



Extreme 9920 Software gNOI Protocol Reference, 21.1.0.0

Supporting Extreme 9920

9037110-00 Rev AA
June 2021



Copyright © 2021 Extreme Networks, Inc. All rights reserved.

Legal Notice

Extreme Networks, Inc. reserves the right to make changes in specifications and other information contained in this document and its website without prior notice. The reader should in all cases consult representatives of Extreme Networks to determine whether any such changes have been made.

The hardware, firmware, software or any specifications described or referred to in this document are subject to change without notice.

Trademarks

Extreme Networks and the Extreme Networks logo are trademarks or registered trademarks of Extreme Networks, Inc. in the United States and/or other countries.

All other names (including any product names) mentioned in this document are the property of their respective owners and may be trademarks or registered trademarks of their respective companies/owners.

For additional information on Extreme Networks trademarks, see: www.extremenetworks.com/company/legal/trademarks

Open Source Declarations

Some software files have been licensed under certain open source or third-party licenses. End-user license agreements and open source declarations can be found at: <https://www.extremenetworks.com/support/policies/open-source-declaration/>



Table of Contents

Preface.....	4
Text Conventions.....	4
Documentation and Training.....	5
Getting Help.....	6
Subscribe to Product Announcements.....	6
Providing Feedback.....	6
About this Document.....	8
What's New in this Document.....	8
Extreme 9920 gNOI Protocols.....	9
packet_capture.proto.....	9
config_mgmt.proto.....	11
auth.proto.....	12
os.proto.....	14
OpenConfig gNOI Protocols.....	20
file.proto.....	21
interface.proto.....	24



Preface

Read the following topics to learn about:

- The meanings of text formats used in this document.
- Where you can find additional information and help.
- How to reach us with questions and comments.

Text Conventions

Unless otherwise noted, information in this document applies to all supported environments for the products in question. Exceptions, like command keywords associated with a specific software version, are identified in the text.

When a feature, function, or operation pertains to a specific hardware product, the product name is used. When features, functions, and operations are the same across an entire product family, such as ExtremeSwitching switches or SLX routers, the product is referred to as *the switch* or *the router*.

Table 1: Notes and warnings






Icon	Notice type	Alerts you to...
	Tip	Helpful tips and notices for using the product
	Note	Useful information or instructions
	Important	Important features or instructions
	Caution	Risk of personal injury, system damage, or loss of data
	Warning	Risk of severe personal injury

Table 2: Text

Convention	Description
screen displays	This typeface indicates command syntax, or represents information as it is displayed on the screen.
The words <i>enter</i> and <i>type</i>	When you see the word <i>enter</i> in this guide, you must type something, and then press the Return or Enter key. Do not press the Return or Enter key when an instruction simply says <i>type</i> .
Key names	Key names are written in boldface, for example Ctrl or Esc . If you must press two or more keys simultaneously, the key names are linked with a plus sign (+). Example: Press Ctrl+Alt+Del
Words in italicized type	Italics emphasize a point or denote new terms at the place where they are defined in the text. Italics are also used when referring to publication titles.
NEW!	New information. In a PDF, this is searchable text.

Table 3: Command syntax

Convention	Description
bold text	Bold text indicates command names, keywords, and command options.
<i>italic</i> text	Italic text indicates variable content.
[]	Syntax components displayed within square brackets are optional. Default responses to system prompts are enclosed in square brackets.
{ x y z }	A choice of required parameters is enclosed in curly brackets separated by vertical bars. You must select one of the options.
x y	A vertical bar separates mutually exclusive elements.
< >	Nonprinting characters, such as passwords, are enclosed in angle brackets.
...	Repeat the previous element, for example, <i>member</i> [<i>member</i> ...].
\	In command examples, the backslash indicates a “soft” line break. When a backslash separates two lines of a command input, enter the entire command at the prompt without the backslash.

Documentation and Training

Find Extreme Networks product information at the following locations:

[Current Product Documentation](#)

[Release Notes](#)

[Hardware and software compatibility](#) for Extreme Networks products

[Extreme Optics Compatibility](#)

[Other resources](#) such as white papers, data sheets, and case studies

Extreme Networks offers product training courses, both online and in person, as well as specialized certifications. For details, visit www.extremenetworks.com/education/.

