested identity. The expiration time is taken out of the expire parameter from the contact or from the expire header. This time is adjusted to fit into the configured lifetime boarder. If there is no expires parameter the default expired from the configured lifetime is taken.
7. Remove all contacts with expires=0 from the requested identity.
8. Remove a dynamic identity if the identity contains no more contacts.
9. Return 200 OK with all contacts registered or configured from the requested identity

If one of these steps fails the registration fails and an according message is sent. A registered contact is removed out of the location-service after the expiration time has passed and the registration was not refreshed. The “SIP Location-service Configuration Task List” (on page 598) in the context cs is able to forward calls to the registered contacts.

B2B User Agent with Registered Clients

The sip-location-service in the context cs is used to route calls according to the contact bindings. Calls can be routed from other services, interfaces or tables to the sip-location-service. The sip-location-service routes the call directly to the sip-interface over which the according contact was registered. If there are some mappings or adjustments of call parameters needed, this has to be done before routing to the sip-location-service. The address-translation of the destined sip interface is the only mapping that happens after the sip-location-service.

If calls from an ISDN, FXS or FXO interface are routed to the sip-location-service, it is necessary to bind a location-service to the sip-location-service because the domain information is needed. An alternative way would be to map a complete sip-uri to the call destination properties.

The mode command in the sip-location-service configures the behavior of the service when multiple contacts are registered for one address of record. In "distribute" mode, the call is distributed to all contacts at the same time. The first phone that picks up receives the call. In "hunt" mode, one contact after each other is called, and the hunt-timeout parameter defines how long to wait until proceeding to the next contact. The contacts with higher priority are hunted first. In "distribute-and-hunt" mode, all contacts with the same priority are distributed together and the hunt searches from the higher priority contacts to the lower priority contacts. If no hunt-timeout is configured, there is no hunting.

```
context cs switch
  interface sip IF_SIP
    bind context sip-gateway GW_SIP
    route call dest-service SERVICE_SIP
    remote patton.com
    local patton.com

  interface isdn IF_ISDN
    route call dest-service SERVICE_SIP

  service sip-location-service SERVICE_SIP
    bind location-service PATTON

location-service INALP
  domain patton.com

  identity-group DEFAULT
    registration inbound

context sip-gateway GW_SIP

  sip-interface SIP_WAN
    transport-protocol udp+tcp 5060
    bind ipaddress ROUTER WAN WAN

context sip-gateway GW_SIP
  bind location-service PATTON
```

If the sip-location-service receives a call, the following procedure takes place:

1. Determine the requested domain. If none of these three possibilities matches the call is dropped.

   If the call has a destination-uri set the host part of that uri is taken as requested domain.

   If there is no destination-uri set, but a destination-ip-address, this is taken as requested domain.

   If there is no destination-uri and no destination-ip-address set, but a location-service bound, the requested domain is taken from there.

2. Determine the requested user.

   If the call has a destination-uri set the user part of that uri is taken as requested user.

   If there is no destination-uri set, the destination-e164 is taken as requested user.

3. Determine the location-service. The location-services are checked if one domain matches the requested domain and if the imperative of the location-service is “authoritative”. If a location-service is bound and it does not match for the requested domain the call is dropped.
4. Determine the identity. The location-service is searched for an identity where the name or an alias matches the requested user.
5. Get the registered and configured contacts of that identity. The contacts are sorted according to the priorities.
6. Distribute or Hunt the contacts according the configured mode.

If one of these steps fails, the call fails. It is recommended to configure a hunt-group in front of that service to have a fall-back when the call fails because the requested identity is not registered.

Chapter 53 ISDN Overview

Chapter contents

Introduction

This chapter provides an overview of ISDN ports and describes the tasks involved in configuring ISDN ports in Trinity.

ISDN ports are the physical ISDN connections on the Patton devices. There are two types of ISDN ports:

- The ISDN basic rate interface (BRI), and
- The ISDN primary rate interface (PRI).

A BRI port supports two 64kbit/s B-channels for switched voice or data connections, one 16kbit/s D-channel for signaling and always-on data transfer. BRI ports are sometimes called S0 ports. The related PSTN access service is also called Basic Rate Access (BRA).

The PRI port supports thirty 64kbit/s B-channels, one 64kbit/s D-channel and one synchronization timeslot on a standard E1 (G.704) while supporting 23, 64kbit/s B-Channels and one 64kbit/s D-Channel on a standard T1 (G.703) physical layer.

ISDN Reference Points

The ISDN standards define a number of reference points on the interfaces between the various equipment types on an ISDN access line. figure 95 illustrates these reference points. The understanding of these reference points and where they are located is necessary for the configuration of the devices' ISDN ports.

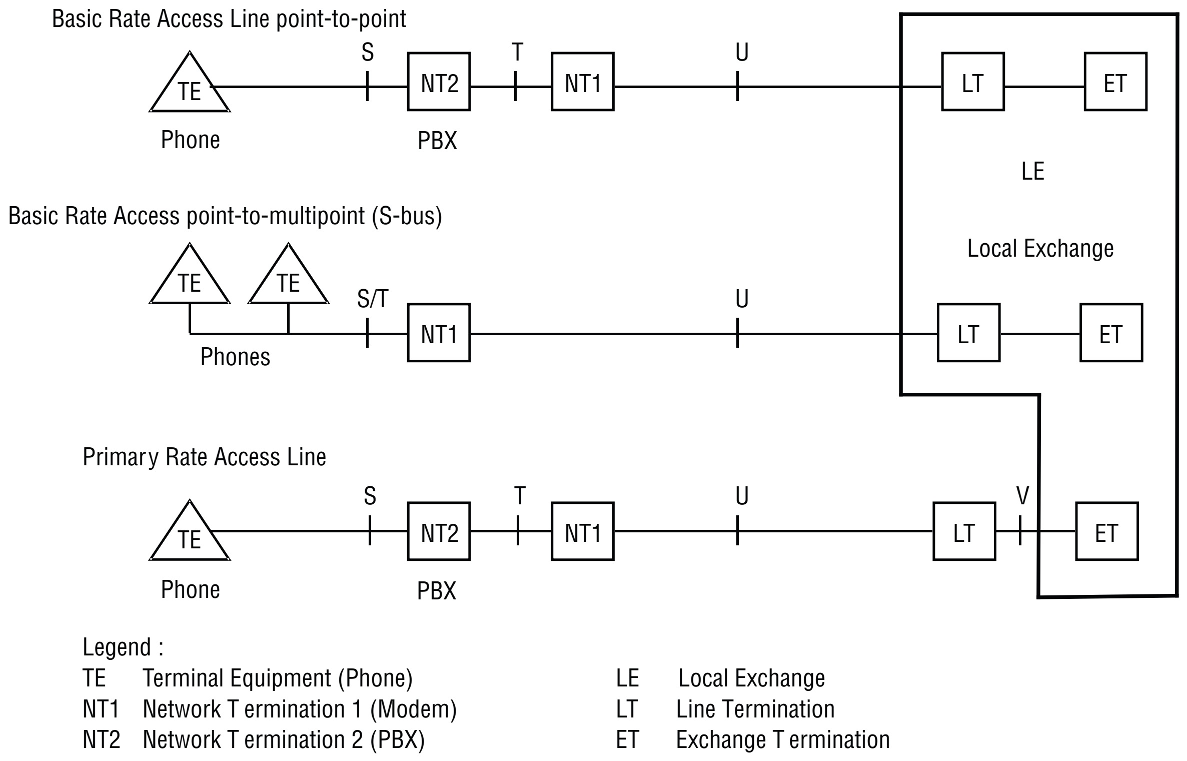

Basic Rate Access Line point-to-point
S
T
U
TE
NT2
NT1
LT
ET
Phone
PBX
LE
Local Exchange
Basic Rate Access point-to-multipoint (S-bus)
TE
TE
S/T
NT1
U
LT
ET
Phones
Primary Rate Access Line
S
T
U
V
TE
NT2
NT1
LT
ET
Phone
PBX
Legend :
TE Terminal Equipment (Phone)
NT1 Network T ermination 1 (Modem)
NT2 Network T ermination 2 (PBX)
LE Local Exchange
LT Line Termination
ET Exchange T ermination

Figure 95. ISDN reference points

The S reference point is on the subscriber interface. This is the typical 4-wire connection between an ISDN phone and an ISDN PBX. Be aware that many ISDN PBX vendors use non-standard proprietary 2-wire interfaces to connect the Terminals to the PBX.

The T reference point is on the trunk interface of a PBX. This is the standard 4-wire interface between the PBX and the network termination unit (NTU) also known as NT1 in standard terminology. The ISDN layer 2 protocol at this point is in point-to-point mode between the NTU and the PBX.

The 4-wire layer 1 specification S and T interfaces is foreseen for in-house installations and carries a maximum of 150 meters.

The S/T reference point is on a point-to-multipoint S-Bus. Here several terminals are connected directly to the same BRI NTU. The S and T reference points are “collapsed”. The NT2 is not represented by any equipment unit.

The U reference point is on the transmission side of the NTU designed to carry the ISDN line over the last mile. For basic rate interfaces this is typically a DSL technology working on legacy copper pairs over a distance up to 12 kilometers. For primary rate lines, DSL, coax and fiber transmission is in use. In most European countries the U interface is not accessible to the subscriber, the operator always provides the NT1. In the US and some other countries the NT1 can be integrated into the NT2, i.e. the PBX is connected directly to the U interface.

The V reference point is typically a y-wire interface between the line card of the public switch and the 2 Mbps transmission equipment which transports the PRI signal over copper (DSL), coax or fiber.

Possible Patton Device Port Configurations

The device’s ISDN ports can be configured for connection to S, T, S/T, and V interfaces. Refer to figure 97, which illustrates some of the possible network integration options.

ISDN UNI Signaling

ISDN is a User-Network Interface (UNI) signaling protocol with a user and a network side. The user side is implemented in ISDN terminals (phones, terminal adapters, etc.) while the network side is implemented in the exchange switches of the network operator. Both sides have different signaling states and messages. Trinity ISDN ports can be configured to work as user (USR) or network (NET) interfaces.

A Patton device in some applications does not replace a standard ISDN equipment (PBX or Terminal) but is inserted between an existing NT and PBX. In such cases the device’s ISDN ports are configured to operate the opposite side of the connected equipment as illustrated in figure 97.

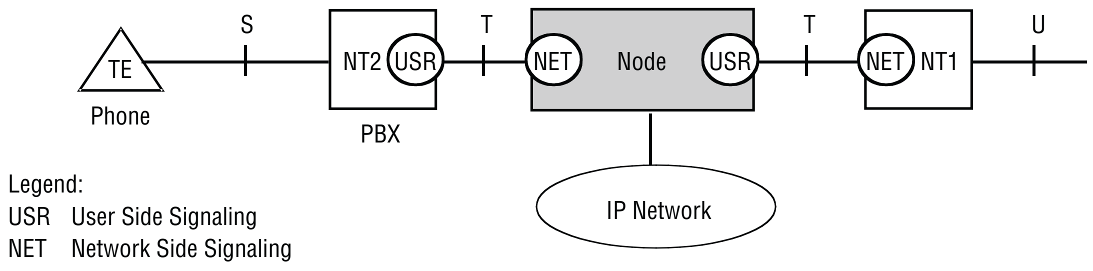


Figure 96. ISDN signaling side

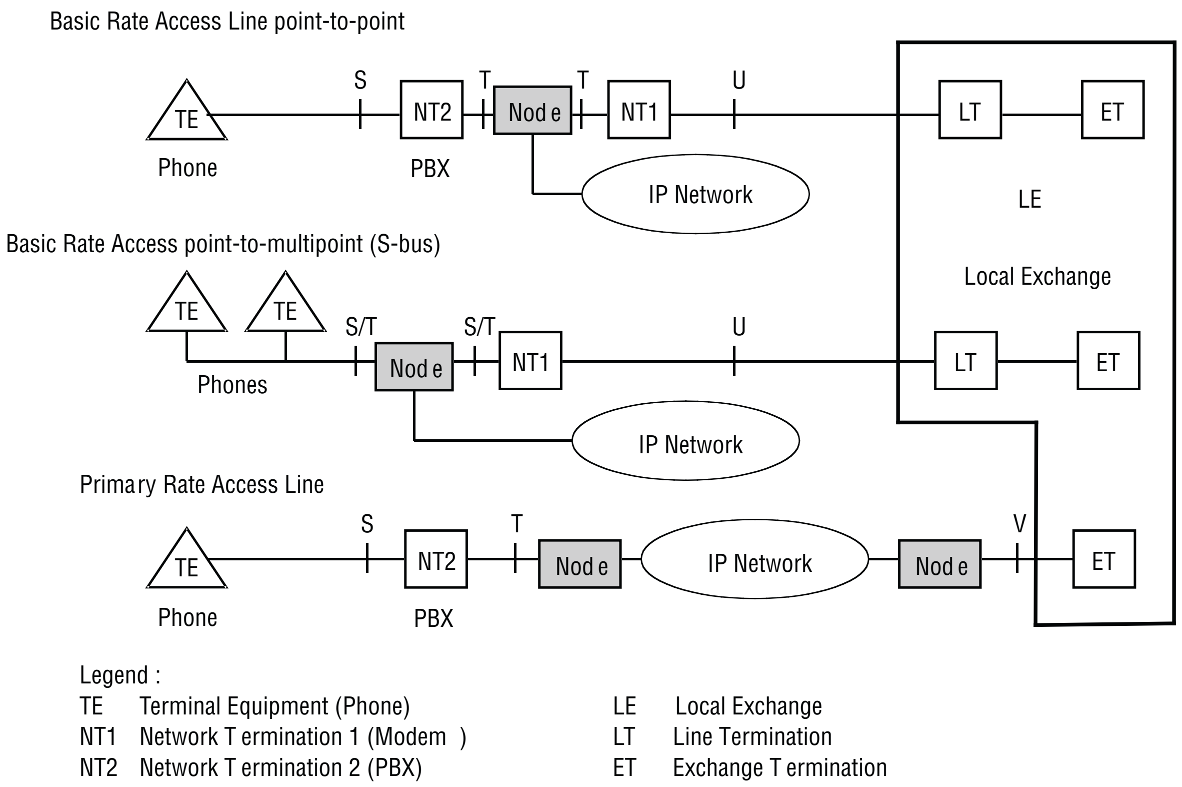


Figure 97. Integration of ISDN access lines

IMPORTANT

Port activation deactivation—ISDN ports can be configured while they are active. However they will be internally disabled to modify the configuration and then re-enabled. All active calls on the port are dropped during this process. Configuration changes should only be performed during planned down times.

IMPORTANT

Reference clock source and synchronization—The Patton device uses a single reference clock source for the synchronization of the 64kbit/s PCM channels on the ISDN ports and in the CS context. This reference clock source can be internal or it can be derived from one of the ISDN ports. If the clock reference is not configured in accordance with the network environment, clock slips and related voice quality degradations can occur. Refer to Chapter 40, "CS Context Overview" on page 394 on how to configure the reference clock

IMPORTANT

Connector pin-out and short circuits—Some of the Patton devices' ISDN BRI ports are configurable to operate as network or terminal ports. The pin-out of the sockets is switched according to this configuration. Wrong port configurations, wrong cabling or wrong connections to neighboring equipment can lead to short circuits in the BRI line powering. Refer to the HW installation guide and the port configuration sections below to avoid misconfiguration.

ISDN Configuration Concept

ISDN Layering

ISDN consists of 3 layers. Each layer has its own parameters that need to be configured.

- Layer 1, often called the physical layer, is responsible to transport single bits between two systems. Layer 1 does not guarantee that a message can be transmitted without errors.

  Parameters: Clock mode, line codes.

- Layer 2 allows a station to reliably send messages to another station using the D channel. Layer 2 implements flow control, error detection and correction (retransmission) as well as addressing mechanism to direct messages to individual devices.

  Parameters: point-to-point or point-to-multipoint mode, network/user side, permanent layer 2 enabled.

- Layer 3 does send and receive application level messages (i.e. call control). It cares for sending broadcast messages and collecting the individual results of the attached devices. It also handles the assignment of the B channels.

  Parameters: network/user side, protocol (i.e. DSS1), maximum number of channels.

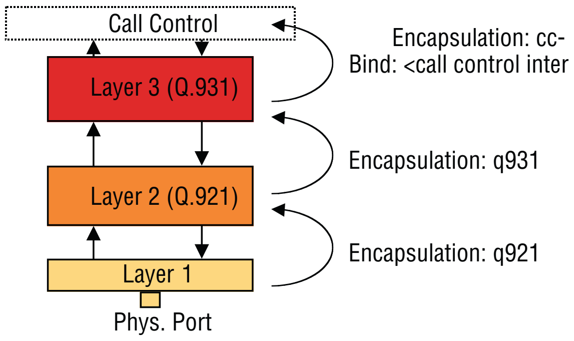


Figure 98. ISDN layering model

The layered model of ISDN is reflected in the configuration by the use of different modes for each layer. The layers are connected by using encapsulations and bindings. The encapsulation defines what the next higher layer protocol will be. On the topmost layer, the binding finally selects a logical interface to connect the port to. For more information how to configure and setup the physical ports for ISDN, please see Chapter 56, "PRI Port Configuration" on page 648. Detailed information about Q.921 and Q.931 configuration are available in Chapter 54, "ISDN Configuration" on page 622.

Chapter 54 ISDN Configuration

Chapter contents

Introduction

This chapter describes the configuration of the Q.921 and Q.931 protocol and how to bind the ISDN protocol to an application like the Call Control. To get an overview of the ISDN protocol and the layered configuration model of Trinity, please see Chapter 53, "ISDN Overview" on page 617. In this chapter it is assumed, that the lower layer on which ISDN will be setup is correctly configured. If ISDN has to run on a TDM port like PRI, please see Chapter 56, "PRI Port Configuration" on page 648.

ISDN Configuration Task List

Configuring ISDN typically consists of the following tasks:

- Enter Q.921 configuration mode
- Configuring Q.921 parameters
- Configuring Q.921 encapsulation
- Enter Q.931 configuration mode
- Configuring Q.931 parameters
- Configuring Q.931 encapsulation

Enter Q.921 Configuration Mode

Normally, Q.921 is running as ISDN Layer 2 protocol on a BRI or PRI port. But it is also possible another protocol is using Q.921 as its next encapsulation step and then Q.921 will not be configured out of a port context. That means, Q.921 encapsulation can be configured in different configuration modes. For this reason, the command description below refers to the configuration mode in which Q.921 can be enabled by setting the encapsulation to 'q921'. This configuration mode is called 'base-mode' but it is only an alias for the real mode. Once the encapsulation of q921 has been configured, the Q.921 configuration mode can be entered.

Mode: base-mode

Step	Command	Purpose
1	**node(base-mode)[slot/port]#[no] encapsulation q921**	Enables/Disables Q.921
2	**node(base-mode)]#q921**	Enter the Q.921 configuration mode

Configuring Q.921 Parameters

This chapter provides an overview of the Q.921 configuration parameters, their syntax and possible restrictions. In case of ISDN, Q.921 settings apply to both BRI and PRI ports. They are defined in the q921 mode. To use Q921, the lower layer encapsulation must be set to q921.

Mode: q921

Step	Command	Purpose
1	***node***(q921)[e1t1-(slot/port)]#protocol pp OR ***node***(q921)[e1t1-(slot/port)]#protocol pmp	Specify Q.921 operating mode (Default: BRI: pmp, PRI: pp). The Q.921 protocol running on BRI ports can operate in point-to-point (pp) or point-to-multipoint (pmp) mode. Point-to-multipoint is used to connect multiple terminals to an ISDN S-Bus. In some cases small PBXs are also connected to the public ISDN in point-to-multipoint mode. Point-to-point is typically used to connect PBXs to a public or private ISDN. The Q.921 protocol of PRI ports always runs in point-to-point (pp) mode.
2	***node***(q921)[e1t1-(slot/port)]#uni-side auto OR ***node***(q921)[e1t1-(slot/port)]#uni-side net OR ***node***(q921)[e1t1-(slot/port)]#uni-side user	Specify the UNI side of the interface (Default: auto) If layer1 clock mode is not defined or set to auto this setting also specifies the clock mode for layer1. NET: clock mode = master USR: clock mode = slave If set to auto the UNI side setting is taken from layer3.
3	***node***(q921)[e1t1-(slot/port)]#[no] permanent-layer2	Enables the Q.921 permanent activity (Default: disabled). By default, the Q.921 protocol is not enabled permanently, i.e. the first call enables it.

Configuring tei Assignment Procedure on ISDN

This command is a configuration option on ISDN net port to support certain PBXs. During the assignment of a terminal endpoint identifier (tei), the user side generates a random value which allows the net side to distinguish multiple connected users. In the strict configuration, this random value is stored to prevent the tei confusion between the users. In the relaxed configuration, it is not stored, which allows a single connected user device to operate without random values.

Mode: port e1t1 / q921

Step	Command	Purpose
1	***node***(q921)#[no] strict-tei-procedure	Specifies if the assignment of tei should be strict or relaxed. Default: strict procedure is enabled.

Configuring Q.931 Encapsulation

This command specifies the next protocol or application that has to be attached to the Q.921 protocol. In case of ISDN this will always be the Q.931 protocol but in a distributed system for example, it could also be a network protocol.

Mode: q921

Step	Command	Purpose
1	***node*(q921)[*slot/port*]#[no] encapsulation q931**	Enables/Disables the next application or protocol. Currently only Q.931 is supported.

Enter Q.931 Configuration Mode

Normally, Q.931 is running as ISDN Layer 3 protocol on Q.921. But it is also possible another protocol is using Q.931 as its next encapsulation step and then Q.931 will not be configured out of the Q.921 context. That means, Q.931 encapsulation can be configured in different configuration modes. For this reason, the command description below refers to the configuration mode in which Q.931 can be enabled by setting the encapsulation to 'q931'. This configuration mode is called here 'base-mode' but it is only an alias for the real mode. Once encapsulation q931 has been configured, the Q.931 configuration mode can be entered.

Mode: base mode

Step	Command	Purpose
1	[*name*]**(base-mode)]#[no] encapsulation q931**	Enables/Disables Q.931
2	[*name*]**(base-mode)]#q931**	Enter the Q.931 configuration mode

Configuring Q.931 Parameters

This section provides an overview of the Q.931 configuration parameters, their syntax and possible restrictions. In case of ISDN, Q.931 settings apply to both BRI and PRI ports. They are defined in the q931 mode. To use Q931, the lower layer encapsulation must be set to q931.

Note QSIG is an ISDN based protocol for signaling between nodes of a Private Integrated Services Network. The formal name of the signaling system by ISO / IEC is PSS1. Both names will co-exist and QSIG will continue to be used as the marketing name.

Mode: q931

Step	Command	Purpose
1	**device(q931)[e1t1-(slot/port)]#protocol dss1** OR **device(q931)[e1t1-(slot/port)]#protocol dms-100** OR **device(q931)[e1t1-(slot/port)]#protocol ni2** OR **device(q931)[e1t1-(slot/port)]#protocol ntt**	Specify the ISDN layer 3 protocol (Default: BRI: dss1, E1: dss1, T1: ni2) The ISDN layer 3 is the network signaling protocol. Trinity ISDN supports: • Euro-ISDN (E-DSS1) • Q.SIG (PSS1) • National ISDN (NI2) • Nippon Telecom NTT for BRI • Nortel Dms-100 for T1 The layer 3 signaling must correspond to the connected ISDN equipment or network.
2	**device(q931)[e1t1-(slot/port)]# uni-side net** OR **device(q931)[e1t1-(slot/port)]# uni-side user**	Specify the UNI side of the interface. If not defined on layer2 (q921 mode) this setting also specifies the UNI side setting for layer2. Make sure that the device connected to a Patton device ISDN port is operating the opposite side of the configured uni-side.
3	**device(q931)[e1t1-(slot/port)]#max-calls <max-calls> [1-32]** OR **device(q931)[e1t1-(slot/port)]#no max-calls**	Limits the total number of concurrent calls on the port. The **no** form of the command restores the default settings. **Note:** If the channel-range and max-calls command are used simultaneously, the lower number of channels is the limiting parameter.

Step	Command	Purpose
4	**device(q931)[e1t1-(slot/port)]#channel-range <low> <high>** OR **device(q931)[e1t1-(slot/port)]#no channel-range**	Specify B-channel range to be used on a PRI port (Default: E1: 0-31, T1: 0-23) Limits the time-slots to be used for calls to the range between *low* and *high*. This is in some cases required for interoperability with ISDN services that impose the same limitations. Call slots outside the defined range are rejected (busy line). If no range is defined (Default) all 30 E1 (23 T1) time-slots are available for use. The **no** form of the command restores the default settings.
5	**device(q931)[e1t1-(slot/port)]# bchan-number- order ascending** OR **device(q931)[e1t1-(slot/port)]# bchan-number-order ascending-cyclic** OR **device(q931)[e1t1-(slot/port)]# bchan-number-order descending** OR **device(q931)[e1t1-(slot/port)]# bchan-number-order descending-cyclic**	Specify B-channel allocation strategy (Default: ascending) The numbering mode defines how the available time slots are filled. The cyclic modes use a “round-robin” implementation. The “ascending” and “descending” modes define whether the time slots are filled at the lowest or highest available slot, i.e. ascending means that the lowest available slot is used first, descending always uses the highest available slot first.

Configuring a Channel Identifier on ISDN PRI

If the sending of a channel identifier in the SETUP message on the ISDN PRI user interface is forced, in some cases, the channel identifier information element is not according to the Standard. This behavior can be explicitly disabled. When the sending of the channel identifier is not forced, it is only sent when it is complete and valid.

Mode: q931

Step	Command	Purpose
1	**[*node*](q931)[X/Y]#[no] force-channel-id-in-setup**	Forces the sending of a channel identifier in SETUP. Default: Enabled

Configuring Q.931 Application Protocol Encapsulation

This command specifies the next protocol or application has to be attached to the Q.931 protocol. In case of ISDN this will always be the CC-ISDN (Call Control) application. For this case also a binding to a pre-created ISDN interface is necessary. For information about creation and configuration of an ISDN interface please see Chapter 55, "ISDN Interface Configuration" on page 631.

Mode: q931

Step	Command	Purpose
1	***node*(q931)[e1t1-(*slot/port)*]#[no] encapsulation cc-isdn**	Enables/Disables the next application or protocol. Currently only CC-ISDN is supported.
2	**node (q931)[e1t1-(slot/port)]# [no] bind interface** <*if-name*>	Bind the Q.931 protocol to an existing call control interface.

Debugging ISDN

The **debug isdn** command may be used for investigating possible Q.921/Q.931 protocol problems or to get call signaling information. The command can be applied to the port on which ISDN is configured and has a further option to switch on or off a specific ISDN layer. Additionally, the '**show port [e1t1** or **bri]**' command can be used to printout information about the current state and statistical information about received and transmitted frames.