Getting Help

If you require assistance, contact Extreme Networks using one of the following methods:

Extreme Portal

Search the GTAC (Global Technical Assistance Center) knowledge base; manage support cases and service contracts; download software; and obtain product licensing, training, and certifications.

The Hub

A forum for Extreme Networks customers to connect with one another, answer questions, and share ideas and feedback. This community is monitored by Extreme Networks employees, but is not intended to replace specific guidance from GTAC.

Call GTAC

For immediate support: (800) 998 2408 (toll-free in U.S. and Canada) or 1 (408) 579 2826. For the support phone number in your country, visit: www.extremenetworks.com/support/contact

Before contacting Extreme Networks for technical support, have the following information ready:

- Your Extreme Networks service contract number, or serial numbers for all involved Extreme Networks products
- A description of the failure
- A description of any actions already taken to resolve the problem
- A description of your network environment (such as layout, cable type, other relevant environmental information)
- Network load at the time of trouble (if known)
- The device history (for example, if you have returned the device before, or if this is a recurring problem)
- Any related RMA (Return Material Authorization) numbers

Subscribe to Product Announcements

You can subscribe to email notifications for product and software release announcements, Field Notices, and Vulnerability Notices.

1. Go to [The Hub](#).
2. In the list of categories, expand the **Product Announcements** list.
3. Select a product for which you would like to receive notifications.
4. Select **Subscribe**.
5. To select additional products, return to the **Product Announcements** list and repeat steps 3 and 4.

You can modify your product selections or unsubscribe at any time.

Providing Feedback

The Information Development team at Extreme Networks has made every effort to ensure the accuracy and completeness of this document. We are always striving to improve our documentation and help you work better, so we want to hear from you. We welcome all feedback, but we especially want to know about:

- Content errors, or confusing or conflicting information.

- Improvements that would help you find relevant information in the document.
- Broken links or usability issues.

If you would like to provide feedback, you can do so in three ways:

- In a web browser, select the feedback icon and complete the online feedback form.
- Access the feedback form at <https://www.extremenetworks.com/documentation-feedback/>.
- Email us at documentation@extremenetworks.com.

Provide the publication title, part number, and as much detail as possible, including the topic heading and page number if applicable, as well as your suggestions for improvement.



About this Document

What's New in this Document on page 8

gRPC Network Operations Interface (gNOI) defines a set of gRPC-based micro-services for executing operational commands on network devices. gNOI protocols use gRPC as the transport protocol and the configuration is same as that of gNMI (<https://github.com/openconfig/gnoi>).

What's New in this Document

This document is new for the release of the Extreme 9920 software with the NPB application.

For more information about this release, see the [Extreme 9920 Software Release Notes, 21.1.0.0](#).



Extreme 9920 gNOI Protocols

[packet_capture.proto](#) on page 9

[config_mgmt.proto](#) on page 11

[auth.proto](#) on page 12

[os.proto](#) on page 14

The following topics describe the supported gNOI protocols.

packet_capture.proto

Defines a gNOI protocol API for PCAP feature.