Mode: Operator execution

ISDN Command	Purpose
isdn-control	Enables/Disables call-control ISDN control trace
isdn-datapath	Enables/Disables call-control ISDN datapath trace
isdn-signaling	Enables/Disables call-control ISDN signaling trace
isdn-stack-pri-control	Enables/Disables ISDN stack PRI control trace
isdn-stack-pri-l2	Enables/Disables ISDN stack PRI layer 2 trace
isdn-stack-pri-l3	Enables/Disables ISDN stack PRI layer 3 trace
master-server	Enables/Disables Master Server trace
all-isdn	Enables/Disables all ISDN trace

Mode: Operator execution

Debug ISDN Command	Purpose
debug-isdn-control	Enables/Disables call-control ISDN control trace
debug isdn-datapath	Enables/Disables call-control ISDN datapath trace
debug isdn-signaling	Enables/Disables call-control ISDN signaling trace
debug isdn-stack-bri-control	Enables/Disables ISDN stack BRI control trace
debug isdn-stack-bri-l2	Enables/Disables ISDN stack BRI layer 2 trace
debug isdn-stack-bri-l3	Enables/Disables ISDN stack BRI layer 3 trace
debug isdn-stack-pri-control	Enables/Disables ISDN stack PRI control trace
debug isdn-stack-pri-l2	Enables/Disables ISDN stack PRI layer 2 trace
debug isdn-stack-pri-l3	Enables/Disables ISDN stack PRI layer 3 trace

ISDN Configuration Examples

Example: Configuring BRI port as Euro-ISDN interface

The following example shows how to configure port 0/0 as a Euro ISDN interface with user side signaling.

```
172.16.40.71(cfg)#port bri 0 0
172.16.40.71(prt-bri)[0/0]#q921
172.16.40.71(q921)[0/0]#q931
172.16.40.71(q931)[0/0]#uni-side user
172.16.40.71(q931)[0/0]#encapsulation cc-isdn
172.16.40.71(q931)[0/0]#bind interface bri00
172.16.40.71(q931)[0/0]#exit
172.16.40.71(q921)[0/0]#exit
172.16.40.71(prt-bri)[0/0]#no shutdown
```

Example: being clock slave on uni network interface

The following example shows how to configure both ports of a Patton device with network signaling but receive the clock (via port 0) from the peer. The peer must be configured accordingly, i.e. port 0 as USR/clock master and port 1 NET/clock slave.

```
172.16.40.71(cfg)#port bri 0 0
172.16.40.71(prt-bri)[0/0]#clock slave
172.16.40.71(prt-bri)[0/0]#q921
172.16.40.71(q921)[0/0]#q931
172.16.40.71(q931)[0/0]#uni-side net
172.16.40.71(q931)[0/0]#encapsulation cc-isdn
172.16.40.71(q931)[0/0]#bind interface bri00
172.16.40.71(q931)[0/0]#exit
172.16.40.71(q921)[0/0]#exit
172.16.40.71(prt-bri)[0/0]#no shutdown

172.16.40.71(cfg)#port bri 0 1
172.16.40.71(prt-bri)[0/0]#q921
172.16.40.71(q921)[0/0]#q931
172.16.40.71(q931)[0/0]#uni-side net
172.16.40.71(q931)[0/0]#encapsulation cc-isdn
172.16.40.71(q931)[0/0]#bind interface bri01
```

```
    172.16.40.71(q931)[0/0]#exit
    172.16.40.71(q921)[0/0]#exit
    172.16.40.71(prt-bri)[0/0]#no shutdown
```

Example: PRI

Assume the scenario as illustrated in figure 99:

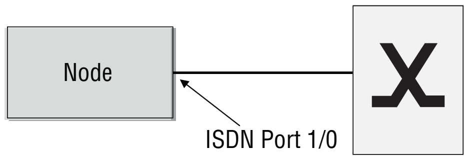


Figure 99. PBX connected to ISDN port 1/0

Configure PRI port 1/0 as clock master. From the Local Exchange timeslots 1 through 20 are available and the total number of concurrent calls shall be limited to 10. Use down-cyclic channel numbering.

```
    172.16.40.71(cfg)#port e1t1 1 0
    172.16.40.71(prt-e1t1)[1/0]#q921
    172.16.40.71(q921)[e1t1-1/0]#q931
    172.16.40.71(q931)[e1t1-1/0]#uni-side net
    172.16.40.71(q931)[e1t1-1/0]#max-channels 10
    172.16.40.71(q931)[e1t1-1/0]#channel-range 1 20
    172.16.40.71(q931)[e1t1-1/0]#bchan-number-order descending-cyclic
    172.16.40.71(q931)[e1t1-1/0]#exit
    172.16.40.71(q921)[e1t1-1/0]#exit
    172.16.40.71(prt-e1)[e1t1-1/0]#no shutdown
```

Chapter 55 ISDN Interface Configuration

Chapter contents

Introduction

This chapter provides an overview of ISDN interfaces, and the tasks involved in their configuration. This chapter does not explain the basic configuration steps equal to all CS interfaces. Information about basic interface configuration can be found in the general chapter about CS interface configuration (see Chapter 41, "CS Interface Configuration" on page 416)

An ISDN interface represents the connection of an ISDN signaling channel to the call control. It encapsulates the ISDN layer 3 protocol of an ISDN port's D-channel, allows incoming and outgoing calls on this port, controls its B-channels and provides a set of services.

There is a one-to-one relation between the port and the interface: Only one port can bind to an existing interface, and there must be a port that binds to the interface for the interface to become functional (see figure 100).

An ISDN interface can encapsulate user and network side of the following protocols: DSS1, NI2, NTT. The settings are automatically taken from the port that binds to the interface, and changes on the port are automatically reflected on the interface.

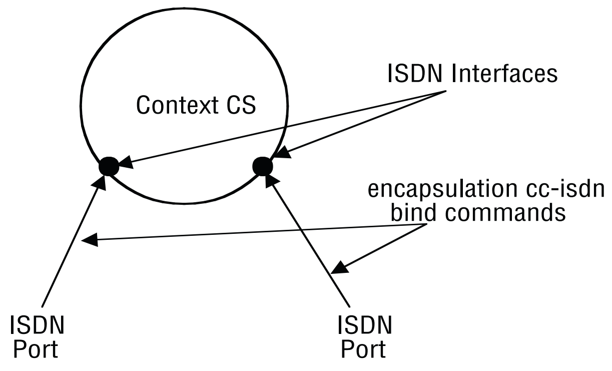


Figure 100. ISDN interfaces on the CS context

ISDN Interface Configuration Task List

This section describes the configuration tasks for ISDN interfaces. There are no mandatory configurations on ISDN interfaces, because all protocol relevant settings are inherited from the port that binds to the interface.

The settings on the interface are those of basic CS interface configuration, as well as settings for interoperability and supplementary services:

- Configuring ringback tone on ISDN user-side interfaces
- Configuring call waiting (optional)
- Disabling call waiting on ISDN DSS1 network interfaces
- Configuring date/time publishing to terminals (optional)
- Enabling sending of date and time on ISDN DSS1 network interfaces
- Defining the 'network-type' in ISDN interfaces
- ISDN explicit call transfer, SIP REFER transmission
- ISDN Advice of Charge support

Configuring DTMF Dialing (optional)

Most ISDN terminals support two modes of call setup: En-bloc dialing and overlap dialing. En-bloc dialing transports the full called party information in the first SETUP message from the terminal. This means that the user must dial the number before going off-hook. Overlap dialing transports the called-party number digit by digit, after the first SETUP message, which contains no called-party information at all. Combinations between en-bloc and overlap dialing are possible.

Most terminals use ISDN *keypad facility* messages to transport digits one-by-one in overlap dialing. But some terminals, especially terminal adapters for analog devices, might transport the digits only using DTMF tones, without associated keypad facility messages.

The **DTMF dialing** command enables the ISDN port for use with the later devices.

Be sure to only use this command when needed. Otherwise, called party information can be corrupted because the digits arrive twice, as keypad facility messages and also as DTMF tones.

Procedure: To enable DTMF dialing

Mode: Interface ISDN

Step	Command	Purpose
1	***node*(if-isdn)**[*if-name*]**#[no] dtmf-dialing**	Enables/Disables DTMF dialing Default: disabled

Example: Enable DTMF dialing

The following example shows how to enable DTMF dialing for a given ISDN interface.

```
node>enable
node#configure
node(cfg)#context cs
node(ctx-cs)[switch]#interface isdn MyIsdnIf
node(if-isdn)[myIsdnIf]#dtmf-dialing
```

Configuring an Alternate PSTN Profile (optional)

The PSTN profile contains the configuration for data/voice transmission on circuit-switched channels (see Chapter 39, "PSTN Profile Configuration" on page 390). In the case of ISDN interfaces, the PSTN profile applies to the ISDN B-Channels associated with the interface.

There is a PSTN profile named *default*, which always exists in the system. If no different PSTN profile name is explicitly configured on the ISDN interface, the profile *default* is used.

Procedure: To define an alternate PSTN profile for the ISDN interface

Mode: Interface ISDN

Step	Command	Purpose
1	***node*(if-isdn)**[*if-name*]**#[no] use profile pstn** *profile-name*	Defines an alternate PSTN profile to be used for this ISDN interface/Reverts the setting to its default (use profile PSTN *default*)

Example: Configure an alternate PSTN profile

The following example shows how to replace the PSTN profile *default* of the ISDN interface with the PSTN profile *myprofile*.

```
node>enable
node#configure
node(cfg)#context cs
node(ctx-cs)[switch]#interface isdn myIsdnIf
node(if-isdn)[myIsdnIf]#use profile pstn myprofile
```

Configuring Ringback Tone on ISDN User-side Interfaces

If a ring-back tone needs to be played towards the PSTN from an ISDN user-side interface, this can be forced using the following command.

Mode: interface ISDN <if-name>

Step	Command	Purpose
1	[*name*] **(pf-isdn)**[*if-name*]**# [no] user-side-ring-back-tone**	Enables ringback tone to be played on ISDN user-side interfaces. Default: *disabled*

Configuring Call Waiting (optional)

The term “call waiting” is used as follows in this context: If the port bound to this interface is configured to be network side, and both ISDN B-Channels are engaged with calls, and there is a new outgoing call over this interface, the interface can:

- Signal the new call to all connected terminals, although both B-Channels are in use. One terminal can then put its current call on hold to accept the new one (putting the call on hold frees its B-Channel).
- Not signal the new call, because there is no B-Channel available. This is the desired behavior particularly if the bound port is part of a hunt-group, and no user terminals are connected.

Default behavior is a), using the command below in the inverted form, behavior b) is selected.

Procedure: To configure call waiting

Mode: Interface ISDN

Step	Command	Purpose
1	*node***(if-isdn)**[*if-name*]**#[no] call-waiting**	Enable/disable call waiting feature as described above Default: enabled

Disabling Call Waiting on ISDN DSS1 Network Interfaces

This procedure disables support for call waiting on an ISDN DSS1 network interface.

Mode: interface isdn <if-name>

Step	Command	Purpose
1	[*name*] **(if-isdn)**[*if-name*]**# no call-waiting**	Disable call waiting.

Configuring Call-Hold on ISDN Interfaces

Normally, the call-hold feature is disabled on ISDN point-to-point links and enabled on ISDN point-to-multipoint links. However, you can manually enable or disable the Call-Hold feature using the following command: The default setting can be achieved using the 'auto' configuration option.

Mode: interface isdn

Step	Command	Purpose
1	***node*(if-isdn)**[if-*name*]**# call-hold {auto|enable|disable}**	Enable or disable the call-hold functionality for an isdn interface. If 'auto' is selected, call-hold is automatically disabled on p2p links and enabled on p2mp links. Default: auto

Enabling Display Information Elements on ISDN Ports

By default no display information elements are sent in ISDN signaling messages. You can enable sending of ISDN Display Information elements in ISDN signaling messages using the following command.

Mode: interface isdn

Step	Command	Purpose
1	***node*(if-isdn)**[if-*name*]**# [no] display emit**	Enable sending of the display information element for an isdn interface. Default: disabled

Configurable calling party or facility IE on ISDN

This function defines if called-party numbers starting with „# or „* characters are sent in a keypad-facility info-element or in a calling-party info-element. When the *Keypad-facility* command is inverted, the called-party number is placed in a calling-party info-element.

Mode: context cs switch

Step	Command	Purpose
1	(if-isdn)[if-name]# [no] keypad-facility	Default: enabled

Configuring Date/Time Publishing to Terminals (optional)

ISDN allows to propagate current time and date information from a port configured as network to the connected terminals. You can configure each ISDN interface to propagate the current device system time and date to the connected terminals with the following command:

Procedure: To configure date and time publishing

Mode: Interface ISDN

Step	Command	Purpose
1	***node*(if-isdn)**[*if-name*]**#[no] isdn-date-time**	Enable/disable publishing of system time to connected ISDN terminals. Default: disabled

Date and time information can only be contained in the ISDN CONNECT message. This message is only delivered to a terminal when a call from the terminal to the device is made and reaches connected state.

Sending the Connected Party Number (COLP) (optional)

The connected party number information element is normally only inserted into the CONNECT message when it has been verified from the remote side. For certain configurations that depend on the presence of such an information element, it is possible to configure so that the connected party number information element is always inserted.

There are three different configurations for the connected party number information element:

- **disabled:** Never insert the connected party number
- **verified:** Insert the connected party number only if it has been verified. This is the default behavior
- **always:** Insert the connected party number always. If the connected party number is not verified, the called party number is taken as connected party number

Mode: interface ISDN

Step	Command	Purpose
1	***node*(if-isdn)**[*if-name*]**#[no] send-connected-party-number** [always | verified]	Configures the sending of the connected-party-number. Default: Verified

Enabling Sending of Progress-indicator on ISDN (optional)

In some setup configurations, it is not desired to send a progress-indicator on in the ISDN signaling. Therefore, it is possible to disable the sending of the progress-indicator.

Mode: interface ISDN <if-name>

Step	Command	Purpose
1	[*name*]**(if-isdn)**[*if-name*]**# [no] send-progress-indicator**	Enable/Disable the sending of the information Element progress indicator. Default: enabled

Defining the 'network-type' in ISDN Interfaces

The following command defines the location code to be inserted in ISDN causes code information elements.

Mode: interface ISDN <if-isdn>

Step	Command	Purpose
1	[*name*] **(if-isdn)**[*if-name*]**# network-type** [international|private|public|transit|user|beyond}	Defines the type of network to which the system belongs.

ISDN Explicit Call Transfer Support (& SIP REFER Transmission)

Additional call transfer support is enabled by default for ISDN interfaces (BRI ports) by accepting or rejecting explicit call-transfer (ECT) invocations. An ISDN phone that is connected to a BRI port and that has two active calls can send an ECT invocation to connect the two calls inside the device. An ISDN interface can be configured to accept or reject ECT invocations.

Trinity detects calls that are looped internally, i.e. calls that leave the device over the same ISDN interface over which they enter the device. If an internal loop is detected for an ISDN interface bound by an ISDN user port, Trinity sends an explicit call-transfer (ECT) to push back the call to the connected network as soon as the call is connected. An ISDN interface can be configured to emit ECT invocations.

SIP interfaces react similarly to internally looped calls. If a call leaves the device over the same SIP gateway over which it entered the device, Trinity sends a REFER message to one of the remote user agents to transfer the call to the two parties. A SIP interface can be configured to emit REFER messages.

Figure 101 shows an example scenario where a SIP network connects two devices to give a home office (HO) access to a PBX in the central office (CO).

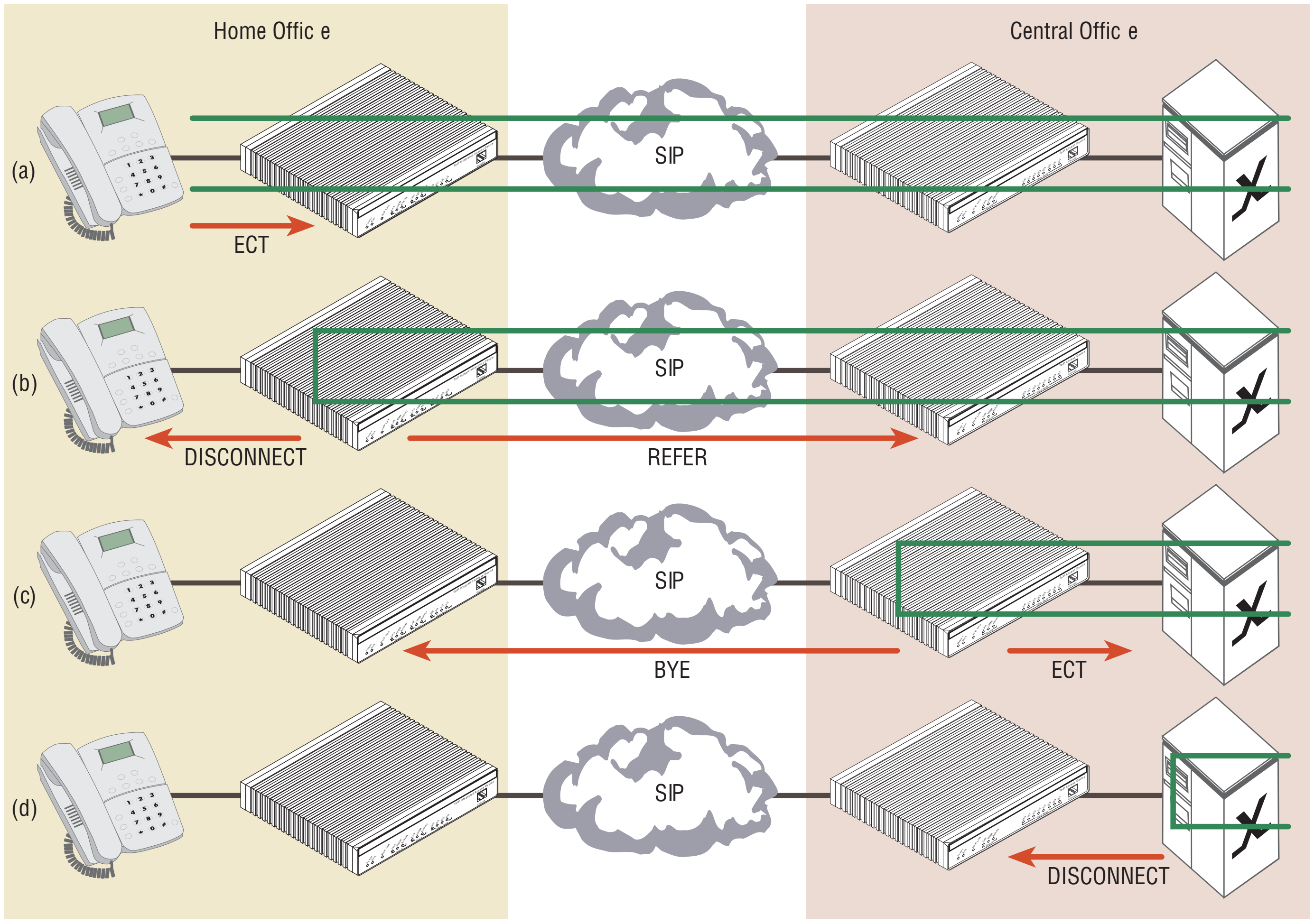


Figure 101. Example SIP network connecting two device to give a home office access to the CO PBX

The phone in the home office has two active calls to other subscribers of the PBX in the central office. The user wants to connect the other two participants and **(a)** sends an explicit call-transfer invocation to the device HO. The device HO internally connects the two calls and sends a DISCONNECT message to the phone for both calls. In a second step **(b)** the firmware on HO detects an internal loop. Both call legs are connected to the same network. In this example, both call legs are handled by the same SIP gateway. The firmware on device HO sends a REFER message to device CO, which connects the two call legs internally and sends a BYE message to the device HO. **(c)** Again the firmware of CO detect an internal loop. This time the call legs are handled by the same SIP interface, connected to the PBX. Since the ISDN port is a user port it sends an explicit call-transfer invocation to the PBX **(d)**, which connects the call and sends the device CO a DISCONNECT message for both calls. During all these push back operations the datapath of the two participants keeps connected.

The push back mechanism over ISDN (using ECT) and SIP (using REFER) works independently of the protocol that invoked the call-transfer.

The push-back mechanism can be configured on each interface separately. Per default push-back is enabled for ISDN and SIP interfaces. You only have to change the configuration if you don't want internally looped calls to be pushed back to the network. The configuration command **[no] call-transfer accept** configures if an incoming call-transfer request (e.g. ECT or REFER) shall be accepted. The configuration command **[no] call-transfer emit** configures if a call reaching the device over this interface and leaving the device over this interface shall be pushed back to the network, i.e. if a call-transfer request (ECT or REFER) shall be sent.

The following procedure disables the push-back mechanism on the ISDN interface connected to the PBX. No ECT invocation is sent when a call is detected that is looped internally.

Step	Command	Purpose
1	***node*(ctx-ip)**[ctx-*name*]**# interface isdn** <if-name>	Go to the ISDN interface, for which you want to disable the push back mechanism.
2	***node*(if-isdn)**[if-*name*]**# no call-transfer emit**	Disable the push back mechanism

The following procedure disables the push-back mechanism on a SIP interface. No REFER message is sent when a call is detected that is looped internally.

Step	Command	Purpose
1	***node*(ctx-ip)[ctx-*name*]# interface sip** <if-name>	Go to the SIP interface, for which you want to disable the push back mechanism.
2	***node*(if-sip)**[if-*name*]**# no call-transfer emit**	Disable the push back mechanism

ISDN Advice of Charge Support

The exchange of “Advice of Charge” information is supported between two ISDN interfaces. The charge information can be transmitted and received over SIP. Without configuration changes Trinity tunnels the “Advice of Charge” information from an ISDN user interface to an ISDN network interface. However, you can disable AOC-S, AOC-D or AOC-E separately on each interface.

The network sends tariff information about a call using AOC-S messages at call (S)etup time and during the call when the tariff changes. Then (D)uring the call, the network sends the current charge in AOC-D messages. Finally at the (E)nd of the call, the network sends the total charge in an AOC-E message encapsulated in the DISCONNECT or RELEASE message.

The following procedure disables the reception of AOC messages from the network on an ISDN user interface.

Step	Command	Purpose
1	***node*(ctx-ip)[ctx-*name*]# interface isdn** <if-name>	Go to the ISDN interface, for which you want to disable AOC
2	***node*(if-isdn)**[if-*name*]**# no aoc-s**	Disables the reception of AOC-S messages at call setup time
3	***node*(if-isdn)**[if-*name*]**# no aoc-d**	Disables the reception of AOC-D messages during the call
4	***node*(if-isdn)**[if-*name*]**# no aoc-e**	Disables the reception of AOC-E messages at the end of the call

AOC is a network option that can be disable or set to be active for all calls or only on a per-call basis. If your network provider offers AOC on a per-call basis the firmware needs to request AOC information on each outgoing call. The following procedure enables the reception of AOC messages on an ISDN user interface. Additionally the interface sends an AOC activation request for each outgoing call to the network.

Step	Command	Purpose
1	***node*(ctx-ip)**[ctx-*name*]**# interface isdn** <if-name>	Go to the ISDN interface, for which you want to enable AOC on a per-call basis
2	***node*(if-isdn)**[if-*name*]**# aoc-s explicit**	Enables the reception of AOC-S messages and sends an AOC-S activation request for each outgoing call
3	***node*(if-isdn)**[if-*name*]**# aoc-d explicit**	Enables the reception of AOC-D messages and sends an AOC-D activation request for each outgoing call
4	***node*(if-isdn)**[if-*name*]**# aoc-e explicit**	Enables the reception of AOC-E messages and sends an AOC-E activation request for each outgoing call

In default, an ISDN network interface provides AOC information to the connected phones only if available, i.e. only if the call is routed to an ISDN user interface that is connected to a network providing AOC information. The following procedure enables the transmission of AOC message on an ISDN network interface even if there is no AOC information from the network. In that case a message containing the value *noChargeAvailable* is sent.

Step	Command	Purpose
1	***node*(ctx-ip)**[ctx-*name*]**# interface isdn** <if-name>	Go to the ISDN network interface, for which you want to enable AOC for all calls
2	***node*(if-isdn)**[if-*name*]**# aoc-s automatic**	Enables the transmission of AOC-S messages even if there is no tariff information from the network for all calls
3	***node*(if-isdn)**[if-*name*]**# aoc-d automatic**	Enables the transmission of AOC-D messages even if there is not charge information from the network for all calls
4	***node*(if-isdn)**[if-*name*]**# aoc-e automatic**	Enables the transmission of AOC-E message even if there is no charge information from the network for all calls

The following procedure enables the transmission of AOC message on a per-call basis. That is AOC messages are sent by the connected phone only if configured for a per-call basis.

Step	Command	Purpose
1	***node*(ctx-ip)**[ctx-*name*]**# interface isdn** <if-name>	Go to the ISDN network interface, for which you want to enable AOC on a per-call basis
2	***node*(if-isdn)**[if-*name*]**# aoc-s explicit**	Enables the transmission of AOC-S messages even if there is no tariff information from the network on a per-call basis
3	***node*(if-isdn)**[if-*name*]**# aoc-d explicit**	Enables the transmission of AOC-D messages even if there is not charge information from the network on a per-call basis
4	***node*(if-isdn)**[if-*name*]**# aoc-e explicit**	Enables the transmission of AOC-E message even if there is no charge information from the network on a per-call basis

The following table shows an overview of the AOC variants:

	no aoc-x	**aoc-x transparent**	**aoc-x automatic**	**aoc-x explicit**
Default option	no	yes	no	no
ISDN User Interface (connected to a PBX switch etc.)				
No message from the network	No information forwarded to the peer interface	No information forwarded to the peer interface	No information forwarded to the peer interface	Sends an aoc-*x* request to the network. If the network rejects the request, no information is forwarded to the peer interface
AOC message from the network	No information forwarded to the peer interface	Information forwarded to the peer interface	Information forwarded to the peer interface	Information forwarded to the peer interface
ISDN Network Interface (connected to phones)				
Phone does not request AOC on a per-call basis	No information sent	Information sent as received from the network, no information sent if the network does not provide information	Always send information, *noChargeAvailable* sent if the network does not provide information	No information sent
Phone requests AOC on a per-call basis	No information sent	Information sent as received from the network, no information sent if the network does not provide information	Always send information, *noChargeAvailable* sent if the network does not provide information	Always send information, *noChargeAvailable* sent if the network does not provide information

ISDN DivertingLegInformation2 Facility

Trinity is now able to extract the redirecting information from the DivertingLegInformation2 Facility and to provide them to the call control. In the other direction, the redirecting information can be sent as DivertingLegInformation2 Facility in addition to the Redirecting Number Information Element.

Transmit direction

Mode: interface isdn <interface>

Step	Command	Purpose
1	[*name*] **(if-isdn)**[*interface*]**#[no] diversion emit**	Enables or disables transmitting of the DivertingLegInformation2 Facility.

Receive direction

Mode: interface isdn <interface>

Step	Command	Purpose
1	[*name*] **(if-isdn)**[*interface*]**#[no] diversion accept**	Enables or disables receiving of the Diverting-LegInformation2 Facility.

T1 Caller-Name Support

The ISDN implementation now supports reception and transmission of the caller-name on T1 links as it is used in NI2 networks according to Bellcore GR-1367-CORE. Transmission of the caller-name is part of the Calling Name Delivery (CNAM) service.

In previous build series (R3.20), the caller-name was already supported for DSS-1 networks using User-User information elements and for Q.SIG (PSS-1) networks using FACILITY messages. Now the caller-name is also supported for NI2 networks following the Bellcore standard.

As a prerequisite, the *caller-name* feature must be enabled on each ISDN interface in the CS context separately. This command now has additional arguments to configure the SETUP retention as follows:

In NI2 networks an incoming ISDN SETUP message may contain a *NameInfomationFollowing* indication instead of the name. This means that the calling-party name is not available yet, but will be sent later, for example, after the dictionary database lookup in progress succeeded. If such an incoming ISDN call is internally routed to another network (e.g. to a SIP network or to a ISDN DSS-1 network), we must know the name before sending the initial INVITE or SETUP message towards the destination network. Therefore we must retain the SETUP message of the incoming ISDN call until the name is present. The caller-name command now allows you to configure the behavior of this SETUP retention mechanism. There are three possible options:

- **caller-name ignore-absence <timeout>:** This configuration command specifies the behavior for incoming ISDN calls. When a *NameInformationFollowing* indication is received with the SETUP message, the call-initiation is retained until the name is received or until this timeout elapses. After that, the call is forwarded to the configured destination interface. When forwarding a call without a caller-name to a SIP network, please note that there is no chance to send the caller-name later over SIP.
- **caller-name early-alerting <timeout>**: This configuration command specifies the behavior for incoming ISDN calls. Some networks only deliver the name after an alerting indication. These networks simulate the mid-ring name delivery feature of analog lines. If early alerting is enabled, we send back a faked ALERTING message after a configurable timeout when we receive a *NameInformationFollowing* indication. This command can be used together with the ignore-absence command. For example, you can configure an interface to first generate an ALERTING message and later forward the call anyway. If used that way, the early-alerting timeout should be smaller than the ignore-absence timeout.
- **caller-name send-information-following**: This configuration command specifies the behavior for outgoing ISDN calls. If there is no name from the originating network, the ISDN interface configured with this command sends a *NameInformationFollowing* indication to the remote side itself.

The following example enables and configures the caller-name feature on a T1 ISDN interface for incoming calls. If no name is present in the SETUP message, but the SETUP message contains the *NameInformationFollowing* indication, an ALERTING message is sent back after 500ms. If there is no name after additional 500ms the call is routed to the destination network anyway.