Table 4: Packet capture remote procedure calls

RPC	Purpose
EnablePCAPOnInterface	Enable packet capture for the specified interface.
DisablePCAPOnInterface	Disable packet capture on the specified interface.
ListEnabledInterfaces	List all interfaces enabled for packet capture.
DeletePCAPFile	Delete the pcap file.
StartStopPcap	Start or stop packet capture.
RetrieveStartStopPcap	RetrieveStartStopPcap

```
syntax = "proto3";  
  
package pcap;  
  
service PacketCapture {  
    // EnablePCAPOnInterface will enable to capture the packets for the provided interface.  
    rpc EnablePCAPOnInterface(PcapInterface) returns (PcapResponse) {};  
  
    // DisablePCAPOnInterface will disable to capture the packets on interface  
    rpc DisablePCAPOnInterface(InterfaceInfo) returns (PcapResponse) {};  
  
    //ListEnabledInterfaces list all the interfaces enabled for packet capture.  
    rpc ListEnabledInterfaces(ListEnabledInterfacesRequest) returns  
    (PcapEnabledInterfacesResponse) {};  
  
    // DeletePCAPFile delete the pcap file  
    rpc DeletePCAPFile(DeletePcapFileRequest) returns (PcapResponse) {};  
  
    // StartStopPcap  
    rpc StartStopPcap(StartStopPcapRequest) returns (PcapResponse) {};
```

```

    // RetrieveStartStopPcap
    rpc RetrieveStartStopPcap(RetrieveStartStopPcapRequest) returns
(RetrieveStartStopPcapResponse) {};
}

message ListEnabledInterfacesRequest {
}

message RetrieveStartStopPcapRequest {
}

message PcapServiceRequest {
    oneof Request {
        PcapInterface pcap_interface = 1;
        InterfaceInfo interface_info = 2;
        ListEnabledInterfacesRequest list_enabled_interfaces_request = 3;
        DeletePcapFileRequest delete_pcap_file_request = 4;
        StartStopPcapRequest start_stop_pcap_request = 5;
        RetrieveStartStopPcapRequest retrieve_start_stop_pcap_request = 6;
    }
}

// It is a reusable message which has the info of interface name and type.

message InterfaceInfo {
//name of the interface
    string name = 1;
//type of the interface
    string type = 2;
}

// PcapInterface is the message for the packet capture based on the interface,
direction and filter
message PcapInterface {
    InterfaceInfo interface_info = 1;
    enum Direction {
        RX = 0;
        TX = 1;
        BOTH = 2;
    }
    Direction direction = 2;
    int32 packet_count = 3;
    enum Protocol {
        L2 = 0;
        L3 = 1;
        ALL = 3;
    }
    Protocol protocol_info = 4;
}

// The PcapResponse is sent from the Target to the Client in response to the
// enable or disable packet capture RPCs and for deleting PCAP file. It indicates the
success
//or error case
message PcapResponse {
    oneof response {
        Success success = 1;
        Failure failure = 2;
    }
}

//The target will reply as CaptureOK

```

```

message Success {}
//the target will reply with Failure message if any.
message Failure {
    string error_message =1;
}

//The PcapEnabledInterfacesResponse will respond with the interface info
//for PCAP enabled interfaces with details about direction, packet count , protocol
message PcapEnabledInterfacesResponse {
    repeated PcapInterface pcap_interface = 1;
}
//The DeletePcapFileRequest is used as request message for deleting the Pcap file
// where the request contains the pcap file name.
message DeletePcapFileRequest {
    string file_name =1;
}

message StartStopPcapRequest {
    enum StartStopEnum {
        start = 0;
        stop = 1;
    }
    StartStopEnum StartStop = 1;
}

message RetrieveStartStopPcapResponse {
    StartStopPcapRequest start_stop_pcap_request = 1;
}

```

config_mgmt.proto

Defines a gNOI protocol API for config management feature.

Table 5: Configuration management remote procedure calls

RPC	Purpose
CopyDefaultToRunning	Copy default-config to running-config.

```

syntax = "proto3";

package cfgmgmt;

import "github.com/openconfig/gnoi/types/types.proto";

//The ConfigManagement service provides an interface for config related operations on
Target.
service ConfigManagement {
    //CopyDefaultToRunning RPC is used for copying default-config to running-config
    //This leads to auto reboot of the device on success
    rpc CopyDefaultToRunning(CopyDefaultRunningRequest) returns (CopyResponse) {};
}

message CopyRequest {
    oneof Request {
        CopyDefaultRunningRequest copy_default_running_request = 1;
    }
}

//CopyDefaultRunningRequest message is sent whenever default-config need to to be applied
to running-config.

```

```

message CopyDefaultRunningRequest {
}

//CopyResponse returns with success or failure in response to CopyDefaultRunningRequest
request
message CopyResponse {
  oneof response {
    Success success = 1;
    Failure failure = 2;
  }
}

//Success message is used to inform the client that the operation is succesfull.
message Success {}

//Failure message is used to inform the client that the operation has failed
//error message is used to return with the failure reason.
message Failure {
  string error_message =1;
}

```

auth.proto

Defines a gNOI protocol for an Authenticate API.