Mode: context cs / interface isdn

Step	Command	Purpose
1	***node*(if-isdn)#caller-name**	Enables reception of the caller-name.
2	***node*(if-isdn)#caller-name early-alerting 500**	(optional) If no name is present in an incoming ISDN call and if the incoming SETUP message contains the NameInformationFollowing indication, we send a fake ALERTING message after 500ms towards the caller. The SETUP message is retained for this period, i.e. the call is not forwarded to the configured destination. This step is optional. When not configured, an ALERTING message is faked after 2s by default. You can disable faking an ALERTING message by using the “no” form of the command. Note: If the ignore-absence timeout is also configured, the early-alerting timeout should have a smaller value than the ignore-absence timeout.
3	***node*(if-isdn)#caller-name ignore-absence 1000**	(optional) If no name is present in an incoming ISDN call and if the incoming SETUP message contains the NameInformationFollowing indication, we forward the call to the routing destination anyway after 1000ms (500ms after faking the ALERTING message in this example). This step is optional. When not configured, the call is forwarded after 4s by default. You can disable forwarding a call without a name by using the “no” form of the command. Note: The specified timeout is measured starting at the reception of the SETUP message, not when the early-alerting timeout elapses.

The following example enables and configures the caller-name feature on a T1 ISDN interface for outgoing calls. It enables the transmission of the NameInformationFollowing indication (encapsulated into sent SETUP message) when no name is present from the originating network:

Mode: context cs / interface isdn

Step	Command	Purpose
1	***node*(if-isdn)#caller-name**	Enables transmission of the caller-name.
2	***node*(if-isdn)#caller-name send-information-following**	If no name has been received from the originating network a NameInformationFollowing indication is send encapsulated into the SETUP message for the outgoing ISDN call. This feature is disabled by default.

Caller-Name Facility Format on QSIG

The format of the calling name facility on QSIG can now be configured to achieve interoperability with the Siemens Hipath PBX. The default behavior is a local invoke identifier and the name argument is simple. To be compatible with Siemens Hipath, it can be changed to send a global invoke identifier and the name argument extended.

Mode: interface ISDN

Step	Command	Purpose
1	***node*(if-isdn)**[name]**#caller-name presentation** [(simple | extended)]	Sets the format for the caller name facility in QSIG. Default = *Simple*

Format Examples

An example of the simple format:

```
Facility : Invoke : invokeid: 00000001
global operation
invoke {
  present = 1
  local = 0
  argumentnamePresentationAllowedSimple = 'UserA'
}
```

An example of the extended format:

```
Facility : Invoke : invoked: 00000001
global operation
invoke {
  present = 1
  global = 1.3.12.9.0
  argumentnamePresentationAllowedExtended {
    nameData = 'UserA'
    characterSet = 1
  }
}
```

Tunneling of ISDN UUI1 Information Over SIP

Patton Devices have the capability to translate User-to-User information from ISDN networks to SIP and vice-versa. This applies only for UUI1 with implicit activation. UUI1 with explicit activation, UUI2 and UUI3 are not supported. On ISDN the information is transported in a UserUser information element in the messages SETUP, ALERTING, CONNECT, DISCONNECT and RELEASE. The RELEASE message is used as transport only if it is the message which initiates the termination of the call.

The protocol on the SIP side uses a User-to-User header according to "draft-ietf-cuss-sip-uui-isdn-11 - Interworking ISDN Call Control User Information with SIP". A User-to-User header can be inserted in the initial INVITE, answers to INVITE, BYE and answers to BYE messages. In the case of interworking from ISDN to SIP the main part of the User-to-User header contains the protocol discriminator and the data from the ISDN UserUser information element encoded as base16. The protocol discriminator is not considered by the Device, it is transported transparently over SIP. The following header parameters with the exact values are appended to the header:

- encoding=hex

- purpose=isdn-uui
- content=isdn-uui

An example of such a User-to-User header when tunneling the IA5 character string "Hubert34" would look like:

```
User-to-User: 044875626572743334;encoding=hex;purpose=isdn-uui;content=isdn-uui
```

In the case of interworking from SIP to SIP the exact content and parameters of the incoming User-to-User header are forwarded.

Mode: interface isdn

Step	Command	Purpose
1	device(if-isdn)[<name>]# [no] user-to-user-information emit	Enables/Disables sending of UUI1 information over ISDN. Default is disabled.

Mode: interface sip

Step	Command	Purpose
1	device(if-sip)[<name>]# [no] user-to-user-information emit	Enables/Disables sending of UUI1 information over SIP. Default is disabled.

Configuring the Message Waiting Indication Feature for ISDN

Note Message Waiting Indication is programmed in two sections of Trinity, ISDN interface, and the SIP Location service. The information below refers to information for configuring the Message Waiting Indication feature for ISDN. For information on configuring the Message Waiting Indication feature for SIP, see “Configuring the Message Waiting Indication Feature for SIP” on page 450.

The ISDN interface configuration mode enables notifications about new voice messages with a stuttered dial tone. With the stuttered dial tone, if there is a new message/messages in the voice mailbox, for 3 seconds after taking the phone off-hook the dial-tone will “stutter”, producing short delays between the dial-tone cadence.

You can enable the stutter dial tone feature with the **message-waiting-indication** command.

Step	Command	Purpose
1	**[*node*](if-isdn)[name]#[no] message-waiting-indication stutter-dial-tone**	Enables/Disables Message Waiting Indication through Stuttered Dial Tone

Configuring the Release Tone on ISDN DSS1 Network Interfaces

The following command enables/disables a release tone when sending a DISCONNECT message on the ISDN DSS1 network interface.

Mode: interface isdn

Step	Command	Purpose
1	**[*node*](if-isdn)#[no] release-tone**	Specifies if a release tone on disconnect should be played. Default: enabled

Configuring the ISDN User-side Timer T304

The timer T304 on the ISDN user-side is optional and can be disabled. The timer T304 on ISDN user-side is started when receiving a SETUP ACKNOWLEDGE message and entering the overlap sending state. Timer T304 is reset after each additional digit send and stopped when overlap sending is terminated by receiving a CALL PROCEEDING, ALERTING or CONNECT message. Timer T304 starts with 30 seconds and expiring causes a DISCONNECT to terminate the call.

Mode: cfg

Step	Command	Purpose
1	**[*node*](cfg)[0/0]#[no] isdn t304-user**	Configures the ISDN user-side timer. Default: enabled

Chapter 56 PRI Port Configuration

Chapter contents

Introduction

This chapter provides an overview of the PRI (Primary Rate Interface) ports, their characteristics and the tasks involved in the configuration. The Patton devices know two different kinds of PRI ports, E1 and T1. This chapter describes the superset of all commands available on the different PRI ports. If a command is only executable for a specific port then this circumstance will be noted or highlighted in the command description. This chapter also explains how to prepare the ports for the usage of the different application protocols like ISDN or PPP. For some applications, there must be the possibility to access user defined sets of timeslots of an E1 or T1 port. On Patton devices' this feature is called a Channel Group and it will be described in this chapter as well.

Terminology

Hardware Type: Dependent on the device it can either be E1, T1 or E1T1. The Hardware Type and its belonging Slot and Port Number must be specified for entering the configuration mode of a port. It is not possible to change the Hardware Type, it is given by the system (i.e. BRI/PRI).

Port Type: This expression is used in relation with the E1T1 port and describes if the E1T1 port is currently running in E1 or in T1 mode. On an E1 or T1 port, the Port Type cannot be changed, it is static and matches the Hardware Type.

PRI Port Configuration task List

This section describes the configuration tasks for the PRI port.

- Enable/Disable PRI port (see page 649)
- Configuring the PRI port type (E1T1 only) (see page 650)
- Configuring PRI clock mode (see page 650)
- Configuring PRI line code (see page 650)
- Configuring PRI framing (E1T1 only) (see page 651)
- Configuring PRI line build out (E1T1 in T1 mode only) (see page 651)
- Configuring PRI application mode (E1T1 only) (see page 651)
- Configuring PRI LOS threshold (E1T1 only) (see page 652)
- Configuring PRI encapsulation (see page 652)
- PRI Debugging (see page 652)

Enable/Disable PRI Port

By default, the PRI port is disabled. The following command is used for enabling or disabling it.

Mode: port <hw-type> <slot> <port>

Step	Command	Purpose
1	[*name*] **(prt-e1t1)**[slot/port]**# [no] shutdown**	Enable/Disable the PRI port. Default: *shutdown* (which is disabled)

Configuring PRI Port-type

An E1T1 Port can either work in T1 or in E1 (G.704) mode. This mode can be changed dynamically as long as no encapsulation or encapsulation "ppp" is set. Be aware that changing the port-type also resets the framing and linecode parameters to the default values of the new port-type. If port-type change is not allowed due to current configuration, an error message will be issued.

Mode: port e1t1 <slot> <port>

Step	Command	Purpose
1	[*name*] **(prt-e1t1)**[slot/port]**# port-type {e1 | t1}**	Changes operation mode of the port. Restriction: *Only available for e1t1 ports* Default: *e1*

Configuring PRI Clock-mode

The PRI Port can either work in clock-master or in clock-slave mode. This setting defines the clock dependency of the internal data processing. In clock-master mode the internal data processing is running on an independent clock source. In clock-slave mode the clock source for internal data processing is recovered from the receive line interface. Be aware that always a port-pair of clock-master and clock-slave are connected together. In the other case the data transmission will fail due to bit failures. This command also has the option 'auto' that can be used if the application running on the port is also of an asymmetric nature like master/slave, server/client or user/net. Normally, the option 'auto' is used if the port is setup for ISDN. In this case, the clock mode will automatically be derived from the Q.921 layer protocol. If the UNI-Side (User-Network Interface) of Q.921 is set to 'net', then the clock mode of the port is automatically set to 'master' and if Q.921 is configured as 'user' it will be set to 'slave'.

Mode: port <hw-type> <slot> <port>

Step	Command	Purpose
1	[*name*] **(prt-e1t1)**[slot/port]**# clock {auto | master | slave}**	Configures the clock-mode of the port. Default: *master*

Configuring PRI Line-code

Two different line codes can be selected on the PRI port whereas only 'ami' is standardized for E1 and T1. If the port is running in E1 mode, 'hdb3' is also configurable and in T1 mode 'b8zs'. If a linecode will be selected that is not standardized for the current port mode, an error message will be advised.

Mode: port <hw-type> <slot> <port>

Step	Command	Purpose
1	[*name*] **(prt-e1t1)**[slot/port]**# linecode {ami | b8zs | hdb3}**	Configures the line-code of the port. Default for e1: *hdb3* Default for t1: *b8zs*

Configuring PRI Framing

Three framing formats are available for selection on the E1T1 port.

In structured mode, E1 can be configured for *crc4* or *non-crc4* and T1 has the framing option *esf* and *sf*.

- CRC4 (E1): Cyclic Redundancy Check 4. A CRC4 Multi-Frame consists of 16 continuous Basic-Frames. Each Multi-Frame can be divided into two Sub Multi-Frames. The first bit of Timeslot 0 of each even Sub Multi-Frame is called the C-Bit and belongs to the CRC4 check sum.
- ESF (T1): Extended Super Frame. The ESF if made up of 24 Basic-Frames. Each Basic-Frame includes one overhead bit, the F-Bit. The 24 F-Bits of one Extended Super Frame are used for synchronization (6 Bit), transmitting data link information (12 Bit) and for CRC6 calculation (6 Bit).
- SF (T1): Super Frame: The SF is made up of 12 Basic-Frames. Each Basic-Frame includes one overhead bit, the F-Bit. The 12 F-Bits of one Super-Frame represent the frame alignment pattern that is used for synchronization.

Mode: port <hw-type> <slot> <port>

Step	Command	Purpose
1	[*name]* **(prt-e1t1)**[slot/port]**# framing {crc4 | non-crc4 | esf | sf}**	Configures the framing of the port. Restriction: *Only available for e1t1 ports* E1 mode formats are: *crc4*, *non*-crc4, *unframed*. T1 mode formats are: *esf or sf*. Default for e1: *crc4* Default for t1: *esf*

Configuring PRI Line-Build-Out (E1T1 in T1 mode only)

The line build out configuration is used in long haul applications to prevent cross talk in the far end device.

Mode: port <hw-type> <slot> <port>

Step	Command	Purpose
1	[*name*] **(prt-e1t1)**[slot/port]**# line-build-out {0-133 | 133-266 | 266-399 | 399-533 | 533-655}**	Specifies the length attenuation in feet on the line interface. Restriction: *Only available for e1t1 ports in T1 mode.* Default for t1: *0-133 ft*

Configuring PRI Application Mode (E1T1 only)

The PRI port can be configured to work in either short-haul or in long-haul mode. **Short-haul** is the default application and should be used for transmission distances up to **180m/600ft**. For transmission distances up to **1800m/6000ft**, select the **long-haul** application.

Mode: port <hw-type> <slot> <port>

Step	Command	Purpose
1	[*name*] **(prt-e1t1)**[*slot/port*]**#application {long-haul | short-haul}**	Specifies the e1/t1 application mode Restriction: *Only available for e1t1 ports* Default: *short-haul*

Configuring PRI LOS Threshold (E1T1 only)

This command takes effect only if the PRI port is configured for **long-haul** applications. It specifies the sensitivity for **Loss Of Signal threshold**. A signal suffers more attenuation over long distances than over short distances. Therefore the LOS-Threshold must be set higher for longer transmission distances. This command has a default value of -46dB which should be enough for distances up to 1600 m/5250 ft.

Mode: port <hw-type> <slot> <port>

Step	Command	Purpose
1	[*name*] **(prt-e1t1)[***slot/port***]#los-threshold {-4dB | -6dB | -8dB … -46dB | -48dB}**	Specifies Loss Of Signal Threshold Restriction: *Only available for e1t1 ports* Default: *-46dB*

Configuring PRI Encapsulation

The PRI encapsulation command prepares the port for a specific application protocol.

The q921 encapsulation type automatically binds the signaling timeslot (D-channel) of the selected port to the ISDN Layer 2 protocol. This is timeslot 16 for an E1 and timeslot 24 for a T1 port. If in the q921 configuration mode q931 is specified as next encapsulation, the control of all remaining timeslots (B-channels) is given to the ISDN Layer 3 protocol. For more information please see Chapter 53, "ISDN Overview" on page 617 and Chapter *54,* "ISDN Configuration" on page 622.

Mode: port <hw-type> <slot> <port>

Step	Command	Purpose
1	[*name*] **(prt-e1t1)**[slot/port]**#[no] encapsulation {q921}**	Specifies the encapsulation type of the PRI port. Default: *no encapsulation*

PRI Debugging

For the investigation of possible problems in link establishment, data transmission or synchronization, there exists a debug command with the options 'event' and 'error'. The command has a hierarchical characteristic and can be applied to all ports of given type on the whole device, or to all ports of slot or just to one specific port.