Table 6: Authentication remote procedure calls

RPC	Purpose
Authenticate	Validate credentials and provide a JWT access token.
GetAccessToken	Accept a valid refresh token and generate a new access token.
ListRoles	List all available roles.

```

syntax = "proto3";

package auth;

service Auth {
  // Authenticate will validate user credentials and provides access token in response
  rpc Authenticate (AuthenticateRequest) returns (AuthenticateResponse) {}

  // GetAccessToken will accept a valid refresh token and generate a new access token
  rpc GetAccessToken (TokenRequest) returns (TokenResponse) {}

  // ListRoles lists all the available roles in the system
  rpc ListRoles (ListRolesRequest) returns (ListRolesResponse) {};
}

message AuthenticateRequest {
  // user credentials
  string username = 1;
  string password = 2;
}

message AuthenticateResponse {
  // JWT access token
  string access_token = 1;
  // JWT refresh token

```

```
    string refresh_token = 2;
  }

  message TokenRequest {
    // JWT refresh token
    string refresh_token = 1;
  }

  message TokenResponse {
    // JWT access token
    string access_token = 1;
  }

  message ListRolesRequest {
  }

  //ListRolesResponse returns information about all roles available in the system
  message ListRolesResponse {
    repeated RoleInfo role_info = 1;
  }

  //RoleInfo contains the role specific information like role name, type and description
  message RoleInfo {
    string role_name = 1;
    enum RoleType {
      UNSPECIFIED = 0;
      SYSTEM_DEFINED = 1;
    }
    RoleType role_type = 2;
    string description = 3;
  }
}
```

os.proto

Defines a gNOI protocol API used for operating system installation.

Table 7: Operating system remote procedure calls

RPC	Purpose
Activate	Set the requested OS version as the version to be used at the next reboot, then reboot the target. If reboot fails, target recovers by booting the previously running OS package.
Verify	Check the running OS version.

```

syntax = "proto3";

package gnoi.os;

import "github.com/openconfig/gnoi/types/types.proto";

option go_package = "github.com/openconfig/gnoi/os";

option (types.gnoi_version) = "0.1.1";

// The OS service provides an interface for OS installation on a Target. The
// Client progresses through 3 RPCs:
// 1) Installation - provide the Target with the OS package.
// 2) Activation - activate an installed OS package.
// 3) Verification - verify that the Activation was successful.
//
// Dual Supervisor Target is supported, where the above process is executed once
// for each Supervisor.
//
// Note that certain platforms may have particular approaches to upgrade the
// firmware of specific components, eg., power supply units, etc.. In addition,
// platforms may have processes to apply patches to the running OS. Handling
// these exceptions introduces extra complexities. For Targets that implement
// this service, component firmware upgrade or OS patching MUST be embedded
// within an OS upgrade.
service OS {
    // Install transfers an OS package into the Target. No concurrent Install RPCs
    // MUST be allowed to the same Target.
    //
    // The OS package file format is platform dependent. The platform MUST
    // validate that the OS package that is supplied is valid and bootable. This
    // SHOULD include a hash check against a known good hash. It is recommended
    // that the hash is embedded in the OS package.
    //
    // The Target manages its own persistent storage, and OS installation process.
    // It stores a set of distinct OS packages, and always proactively frees up
    // space for incoming new OS packages. It is guaranteed that the Target always
    // has enough space for a valid incoming OS package. The currently running OS
    // packages MUST never be removed. The Client MUST expect that the last
    // successfully installed package is available.
    //
    // The Install RPC allows the Client to specify the OS package version. If
    // the Target already has an OS package with the same version then there is no
    // need to transfer the OS package to the Target. If the Target does not have
    // an OS package with the same version, then the OS package is copied.
    //
    // Scenario 1 - When the Target already has the OS package:

```

```

//
//      Client :-----|-----> Target
//      TransferRequest -->
//                  <-- [Validated|InstallError]
//
//
// Scenario 2 - When the Target does not have the OS package:
//
//      Client :-----|-----> Target
//      TransferRequest -->
//                  <-- [TransferReady|InstallError]
//      transfer_content -->
//                  ...
//                  <-- [TransferProgress|InstallError]
//                  ...
//      TransferEnd -->
//                  <-- [Validated|InstallError]
//
// On a dual Supervisor Target, only the Active Supervisor runs this gNOI
// Service. The Install RPC applies to the Active Supervisor unless
// InstallRequest->TransferRequest->standby_supervisor is set, in which case
// it applies to the Standby Supervisor. One Install RPC is required for each
// Supervisor. The Supervisor order of package installation MUST not be fixed.
//
// The Target MUST always attempt to copy the OS package between Supervisors
// first before accepting the transfer from the Client. The syncing progress
// is reported to the client with InstallResponse->SyncProgress messages.
//
// If a switchover is triggered during the Install RPC, the RPC MUST
// immediately abort with Error->type->UNEXPECTED_SWITCHOVER.
//
// Scenario 3 - When both Supervisors already have the OS package, regardless
// of the value in Start.standby_supervisor:
//
//      Client :-----|-----> Target
//      TransferRequest -->
//                  <-- [Validated|InstallError]
//
//
// Scenario 4 - When one of the Supervisors already has the OS package but the
// other Supervisor is the target of the Install:
//
//      Client :-----|-----> Target
//      TransferRequest -->
//                  <-- [SyncProgress|InstallError]
//                  ...
//                  <-- [Validated|InstallError]
//
//
// Scenario 5 - When neither of the two Supervisors has the OS package:
//
//      Client :-----|-----> Target
//      TransferRequest -->
//                  <-- [TransferReady|InstallError]
//      transfer_content -->
//                  ...
//                  <-- [TransferProgress|InstallError]
//                  ...
//      TransferEnd -->
//                  <-- [Validated|InstallError]
//
rpc Install(stream InstallRequest) returns (stream InstallResponse);

// Activate sets the requested OS version as the version which is used at the

```

```

// next reboot, and reboots the Target if the 'no_reboot' flag is not set.
// When booting the requested OS version fails, the Target recovers by
// booting the previously running OS package.
rpc Activate(ActivateRequest) returns (ActivateResponse);

// Verify checks the running OS version. This RPC may be called multiple times
// while the Target boots, until successful.
rpc Verify(VerifyRequest) returns (VerifyResponse);
}

message InstallRequest {
  oneof request {
    TransferRequest transfer_request = 1;
    bytes transfer_content = 2;
    TransferEnd transfer_end = 3;
  }
}

message TransferRequest {
  // The version string is a vendor defined string that identifies the OS
  // version. It is provided by the vendor and embedded in the OS package. This
  // value states the desired OS package version to transfer to the Target. If
  // the Target already has the OS package version it will reply with
  // InstallResponse->Validated. In the case that the target is a
  // single Supervisor device, or the partner Supervisor does not have the OS
  // image specified, it will respond with InstallResponse->TransferReady. In
  // this case, the client MUST subsequently transfer the image. In the case
  // that the image is available on the peer Supervisor of a dual Supervisor
  // system, it will respond with InstallResponse->SyncProgress. In this,
  // latter, case - the client does not need to transfer the OS image. This
  // value can also be set empty, in which case the OS package is forced
  // transferred to the Target. The Target MUST never validate that this value
  // matches the one in the InstallResponse->Validated message, that is the
  // Client's responsibility.
  string version = 1;

  // For a Target with dual Supervisors setting this flag instructs the Target
  // to perform the action on the Standby Supervisor.
  bool standby_supervisor = 2;
}

// The TransferEnd message is sent whenever the Client finishes transferring
// the OS package to the Target. At this point the Target MUST perform a general
// health check to the OS package. If the Target fails to parse the OS package
// it MUST immediately reply with an InstallError->type->PARSE_FAIL. If the
// integrity check of the OS package fails it MUST immediately reply with an
// InstallError->type->INTEGRITY_FAIL. If the identified OS version contained in
// the package is not compatible with the Target either because of the platform
// type or the running OS, it MUST immediately reply with an
// InstallError->type->INCOMPATIBLE. If the image is force transferred by
// omitting the InstallRequest->TransferRequest->version value, and the OS
// package is the same as the one running in the Target, the RPC MUST
// immediately abort and reply with an InstallError->type->INSTALL_RUN_PACKAGE.
message TransferEnd {
}

// The InstallResponse is used by the Target to inform the Client about the
// state of the Install RPC. At any stage of the process the Target can reply
// with an Error message which MUST terminate the stream.
message InstallResponse {
  oneof response {
    TransferReady transfer_ready = 1;
    TransferProgress transfer_progress = 2;
    SyncProgress sync_progress = 3;
  }
}