Mode: Operator execution

Step	Command	Purpose
1	[*name*]**#[no] debug *hw-type* [([<slot> | [<port>]])]**	Enables/Disables the PRI event/error monitor for the device a slot or a port. Examples: 1)`[no] debug e1t1` Enables/Disables the event and the error monitor for all e1t1 ports of the device. -detail: detail level <0-3> (Def: 0) -full-detail: Enables maximum debug output

Mode: Operator execution

Step	Command	Purpose
1	[*name*]**#show port hw-type [[<slot> <port>]**	Prints information about the specified port with a given detail level.

Chapter 57 E1/T1 Port Configuration

Chapter contents

Introduction

This chapter is incomplete. Please contact Patton Tech Support for assistance with configuring and trouble-shooting E1/T1 ports.

Patton support headquarters in the USA

- Online support: Available at www.patton.com
- E-mail support: E-mail sent to support@patton.com will be answered within 1 business day
- Telephone support: Standard telephone support is available five days a week—from **8:00 am** to **5:00 pm EST** (**1300** to **2200 UTC/GMT**)—by calling **+1 (301) 975-1007**
- Support via VoIP: Contact Patton free of charge by using a VoIP ISP phone to call sip:support@patton.com
- Fax: **+1 (301) 869-9293**

Alternate Patton support for Europe, Middle East, and Africa (EMEA)

- Online support: Available at www.patton-inalp.com
- E-mail support: E-mail sent to support@patton-inalp.com will be answered within 1 business day
- Telephone support: Standard telephone support is available five days a week—from **8:00 am** to **5:00 pm CET** (**0900** to **1800 UTC/GMT**)—by calling **+41 (0)31 985 25 55**
- Fax: **+41 (0)31 985 25 26**

Chapter 58 BRI Port Configuration

Chapter contents

Introduction

This chapter provides an overview of the BRI (Basic Rate Interface) ports, their characteristics and the tasks involved in the configuration. A BRI port supports two 64kbit/s B-channels for switched voice or data connections, one 16kbit/s D-channel for signaling and always-on data transfer. This results a usable data bit rate of 144kBit/s.

BRI Port Configuration task List

This section describes the configuration tasks for the BRI port.

- Enable/Disable BRI port
- Configuring BRI clock mode
- Configuring BRI Power-Feed
- Configuring BRI encapsulation
- BRI Debugging

Enable/Disable BRI Port

By default, the BRI port is disabled. The following command is used for enabling or disabling it.

Mode: port bri <slot> <port>

Step	Command	Purpose
1	[*name*] **(prt-bri)**[slot/port]**# [no] shutdown**	Enable/Disable the selected port. Default: *shutdown* (which is disabled)

Configuring BRI Clock-mode

The BRI Port can either work in clock-master or in clock-slave mode; the modes define the clock dependency of the internal data processing. In clock-master mode the internal data processing is running on an independent clock source. In clock-slave mode the clock source for internal data processing is recovered from the receive line interface. At all times, a port-pair of clock-master and clock-slave are connected together.

In the other case the data transmission will fail due to bit failures. The command below has the 'auto' option so it can be used if the application running on the port is also of an asymmetric nature like master/slave, server/client or user/net. Normally, the 'auto' option is used if the port is setup for ISDN. In this case, the clock mode will automatically derive from the Q.921 protocol. If the UNI-Side (User-Network Interface) of Q.921 is set to 'net', then clock mode of the port is automatically set to 'master' and in the other case to 'slave'.

Mode: port bri <slot> <port>

Step	Command	Purpose
1	[*name*] **(prt-bri)**[slot/port]**# clock {auto | master | slave}**	Configures the clock-mode of the port. Default: auto

Configuring BRI Power-Feed

Enable the application of power on the BRI port to provide power to ISDN terminals. This command applies only if the port is clock master (network side). It is only available on products with an internal, configurable ISDN power supply.

Mode: port bri <slot> <port>

Step	Command	Purpose
1	[*name*] **(prt-bri)**[slot/port]**#[no] power-feed**	Enables/Disables power-feed on the selected port. Default: disabled

Configure BRI Line-Termination

Switch internal termination impedance to on or off. Available for SN466x and SN467x devices for ports in TE/User/Slave mode.

Step	Command	Purpose
1	[*name*] **(prt-bri)**[slot/port]**#[no] line-termination**	Enables/Disables internal line-termination for TE ports. Default: disabled

Configuring BRI Encapsulation

The BRI encapsulation command prepares the port for a specific application protocol. After the right encapsulation type has been set, the configuration mode command for the selected protocol can be executed for protocol specific configuration.

q921: This encapsulation type automatically binds the signaling timeslot of the selected port to the ISDN Layer 2 protocol. For the BRI port this is the 16kbit/s D-channel. If in the q921 configuration mode q931 is specified as next encapsulation, the control of the two remaining timeslots (B-channels) is given to the ISDN Layer 3 protocol.

Mode: port bri <slot> <port>

Step	Command	Purpose
1	[*name*] **(prt-bri)**[slot/port]**#[no] encapsulation { channelized | q921}**	Specifies the encapsulation type of the BRI port. Default: *q921*

BRI Debugging

If a problem in the link is established, data transmission or synchronization, there are executable debug commands. The command has a hierarchical characteristic and can be applied to all ports on the whole device, or to all ports of a slot or just to one specific port. In addition, the 'show port' command can be used to printout information about the current configuration and about received and transmitted frames.

Mode: Operator execution

Step	Command	Purpose

1	**[node]#[no] debug bri { [<slot> [<port>]] | [detail <level>] | [full-detail] }**
	OR
	[node]#[no] trace bri { [<slot> [<port>]] | [alert | critical | emergency | error | info | notice | warning | { debug [detail <level>] | [full-detail] }] }
		Enables/Disables the numerous BRI traces viewed, and verbosity levels on a specific slot and BRI port.
		Default: no debug bri
		Examples:
		1) [no] debug bri 0 0 full detail
		Enables/Disables all debugs on the BRI port 0 on slot 0
		2) [no] trace bri 0 1 warning debug full detail
		Enables/Disables the warning trace in full detail on BRI port 1 on slot 0

Mode: Operator execution

Step	Command	Purpose
1	[*name*]**#show port bri [[<slot> <port>] | [detail <level>]]**	Prints information about the specified port with a given detail level.

BRI Configuration Examples

- Example 1: ISDN with auto clock/uni-side settings
- Example 2: ISDN with manual clock/uni-side settings

Example 1: ISDN with auto clock/uni-side settings

```
port bri 0 4
  power-feed
  encapsulation q921

  q921
    uni-side auto
    encapsulation q931

    q931
      protocol dss1
      uni-side net
      bchan-number-order ascending
      encapsulation cc-isdn
      bind interface bri04 switch

port bri 0 4
  no shutdown
```

Example 2: ISDN with manual clock/uni-side settings

```
port bri 0 4
  clock slave
  encapsulation q921
  q921
    uni-side user
    encapsulation q931

    q931
      protocol dss1
      uni-side user
      bchan-number-order ascending
      encapsulation cc-isdn
      bind interface bri04 switch

port bri 0 4 no shutdown
```

Chapter 59 Debug and Monitoring

Chapter contents

Introduction

This chapter describes how to use the internal debugging capabilities within Trinity. It provides debugging strategies to help locate the source of a problem, and describes the **show** and **debug** commands used to verify correct system operation and to troubleshoot problems.

This chapter includes the following sections:

- Debugging strategy (see page 662)
- Verifying IP connectivity (see page 663)
- Debugging call signaling (see page 663)
- Debugging voice data (see page 666)
- Debugging the SIP Stack (see page 666)

Debugging Strategy

Multi-service IP networks are highly comprised of sophisticated systems and protocols that offer a great many possibilities. Unfortunately, the possible sources of trouble are almost as many, so it is important to use a very methodical approach when tracking down a problem:

- Work from the bottom to the top of the protocol stack. Always start by checking your physicals. Verify that cables and connectors are in good shape, verify the link layer, and check IP connectivity before working on application problems.
- Work from the core to the edge. Problems always show up end-to-end, the phone does not ring, or the browser cannot find the web site. To track down network problems it is however helpful to start with a minimal number of hops, make sure everything is OK and then increase the end-to-end distance hop by hop.

IMPORTANT

Enabling some or all debug monitors may degrade system performance (IP routing, call signaling). To avoid inadvertent permanent system performance degradation make sure all debug monitors are switched off once the configuration is debugged and running.

Note Event log files record warnings and other information from system components. Entries in the logs are time-stamped with the actual system time, so make sure the device time and its system time are the same. Otherwise, you will not be able to get some information from the event logs because the time stamps are unusable.

You can enter the system time manually or have it be automatically set via SNTP. Refer to chapter 10, "Basic System Management" on page 97, or chapter 28, "SNTP Client Configuration" on page 283.

Verifying IP Connectivity

This procedure describes how to use the **ping** command to test IP connectivity. It verifies that your device can communicate with such hosts as an IP phone, a registrar, and other VoIP gateways.

Use Telnet to access your device, then use the **ping** command to verify that an IP packet can be sent to, and received from, all hosts with which the device should be able to communicate (e.g. IP phone, Registrar, other VoIP gateways).

Mode: Administrator execution

Step	Command	Purpose
1	***device*#ping** *ipaddress* [*number-of-packets*] **[timeout** *seconds*]	Determines whether an IP host can be contacted.

Network Sniff Tool

A new executable command sniff has been added. This command displays the network packets going through the given interface on the CLI or save them in a temporary file. The file content can be retrieved with the show capture command or it can be sent on a TFTP server using the copy capture:sniff.cap tftp://<target> command. The capture file can then be analyzed with Wireshark. In the background the command uses tcpdump 4.5.1, and we can also apply tcpdump filters using the expression option. For more detail on filters please refer to the tcpdump documentation: http://www.tcpdump.org/manpages/pcap-filter.7.html.

Mode: Administrator Execution

Step	Command	Purpose
1	**device#sniff [interface <ipInterface>] [save-to-file [file-size <megabytes>]] [packets <max-packets-to-capture>] [truncate <bytes-per-packet>] [expression <boolean-exp>]**	Capture as many packets as max-packets-to-capture on ipInterface and saves it to the capture file in the system. This file size is fixed to megabytes and it will overwrite itself once this limit is reached. To define how many bytes per packet should be captured, the option truncate can be used. expression option can be used to specify a tcpdump filter (see http://www.tcpdump.org/manpages/pcap-filter.7.html for more detail on filter)

Debugging Call Signaling

If calls do not connect, or disconnect during the process of connecting, there is a problem in call signaling. We suggest the following steps to debug call signaling:

1. Work from the call source to the call destination.
2. Make sure that the call enters correctly the context CS of your unit (debug the source signaling protocol, depending on where the call comes from).
3. Make sure that the call leaves correctly the context CS of your unit (debug the destination signaling protocol, depending on where the call goes to).

4. Debug call routing when the call enters the context CS, but it does not leave it. Remember that context CS must be activated (“no shutdown”) for call routing to work.

Please make yourself familiar with the context CS concept described in chapter 40, “CS Context Overview” on page 394. The following terminology will be used in this chapter:

- *Incoming call*: Call setup attempt from a call signaling protocol towards the context CS
- *Outgoing call*: Call setup attempt from the context CS towards a call signaling protocol

A basic call from an ISDN terminal connected to your unit over SIP to another SIP gateway is thus an *incoming* and *outgoing* call a the same time, from the context CS perspective: It *comes in* from ISDN and *goes out* over SIP.

Debugging ISDN Signaling

Overview: ISDN debug monitors

Step	Command	Purpose
1	**debug isdn-signaling [<detail-level> [0-5] | full-detail]**	Prints all ISDN layer 3 signaling messages, and call control related activity of the ISDN interface. This is a good monitor to start with when debugging ISDN.
2	**debug isdn-datapath [<detail-level> [0-5] | full-detail]**	Prints operations on the ISDN part of the voice data path. Use this monitor if you experience problems in the data path (no speech connectivity, speech only in one direction). This monitor is not needed to debug signaling.
3	**debug isdn-control [<detail-level> [0-5] | full-detail]**	Enables/Disables call-control ISDN control trace

Debugging SIP Signaling

Mode: Administrator Execution

Step	Command	Purpose
1	**debug sip-datapath [<detail-level> [0-5] | full-detail]**	Logs information related to the media channels
2	**debug sip-registration [<detail-level> [0-5] | full-detail]**	Logs information about user registration activities
3	**debug sip-signaling [<detail-level> [0-5] | full-detail]**	Logs call signaling related information
4	**debug sip-transport [<detail-level> [0-5] | full-detail]**	Logs all SIP messages sent or received over the IP network
5	**debug sip-location-service [<detail-level> [0-5] | full-detail]**	Logs information about the SIP Location-Service database

Verify an incoming call

Make sure that an incoming call from the SIP network enters correctly context CS. The following sequence shows a working call setup.

```
unit(cfg)#debug sip-transport
unit(cfg)#18:53:40  SIP_TR> Received INVITE sip:50@172.16.32.32 SIP/2.0
18:53:40 SIP_TR> Sent SIP/2.0 100 Trying
18:53:40 SIP_TR> Sent SIP/2.0 180 Ringing
18:53:43 SIP_TR> Sent SIP/2.0 200 OK
18:53:43 SIP_TR> Received ACK sip:50@172.16.32.32:5060 SIP/2.0
```

Explanation:

- The line **`18:53:40 SIP_TR> Received INVITE sip:50@172.16.32.32 SIP/2.0`** indicates that the *INVITE* message has been received. This means that the SIP network is functional
- **`18:53:40 SIP_TR > Sent SIP/2.0 100 Trying`** and **`18:53:40 SIP_TR> Sent SIP/2.0 180 Ringing`** indicate that responses are sent back to the SIP network. This means that the call routing is working correctly, and the call has found its destination on the gateway that is debugged. If there are no responses, or a negative response, continue debugging call routing and the destination protocol.

Verify an outgoing call

Make sure that an outgoing call from context CS leaves correctly to the SIP network. The following sequence shows a working call setup.

```
unit(cfg)#debug sip-transport
unit(cfg)#18:59:07 SIP_TR> Sent INVITE sip:60@172.16.32.33 SIP/2.0
18:59:07 SIP_TR> Received SIP/2.0 100 Trying
18:59:07 SIP_TR> Received SIP/2.0 180 Ringing
18:59:10 SIP_TR> Received SIP/2.0 200 OK
18:59:10 SIP_TR> Sent ACK sip:60@172.16.32.33:5060 SIP/2.0
```

Explanation:

- The line **`18:59:07 SIP_TR> Sent INVITE sip:60@172.16.32.33 SIP/2.0`** indicates that the *INVITE* was sent. Thus, call routing worked in context CS and the message left to the SIP network.
- **`18:59:07 SIP_TR > Received SIP/2.0 100 Trying`** indicate that responses are received from the SIP network. This means that IP connectivity is OK and the remote gateway can be reached. If there are no responses, or negative ones, continue debugging the remote SIP gateway.