```



```
Validated validated = 4;
InstallError install_error = 5;
}
}

// The TransferReady message tells the Client that the Target is ready to accept
// the transfer of the OS package. At this stage the Target MUST have cleared
// enough space to accept the incoming OS package.
message TransferReady {
}

// The TransferProgress message is sent by the target asynchronously during a
// file transfer. The device SHOULD not respond to each input block received
// from the client, but rather determine reasonable intervals at which to send
// the message (e.g., 5MB).
message TransferProgress {
  // The number of bytes transferred.
  uint64 bytes_received = 1;
}

// The SyncProgress message signals the Client about the progress of
// transferring the OS package between Supervisors.
message SyncProgress {
  // The percentage that has transferred between Supervisors.
  uint32 percentage_transferred = 1;
}

// The Validated message asserts that the Target was able to parse the package
// and perform integrity checks to its contents.
message Validated {
  // The OS version string that identifies the OS version in the OS package.
  string version = 1;
  // Informational field that SHOULD be used for providing more details about
  // the OS package and its version. This MUST be strictly informational if
  // used, and can contain information such as build date, target platform,
  // developer, etc.
  string description = 2;
}

// The InstallError message MUST be sent by the Target to the Client whenever an
// issue occurs. The Target MUST immediately close the RPC without a gRPC error.
message InstallError {
  enum Type {
    // An unspecified error. Must use the detail value to describe the issue.
    UNSPECIFIED = 0;
    // The newly transferred package is not compatible with the Target platform.
    // The detail field MUST contain the detailed error message.
    INCOMPATIBLE = 1;
    // The OS package being transferred is larger than the available size the
    // Target provisioned. This is unexpected since the Target MUST clear disk
    // space for the new OS packages. The available space and the OS package
    // size MUST be guaranteed by the platform maker, therefore the most likely
    // cause of this error is that a wrong package is being transferred.
    TOO_LARGE = 2;
    // Used whenever the system is unable to parse the newly transferred
    // package, like reading the OS version or the integrity checksums.
    PARSE_FAIL = 3;
    // The transferred OS package fails integrity check.
    INTEGRITY_FAIL = 4;
    // Attempting to force transfer an OS package with the same version as the
    // currently running.
    INSTALL_RUN_PACKAGE = 5;
    // Another Install RPC to this Target is already in progress.
    INSTALL_IN_PROGRESS = 6;
  }
}
```

```

    // A switchover happened during the Install RPC.
    UNEXPECTED_SWITCHOVER = 7;
    // Failed to sync the transferred OS package to the standby Supervisor. The
    // detail value MUST have more information.
    SYNC_FAIL = 8;
  }
  Type type = 1;
  string detail = 2;
}

// The ActivateRequest is sent by the Client to the Target to initiate a change
// in the next bootable OS version that is to be used on the Target.
message ActivateRequest {
  // The version that is required to be activated and optionally immediattely
  // booted.
  string version = 1;
  // For dual Supervisors setting this flag instructs the Target to perform the
  // action on the Standby Supervisor.
  bool standby_supervisor = 2;
  // If set to 'False' the Target will initiate the reboot process immediattely
  // after changing the next bootable OS version.
  // If set to 'True' a separate action to reboot the Target and start using
  // the activated OS version is required. This action CAN be executing
  // the gNOI.system.Reboot() RPC.
  bool no_reboot = 3;
}

// The ActivateResponse is sent from the Target to the Client in response to the
// Activate RPC. It indicates the success of making the OS package version
// active.
message ActivateResponse {
  oneof response {
    ActivateOK activate_ok = 1;
    ActivateError activate_error = 2;
  }
}

// If the Target is already running the requested version in ActivateRequest,
// then it replies with ActivateOK. If the Target has the OS package version
// requested in ActivateRequest then it replies with ActivateOK and proceeds to
// boot. In a Target with dual Supervisor, performing this RPC on the Active
// Supervisor triggers a switchover before booting the (old)Active Supervisor.
// The Target should always perform a switchover with the least impact possible
// to forwarding.
message ActivateOK {
}

message ActivateError {
  enum Type {
    // An unspecified error. Must use the detail value to describe the issue.
    UNSPECIFIED = 0;
    // There is no OS package with the version requested for activation. This is
    // also used for an empty version string.
    NON_EXISTENT_VERSION = 1;
  }
  Type type = 1;
  string detail = 2;
}

message VerifyRequest {
}

message VerifyResponse {
  // The OS version currently running.

```

```
string version = 1;
// Informational message describing fail details of the last boot. This MUST
// be set when a newly transferred OS fails to boot and the system falls back
// to the previously running OS version. It MUST be cleared whenever the
// systems successfully boots the activated OS version.
string activation_fail_message = 2;

VerifyStandby verify_standby = 3;
}

message VerifyStandby {
  oneof state {
    StandbyState standby_state = 1;
    StandbyResponse verify_response = 2;
  }
}

message StandbyState {
  enum State {
    UNSPECIFIED = 0;
    // The Target does not support dual Supervisors.
    UNSUPPORTED = 1;
    // Standby Supervisor is supported but does not exist.
    NON_EXISTENT = 2;
    // Standby Supervisor is supported but is not available, eg.: rebooting.
    UNAVAILABLE = 3;
  }
  State state = 1;
}

message StandbyResponse {
  // Standby Supervisor ID, usually the slot number.
  string id = 1;
  string version = 2;
  string activation_fail_message = 3;
}
```



OpenConfig gNOI Protocols

[file.proto](#) on page 21

[interface.proto](#) on page 24

OpenConfig gNOI copyright statements can be found at <https://github.com/openconfig/gnoi>.

The following topics describe the OpenConfig gNOI protocols that are supported by Extreme 9920 software and Extreme 9920.

file.proto

Defines gNOI protocol for file.

Table 8: File remote procedure calls

RPC	Purpose
TransferToRemote	Transfers the contents of a file from the target to a remote location.
Remove	Removes the specified file from the target.
Put	Streams data into a file on the target.
Stat	Returns metadata about a file on the target.
GET	Retrieves a snapshot of data from the target.

```
//
// Copyright 2017 Google Inc. All Rights Reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
//

syntax = "proto3";

package gnoi.file;

import "github.com/openconfig/gnoi/common/common.proto";
import "github.com/openconfig/gnoi/types/types.proto";

option go_package = "github.com/openconfig/gnoi/file";

option (types.gnoi_version) = "0.1.0";

service File {
    // Get reads and streams the contents of a file from the target.
    // The file is streamed by sequential messages, each containing up to
    // 64KB of data. A final message is sent prior to closing the stream
    // that contains the hash of the data sent. An error is returned
    // if the file does not exist or there was an error reading the file.
    rpc Get(GetRequest) returns (stream GetResponse) {}

    // TransferToRemote transfers the contents of a file from the target to a
    // specified remote location. The response contains the hash of the data
    // transferred. An error is returned if the file does not exist, the file
    // transfer fails, or if there was an error reading the file. This is a
    // blocking call until the file transfer is complete.
    rpc TransferToRemote(TransferToRemoteRequest)
        returns (TransferToRemoteResponse) {}

    // Put streams data into a file on the target. The file is sent in
```

```

// sequential messages, each message containing up to 64KB of data. A final
// message must be sent that includes the hash of the data sent. An
// error is returned if the location does not exist or there is an error
// writing the data. If no checksum is received, the target must assume the
// operation is incomplete and remove the partially transmitted file. The
// target should initially write the file to a temporary location so a failure
// does not destroy the original file.
rpc Put(stream PutRequest) returns (PutResponse) {}

// Stat returns metadata about a file on the target. An error is returned
// if the file does not exist or there is an error in accessing the metadata.
rpc Stat(StatRequest) returns (StatResponse) {}

// Remove removes the specified file from the target. An error is
// returned if the file does not exist, is a directory, or the remove
// operation encounters an error (e.g., permission denied).
rpc Remove(RemoveRequest) returns (RemoveResponse) {}
}

// A PutRequest is used to send data to be written on a file on the target.
//
// The initial message contains an Open message. The Open message contains
// information name of the file and the file's permissions.
//
// The remote_file must be an absolute path. If remote_file already exists on
// the target, it is overwritten, otherwise it is created. If the path to
// remote_file doesn't exist it will be created.
//
// The contents to be written are streamed through multiple messages using the
// contents field. Each message may contain up to 64KB of data.
//
// The final message of the RPC contains the hash of the file contents.
message PutRequest {
  message Details {
    string remote_file = 1;
    // Permissions are represented as the octal format of standard UNIX
    // file permissions.
    // ex. 775: user read/write/execute, group read/write/execute,
    // global read/execute.
    uint32 permissions = 2;
  }
  oneof request {
    Details open = 1;
    bytes contents = 2;
    types.HashType hash = 3; // hash of the file.
  }
}

message PutResponse {
}

// A GetRequest specifies the remote_file to be streamed back
// to the caller. The remote_file must be an absolute path to an
// existing file.
message GetRequest {
  string remote_file = 1;
}

// A GetResponse either contains the next set of bytes read from the
// file or, as the last message, the hash of the data.
message GetResponse {
  oneof response {
    bytes contents = 1;
  }
}