Using Trinity's Internal Call Generator

The Trinity has a powerful internal call generator that creates calls from the center of context CS. It is very useful to debug call signaling or call routing problems to verify the correct working of one call signaling protocol at a time. Calls can be placed towards any interface on context CS, or any routing table within context CS.

Step	Command	Purpose
	To make a call: device(cfg)#call <calling-e164-number> dial <remote-e164-number> dest-<interface|service> <interface or service name> **To drop the call:** device(cfg)#call <called-e164-number> drop	Generates a test call internally within the device with the ability to send the call out an interface or call-service. Only signaling can be generated, no RTP.

Debugging the SIP Stack

The following commands will enable debugging of the SIP-Stack within Trinity VoIP Gateways

Overview: Operator execution

Step	Command	Purpose
1	**device>debug sip-stack-core**	Enables debug log traces to the current terminal.
2	**device>debug sip-stack-core-service**	Enables debug log traces to the current terminal.
3	**device>debug sip-stack-framework**	Enables debug log traces to the current terminal.
4	**device>debug sip-stack-parser**	Enables debug log traces to the current terminal.
5	**device>debug sip-stack-sdp**	Enables debug log traces to the current terminal.
6	**device>debug sip-stack-transaction**	Enables debug log traces to the current terminal.
7	**device>debug sip-stack-transport**	Enables debug log traces to the current terminal.
8	**device>debug sip-stack-useragent**	Enables debug log traces to the current terminal.

Debugging Voice Data

There are several debug monitors that can help identify problems in VoIP connections. The most common VoIP problems are: voice quality problems (dropouts), fax transmission errors, no establishment of voice connection, and wrong tone or playback. Dependent on the Patton devices, the output of the different Media Gateway monitors can differ.

An overview of all available Media Gateway debug monitors is given below. Some more specific examples for debugging cases follow after that.

Overview: Media Gateway debug monitors

[no] debug mg-dejitter	Displays changes to the settings of the dejitter buffer, exceptions (underrun, overrun, packet drops) and size changes. Usage: To investigate problems related to voice quality, voice packet payload sizes, delays, jitter

[no] debug mg-control [detail | full-detail]	Displays control activities on the Data Path (path of voice/fax data packets within your unit): State changes, tone start/stop, DTMF play-back/detection, fax/modem detection. Usage: To investigate problems with voice connections, DTMF, tone playback, fax and modem transmissions.
[no] debug mg-rtp [detail | full-detail]	Displays RTP related call parameters at call setup: local/remote IP address and port, SSRC. During operation, displays periodically updated statistics containing the number of sent and received packets, the number of lost packets. Usage: To verify that RTP packets are sent/received, and to debug network quality issues (lost packets).
[no] debug mg-switch [detail | full-detail]	Displays control activities on the TDM part of the Data Path. Usage: To investigate problems with hair pinning or timeslot switching.
[no] debug mg-dsp [detail | full-detail]	Displays control activities on the DSP including all call parameters (e.g. the used codec). Usage: To document the DSP parameters used for each call setup.
[no] debug mg-fax-data [detail | full-detail]	Traces detected Modem and Fax tones. Traces the flow of T.30 communication states between the fax machines. Decodes the IFP (Internet Fax Protocol) elements within T.38 packets. Displays T.38 related call parameters and operational errors like lost packets. Tracks the operation of the dejitter buffer used for T.38 Fax transmissions. Usage: To investigate problems with T.38 Fax and Modem transmissions in general.

Depending on the type of problem, some debug monitors are more useful than others. Try to avoid enabling all monitors at the same time, as this generates a lot of output and can degrade system performance. The following examples show some typical debug cases and what monitors should be switched on in these cases.

Example: Debugging voice connections

Symptoms: Voice quality is bad (dropouts), the voice connection is only established in one direction or not at all, there is only noise instead of voice, no tones are played (e.g. dialtone).

Prerequisite: The call is established from a signaling point of view.

Use the following debug monitors:

Step	Command	Purpose
1	***unit*#debug mg-rtp**	**Enable the RTP/RTCP debug monitor**
2	***unit*#debug mg-dejitter**	**Enable the dejitter buffer monitor.**

Example: Debugging Fax connections

Symptoms: Fax transmission starts but is interrupted and the Fax machines terminate with an error message, Fax transmission does not start at all, the unit's firmware does not detect Fax.

Prerequisite: Fax transmission is configured for T.38 relay in the voip profile. The call is established from a signaling point of view (see section "Debugging the SIP Stack" on page 666).

Attention: Special signaling procedures are used for the transition between voice and fax data transmission. It may be that the initial call setup is correct, but that the signaling to T.38 over SIP is faulty. The session-control and SIP signaling monitors (see Debug Session Control Data, and Debug SIP Data) might also be helpful to identify these cases.

Use the following debug monitors if you assume that signaling is OK:

Step	Command	Purpose
1	***unit*#debug mg-dsp**	Enable the dsp event monitor.
2	***unit*#debug mg-fax-data**	Enable the Fax/Modem event monitor
3	***unit*#debug mg-control**	Enable the control monitor

How to Submit Trouble Reports to Patton

Due the wealth of functionality and complexity of the products there remains a certain number of problems, either pertaining to the Patton product or the interoperability with other vendor's products.

If you have a problem for which you need supplier help please prepare and send the following information:

- **Problem description**—Add a description of the problem, if possible together with applicable augmented information with a diagram of the network setup (with Microsoft tools).
- **Running configuration and software and hardware version information**—With the Command Line Interface commands **show running-config** and **show version** you can display the currently active configuration of the system (in a Telnet and/or console session). Adding to the submitted trouble report will help us analyze the configuration and preclude possible configuration problems.In the unlikely case of a suspected hardware problem, also submit the serial number of your unit(s) and/or interface cards.
- **Event logs**—Add the system event logs, which you can display with the Command Line Interface commands **show log** and **show log supervisor**. To ensure that the logs are useful, it is necessary to set upon start up the clock to actual date and time (by hand or by enabling SNTP client)

If possible, add the following information in addition to the above:

- **Logs of protocol monitors**—Protocol traces contain a wealth of additional information which may be very helpful in finding or at least pinpointing the problem. Various protocol monitors with different levels of detail are an integral part of the firmware and can be started (in a Telnet and/or console session) individually (**debug** command).

  Note In order to correlate the output of different protocol monitors (e.g. ISDN signaling and gateway SIP signaling), run the monitors concurrently. You can do this either in the same Telnet session, or using different Telnet sessions.

- **Network traffic traces**—In certain cases it may be helpful to have a trace of the traffic on the IP network in order to inspect packet contents. Please use one of the following tools (supporting trace file formats which our tools can read):
  - Network Associates Sniffer—Details are available at www.sniffer.com

- TTC Firebird—Details are available at www.ttc.com
- Wireshark - Details are available at www.wireshark.com (freeware)

When possible, submit the package of trouble report files by mail to the following address: support@patton.com (use fax only in exceptional cases).

PacketSmart

PacketSmart, the network monitoring tool developed by BroadSoft, has been integrated into Trinity for the products SN4970, SN4980, SN4990, SN5480, SN5490, SN5300, SN4130, SN4170, SN5530, and SN5570. In order to use it, a Patton PacketSmart License needs to be purchased and activated.

Step	Command	Purpose
1	**node(cfg)]#profile packetsmart <pf-name>**	Access PacketSmart profile mode
2	**node(pf-psmart)[<pf-name>]]#[no] bind interface <ip-if>**	Choose an IP interface to upload data to PacketSmart server
3	**node(pf-psmart)[<pf-name>]]#server [<ip-or-host> [[port <port-number>] [ip-mode { ipv4 | ipv6 }]]]**	Configure the URI of the PacketSmart server and the IP protocol (IPv4 or IPv6) to communicate with it
4	**node(pf-psmart)[<pf-name>]]#assessment sip port [<port>]**	Configure SIP port to monitor
5	**node(pf-psmart)[<pf-name>]]#assessment rtp port-range [<port-min> <port-max>]**	Configure RTP ports to monitor
6	**node(prt-eth)[<slot/port>]]#[no] use profile packetsmart <pf-name>**	In Port mode (DSL/ETH) or Switch/Bridge mode, use profile packetsmart to start monitoring

Chapter 60 Contacting Patton for Assistance

Chapter contents

Introduction

This chapter contains the following information:

- "Contact information"—describes how to contact Patton technical support for assistance.
- "Warranty Service and Returned Merchandise Authorizations (RMAs)"—contains information about the warranty and obtaining a return merchandise authorization (RMA).

Contact information

Patton Electronics offers a wide array of free technical services. If you have questions about any of our other products we recommend you begin your search for answers by using our technical knowledge base. Here, we have gathered together many of the more commonly asked questions and compiled them into a searchable database to help you quickly solve your problems.

Contacting Patton Technical Services for Free Support

REGION	North America	Western Europe	Central & Eastern Europe	Middle East North Africa
Location	**Maryland, USA**	**Bern, Switzerland**	**Budapest, Hungary**	**Beirut, Lebanon**
Time Zone	EST/EDT UTC/GMT - 4/5 hours	CET/CEDT UTC/GMT + 1/2 hours	CET/CEDT UTC/GMT + 1/2 hours	EET/EEDT UTC/GMT + 2/3 hours
Business Hours	Monday-Friday 8:00am to 5:00pm	Monday-Friday 09:00 to 12:00 13:30 to 17:30	Monday-Friday 8:30 to 17:00	Monday-Friday 8:00am to 5pm
Email	support@patton.com	support@patton.com	support@patton.com	support@patton.com
Phone	+ 1 301 975 1007	+41 31 985 25 55	+36 439 3835	+96 1 359 1277
Fax	+1 301 869 9293	+41 31 985 2526		

Warranty Service and Returned Merchandise Authorizations (RMAs)

Patton Electronics is an ISO-9001 certified manufacturer and our products are carefully tested before shipment. All of our products are backed by a comprehensive warranty program.

> **Note** If you purchased your equipment from a Patton Electronics reseller, ask your reseller how you should proceed with warranty service. It is often more convenient for you to work with your local reseller to obtain a replacement. Patton services our products no matter how you acquired them.

Warranty coverage

Our products are under warranty to be free from defects, and we will, at our option, repair or replace the product should it fail within one year from the first date of shipment. Our warranty is limited to defects in workmanship or materials, and does not cover customer damage, lightning or power surge damage, abuse, or unauthorized modification.

Out-of-warranty service

Patton services what we sell, no matter how you acquired it, including malfunctioning products that are no longer under warranty. Our products have a flat fee for repairs. Units damaged by lightning or other catastrophes may require replacement.

Returns for credit

Customer satisfaction is important to us, therefore any product may be returned with authorization within 30 days from the shipment date for a full credit of the purchase price. If you have ordered the wrong equipment or you are dissatisfied in any way, please contact us to request an RMA number to accept your return. Patton is not responsible for equipment returned without a Return Authorization.

Return for credit policy

- Less than 30 days: No Charge. Your credit will be issued upon receipt and inspection of the equipment.
- 30 to 60 days: We will add a 20% restocking charge (crediting your account with 80% of the purchase price).
- Over 60 days: Products will be accepted for repairs only.

RMA numbers

RMA numbers are required for all product returns. You can obtain an RMA by doing one of the following:

- Completing a request on the RMA Request page in the *Support* section at **www.patton.com**
- By calling **+1 (301) 975-1007** and speaking to a Technical Support Engineer
- By sending an e-mail to **returns@patton.com**

All returned units must have the RMA number clearly visible on the outside of the shipping container. Please use the original packing material that the device came in or pack the unit securely to avoid damage during shipping.

Shipping instructions

The RMA number should be clearly visible on the address label. Our shipping address is as follows:

Patton Electronics Company
RMA#: xxxx
7622 Rickenbacker Dr.
Gaithersburg, MD 20879-4773 USA

Patton will ship the equipment back to you in the same manner you ship it to us. Patton will pay the return shipping costs.

Appendix A Trinity Architecture Terms and Definitions

Chapter contents

Introduction

This chapter contains the terms and their definitions that are used throughout this *Trinity Release 3.0 Command Line Reference Guide*. This guide contains many terms that are related to specific networking technologies areas such as LAN protocols, WAN technologies, routing, Ethernet, and Frame Relay. Moreover various terms are related to telecommunication areas.

Term or Definition	Meaning
Administrator	The person who has privileged access to the CLI.
Application Download	A application image is downloaded from a remote TFTP server to the persistent memory (flash:) of a Patton device.
Application Image	The binary operation code stored in the persistent memory (flash:) of a Patton device.
Batchfile	Script file containing instructions to download one or more software component from a TFTP server to the persistent memory (flash: or nvram:) of a Patton device.
Bootloader	The bootloader is a “mini” application performing basic system checks and starting the application image. The bootloader also provides minimal network services allowing the Patton device to be accessed and upgraded over the network even if the application image should not start. The bootloader is installed in the factory and is in general never upgraded.
Bootloader Image	The binary code of the Bootloader stored in the persistent memory (flash:) of a Patton device.
Bootstrap	The starting-up of a Patton device, which involves checking the Reset button, loading and starting the application image, and starting other software modules, or—if no valid application image is available—the bootloader.
Build	The released software is organized as builds. Each build has its unique identification. A build is part of a release and has software bug fixes. See also *release*.
Call Routing	Calls through Patton device can be routed based on a set of routing criteria. See also Sess*ion Rou*ter.
Call Signaling	The call signaling specifies how to set up a call to the destination Patton device or 3rd party equipment.
Circuit	A communication path between two or more devices.
Circuit Port	Physical port connected to a switching system or used for circuit switching.
Circuit Switching	The switching system in which a dedicated physical circuit path must exist between the sender and the receiver for the duration of the call. Used in the conventional telephone network.
Codec	Abbreviation for the word construct Coder and Decoder. Voice channels occupy 64 kbps using PCM (pulse code modulation) coding. Over the years, compression techniques were developed allowing a reduction in the required bandwidth while preserving voice quality. Such compression techniques are implemented within a Codec.

Term or Definition	Meaning
Comfort Noise	Comfort noise is generated at the remote end of the silent direction to avoid the impression that the connection is dead. See also *Silence Compression*.
Command Line Interface	An interface that allows the user to interact with the Trinity operating system by entering commands and optional arguments. Other operating systems like UNIX or DOS also provide CLIs.
Configuration Download	A configuration file is downloaded from a remote TFTP server via TFTP to the persistent memory (nvram:) or volatile memory (system:)of a Patton device.
Configuration File	The configuration file contains the CLI commands, which are used to configure the Patton device.
Configuration Server	A central server used as a store for configuration files, which are downloaded to or uploaded from a Patton device using TFTP.
Configuration Upload	A configuration file is uploaded from the persistent memory (nvram:) or volatile memory (system:) of a Patton device via TFTP to a TFTP server.
Context	Represents one specific networking technology or protocol, e.g. IP or circuit switching.
Data Port	Physical port connected to a network element or used for data transfer.
Dejitter Buffer	The element used to compensate variable network delays. Storing packets in a dejitter buffer before they are transferred to the local ISDN equipment, e.g. telephone, a variable delay is converted into a fixed delay, giving voice a better quality. See also *Jitter*.
Digit Collection	Some devices (PBX, ISDN network, and remote gateways) may require bloc sending of the dialed number. Digit collection collects the overlap dialed digits and forwards them in a single call setup message
Driver Software Download	A driver software image is downloaded from a remote TFTP server to the persistent memory (flash:) of a Patton device.
Driver Software Image	The software used for peripheral chips on the main board and optional PMC interface cards is stored in the persistent memory (flash:) of a Patton device.
DTMF Relay	DTMF relay solves the problem of DTMF distortion by transporting DTMF tones over low-bit-rate codecs out-of-band or separate from the encoded voice stream
Echo Canceller	Some voice devices unfortunately have got an echo on their wire. Echo cancellation provides near-end echo compensation for this device.
Factory Configuration	The factory configuration (factory-config) represents the system default settings and is stored in the persistent memory (nvram:) of a Patton device.
Fast Connect	Since a normal call setup is often too slow, fast connect is a new method of call setup that bypasses some usual steps in order to make it faster.
Flash Memory	Persistent memory section of a Patton device containing the Application Image, Bootloader Image and the driver software Image.
flash:	A region in the persistent memory of a Patton device. See also *flash memory*.

Term or Definition	Meaning
Gateway	A gateway refers to a special purpose component that connects two contexts of different types, For example, the CS and the IP context. It handles connections between different technologies or protocols.
High-Pass Filter	A high-pass filter is normally used to cancel noises at the voice coder input. See also *post filter*
Host	Computer system on a network. Similar to node, except that host usually implies a PC or workstation, whereas node generally applies to any networked system, including access servers and routers. See also *node*.
Hostname	Name given to a computer system, e.g. a PC or workstation.
Hunt Group	In Patton device terminology, a hunt groups allows you to apply the interface configuration to multiple physical ports. Within the hunt groups free channels for outgoing calls are hunted on all available ports. In general a hunt group represents a group of trunk lines as used for direct dialing in (DDI).
Interface	An interface is a logical construct that provides higher-layer protocol and service information. An Interface is configured as a part of a context, and is independent of a physical port or circuit.
Interface Card	An optional plug-in card offering one or more ports of a specific physical standard for connecting the Patton device to the outside world.
ISDN	Integrated Services Digital Network
ISDN Services	ISDN Services comprise voice, data, video and supplementary services. Supplementary services are services available in the ISDN network, such as calling line identification presentation (CLIP) or call waiting (CW). See also *Q.SIG*
Jitter	Jitter is the variation on packets arriving on a Patton device. See also *dejitter buffer*.
Mode	The CLI is comprised of modes. There are two basic mode groups, the execution mode group and the configuration mode group.
Network Management System	System responsible for managing at least part of a network. An NMS is generally a reasonably powerful and well-equipped computer, such as an engineering workstation. NMSs communicate with agents to help keep track of network statistics and resources.
Node	Endpoint of a network connection or a junction common to two or more lines in a network. A Node can be a router, e.g. a Patton device. Nodes, which vary in routing and other functional capabilities, can be interconnected. Node sometimes is used generically to refer to any entity that can access a network, and frequently is used interchangeably with device.
Nodename	Name given to a Patton device or network element.
nvram:	Persistent memory section of a Patton device containing the startup configuration, the factory configuration and used defined configurations.
Operator	The person who has limited access to the CLI.

Term or Definition	Meaning
PCI Local Bus	The PCI Local Bus is a high performance, 32-bit or 64-bit bus with multiplexed address and data lines. The bus is intended for use as an interconnect mechanism between highly integrated peripheral controller components, peripheral add-in boards, and processor/memory systems.
PCM Highway	A 30 channel interface connecting the switching engine with optional interface cards containing circuit ports.
Port	A port represents a physical connector on the Patton device.
Port Address	A port address can be assigned to a CS interface to realize a virtual voice tunnel between two nodes.
Post Filter	The voice decoder output is normally filtered using a perceptual post-filter to improve voice quality. See also *High-Pass Filter*.
POTS	Plain Old Telephone Service
Profile	A profile provides configuration shortcutting. A profile contains specific settings that can be used on multiple contexts, interfaces or gateways.
PSTN	Public Switched Telephone Network. Contains ISDN and POTS
Q.931 Tunneling	Q.931 tunneling is able to support ISDN services and Q.SIG over an IP network.
Q.SIG	ISDN Services comprise additional services for the Private ISDN network such as CNIP (Calling Name Identification Presentation), CNIR (Calling Name Identification Restriction) etc. See also *ISDN Services*.
Release	A software release describes the main voice and data feature set. It consists of a series of builds.
Routing Engine	The routing engine handles the basic IP routing.
Running Configuration	The currently running configuration (running-config), which is executed from the volatile memory (system:).
Session Router	Calls through Patton device can be routed based on a set of routing criteria. The entity that manages call routing is called Session Router.
Session Initiation Protocol (SIP)	Used for setting up communications sessions (such as conferencing, instant messaging, and telephony) on the Internet
Silence Compression	Silence suppression (or compression) detects the silent periods in a phone conversation and stops the sending of media packets during this periods.
Startup Configuration	The startup configuration is stored in the persistent memory (nvram:) and is always copied for execution to the running configuration in the volatile memory (system:) after a system start-up.
Switching Engine	Part of the Patton device hardware which allows software controlled circuit switching of circuit ports.
System Image	A collective term for application images and interface card driver software, excluding configuration files.
System Memory	The volatile memory, that includes the system: region, holding the running-config for the Trinity during operation of a Patton device.
system:	A region in the volatile memory of a Patton device. See also *system memory.*

Term or Definition	Meaning
TFTP Server	A central server used for configuration up- and download, download of application and interface card driver software, that is accessed using TFTP.
tftp:	Identification of a remote storing location used for configuration up- and download, download of application and interface card driver software, that is accessed using TFTP.

Appendix B Command summary

Chapter contents

Introduction

This chapter provides an overview of all CLI commands and modes available. It is organized as follows:

```
Mode Name
Enter Command
Command 1
…
Exit

Mode Name
…
```

Several commands contain a lot of parameters and arguments. The command syntax is described as follows:

- Arguments where you must supply the value are surrounded by <angle brackets>.
- Optional arguments within commands are shown in square brackets ([ ])
- Alternative parameters within commands are separated by vertical bars (|).
- Alternative but required parameters are shown within grouped braces ({ }) and are separated by vertical bars (|).

Command syntax is illustrated by an example in figure 102.

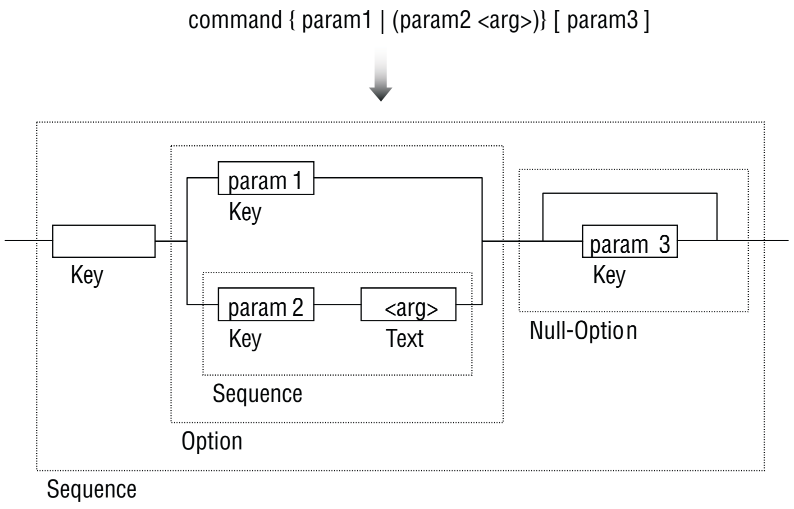


Figure 102. EBNF syntax

New Configuration Commands

The commands documented in the Release Note only cover new additions which are not yet included in the current revision of the Software Configuration Guide. You may download the release notes at www.patton.com/support.

Current Revision:

Document Number: 13211U8-001 Rev. D

Part Number: 07MSWR320_SCG

Revised: July 17, 2006

Other

Show help

Step	Command	Purpose
1	**help** [*topic*]	Shows command help.

Show command history

Step	Command	Purpose
1	**history**	Shows command history.

Use CTRL-N and CTRL-P to browse. The cursor keys (up, down) are not working.

Restart system

Step	Command	Purpose
1	**reset**	Restarts the system.

Appendix C Glossary of Terms

Chapter contents

List of terms

Term	Meaning
Numeric	
10Base-T	Ethernet Physical Medium
A	
AAL	ATM Adaptive Layer
ABR	Available Bit Rate
AC	Alternating Current
AOC	Advice of Charge
ATM	Asynchronous Transfer Mode
audio 3.1	ISDN Audio Service up to 3.1 kHz
audio 7.2	ISDN Audio Service up to 7.2 kHz
B	
BRA	Basic Rate Access
BRI	Basic Rate Interface
C	
CAC	Carrier Access Code
CBR	Constant Bit Rate
CD ROM	Compact Disc Read Only Memory
CDR	Call Detail Record
CFP	Call Forwarding Procedure
CLEC	Competitive Local Exchange Carriers
CLI	Command Line Interface
CLIP	Calling Line Identification Presentation
CO	Central Office
CPE	Customer Premises Equipment
CPU	Central Processor Unit
CRC32	32 bit Cyclic Redundancy Check
D	
DC	Direct Current
DDI	Direct Dialing In number
DHCP	Dynamic Host Configuration Protocol
DLCI	Data Link Connection Identifier
DSL	Digital Subscriber Line
DSLAM	Digital Subscriber Line Access Multiplexor
DSP	Digital Signal Processor
DTMF	Dual Tone Multi-frequency
E	
E1	Transmission Standard at 2.048 Mb/s

Term	Meaning
E-DSS1	ETSI Euro ISDN Standard
EFS	Embedded File System
ET	Exchange Termination
ETH	Ethernet
F	
FAQ	Frequently Asked Questions
FCC	Federal Communication Commission
FR	Frame Relay
G	
G.711	ITU-T Voice encoding standard
G.723	ITU-T Voice compression standard
GUI	Graphic User Interface
GW	Gateway
H	
HFC	Hybrid Fiber Coax
HTTP	HyperText Transport Protocol
HW	Hardware
I	
IAD	Integrated Access Device
ICMP	Internet Control Message Protocol
ILEC	Incumbent Local Exchange Carriers
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISDN NT	ISDN Network Termination
ISDN S	ISDN S(ubscriber Line) Interface
ISDN T	ISDN T(runk Line) Interface
ISDN TE	ISDN Network Terminal Mode
ITC	Information Transfer Bearer Capability
L	
L2TP	Layer Two Tunneling Protocol
LAN	Local Area Network
LCR	Least Cost Routing
LDAP	Lightweight Directory Access Protocol
LE	Local Exchange
LED	Light Emitting Diode
LT	Line Termination
M	
MIB II	Management Information Base II
Modem	Modulator – Demodulator

Term	Meaning
MSN	Multiple Subscriber Number
N	
NAPT	Network Address Port Translation
NAT	Network Address Translation
NIC	Network Interface Card
NT	Network Termination
NT1	Network Termination 1
NT2	Network Termination 2
NT2ab	Network Termination with 2a/b Connections
O	
OEM	Original Equipment Manufacturer
OSF	Open Software Foundation
OSPF	Open Shortest Path First
P	
PBR	Policy Based Routing (principles)
PBX	Private Branch Exchange
PC	Personal Computer
PMC	Production Technology Management Commit-tee
POP	Point of Presence
POTS	Plain Old Telephony Service
PRA	Primary Rate Access
PRI	Primary Rate Interface
PSTN	Public Switched Telephone Network
pt-mpt	point-to-multi point
pt-pt	point-to-point
PVC	Permanent Virtual Circuit
pwd	Password
PWR	Power
Q	
QoS	Quality of Service
R	
RIPv1	Routing Information Protocol Version 1
RIPv2	Routing Information Protocol Version 2
RJ-45	Western Connector Type
RTM	Route Table Manager
RTP	Real-time Protocol
S	
S1	node-connection for Trunk Line
S2	node-connection for Subscriber Line

Term	Meaning
SAR	Segmentation and Reassembly
S-Bus	Subscriber Line (Connection) Bus
SCN	Switched Circuit Network
SCTP	Stream Control Transmission Protocol
SDSL	Symmetric Digital Subscriber Line
SGCP	Simple Gateway Control Protocol
SIP	Session Initiation Protocol.
SME	Small and Medium Enterprises
SNMP	Simple Network Management Protocol
SOHO	Small Office Home Office
SONET	Synchronous Optical Network
SS7	Signaling System No. 7
STM	SDH Transmission at 155 Mb/s
SVC	Switched Virtual Circuit
SW	Software
T	
TCP/IP	Transport Control Protocol/Internet Protocol
TE	Terminal Equipment
TFTP	Trivial File Transfer Protocol
U	
UBR	Unspecified Bit Rate
UD 64	Unrestricted Data 64 kb/s
UDP	User Datagram Protocol
V	
VBR	Variable Bit Rate
VCI	Virtual Channel Identifier
VoIP	Voice over Internet Protocol
VPI	Virtual Path Identifier
W	
WAN	Wide Area Network