```

```
    types.HashType hash = 2; // hash of the file.
  }
}

// A TransferToRemoteRequest specifies the local path to transfer to and the
// details on where to transfer the data from. The local_path must be an
// absolute path to the file.
message TransferToRemoteRequest {
  string local_path = 1;

  // Details to download the remote_file being requested to a remote location.
  common.RemoteDownload remote_download = 2;
}

// A TransferToRemoteResponse contains the hash of the data transferred.
message TransferToRemoteResponse {
  types.HashType hash = 1; // hash of the file.
}

// StatRequest will list files at the provided path.
message StatRequest {
  string path = 1;
}

// StatResponse contains list of stat info of the provided path.
message StatResponse {
  repeated StatInfo stats = 1;
}

// StatInfo provides a file system information about a particular path.
message StatInfo {
  string path = 1;
  uint64 last_modified = 2; // Nanoseconds since epoch.
  // Permissions are represented as the octal format of standard UNIX
  // file permissions.
  // ex. 775: user read/write/execute, group read/write/execute,
  // global read/execute.
  uint32 permissions = 3;
  uint64 size = 4;
  // Default file creation mask. Represented as the octal format of
  // standard UNIX mask.
  uint32 umask = 5;
}

// A RemoveRequest specifies a file to be removed from the target.
message RemoveRequest {
  string remote_file = 1;
}

message RemoveResponse {
}
```

interface.proto

Defines a gNOI protocol AP for interface.

Table 9: Interface remote procedure calls

RPC	Purpose
ClearInterfaceCounters	Reset counters for the specified interface.

```

syntax = "proto3";

package gnoi.interface;

import "github.com/openconfig/gnoi/types/types.proto";

option go_package = "github.com/openconfig/gnoi/interface";

option (types.gnoi_version) = "0.1.0";

service Interface {
  // SetLoopbackMode is used to set the mode of loopback on a interface.
  rpc SetLoopbackMode(SetLoopbackModeRequest)
    returns (SetLoopbackModeResponse) {}

  // GetLoopbackMode is used to get the mode of loopback on a interface.
  rpc GetLoopbackMode(GetLoopbackModeRequest)
    returns (GetLoopbackModeResponse) {}

  // ClearInterfaceCounters will reset the counters for the provided interface.
  rpc ClearInterfaceCounters(ClearInterfaceCountersRequest)
    returns (ClearInterfaceCountersResponse) {}
}

// SetLoopbackModeRequest requests the provide interface be have its loopback
// mode set to the specified mode. The mode may be vendor specific. For example,
// on a transport device, available modes are "none", "mac", "phy",
// "phy_remote", "framer_facility", and "framer_terminal".
message SetLoopbackModeRequest {
  types.Path interface = 1;
  string mode = 2;
}

message SetLoopbackModeResponse {
}

message GetLoopbackModeRequest {
  types.Path interface = 1;
}

message GetLoopbackModeResponse {
  string mode = 1;
}

message ClearInterfaceCountersRequest {
  repeated types.Path interface = 1;
}

message ClearInterfaceCountersResponse {
}

```