NSP
Network Services Platform
Network Functions Manager - Packet (NFM-P)
Release 23.11

XML API Developer Guide

3HE-19017-AAAC-TQZZA
Issue 1
December 2023

Legal notice

Nokia is committed to diversity and inclusion. We are continuously reviewing our customer documentation and consulting with standards bodies to ensure that terminology is inclusive and aligned with the industry. Our future customer documentation will be updated accordingly.

Contents

About this document

Purpose

The *NSP NFM-P XML API Developer Guide* provides information about developing OSS applications, including setting up the development environment, understanding the XML API schemas, and typical operational scenarios, such as receiving a real-time JMS event stream.

Scope

The scope of this document is limited to the NFM-P application. Many configuration, monitoring, and assurance functions that can be accomplished from the NFM-P Java GUI are also delivered in NSP. Readers of this NFM-P guide should familiarize themselves with the capabilities of the NSP, which often offers more efficient and sophisticated features for network and service management.

Safety information

For your safety, this document contains safety statements. Safety statements are given at points where risks of damage to personnel, equipment, and operation may exist. Failure to follow the directions in a safety statement may result in serious consequences.

Document support

Customer documentation and product support URLs:

- Documentation Center
- Technical support

How to comment

Please send your feedback to documentation.feedback@nokia.com.

1 Product and support

1.1 NFM-P product overview

1.1.1 Overview

The NFM-P is a network management system that is designed to manage proprietary devices. The NFM-P also provides limited management of other third-party devices, which are called generic NEs.

The NFM-P network management key features include:

- alarm correlation up to the service level
- service and routing provisioning using policies and profiles to reduce the repetition of effort
- inventory reporting at the equipment, service, and customer levels using filtered views that can be saved as report files
- network performance and accounting data collection on a per-service or per-port basis using a flat-rate, destination, or usage schema
- troubleshooting at the service level, which includes viewing the associations between services and entities such as customers, equipment, and transport tunnels
- OSS interfaces that provide access to the NFM-P functions from other systems

The NFM-P software architecture is based on industry standards. See the *NSP Architecture Guide* for more information.

The main NFM-P system components are the following:

- a Java-based main server and an Oracle relational database in a standalone or redundant configuration
- auxiliary servers
- Java-based NFM-P GUI clients
- platform-independent OSS clients

See the *NSP Architecture Guide* for information about the NFM-P system structure and components.

1.2 XML API

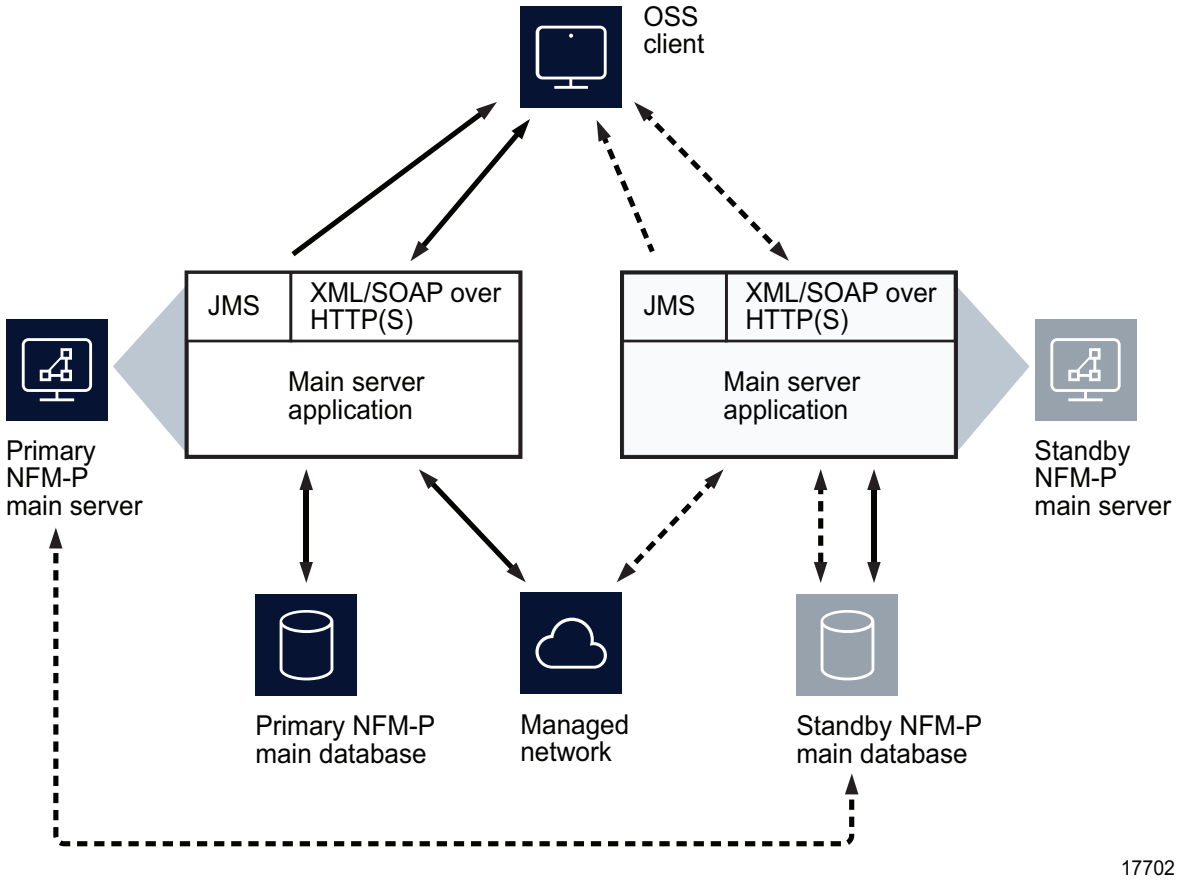
1.2.1 Overview

The XML API is an NFM-P interface that provides an XML interface to the NFM-P through which an OSS client application can perform the following tasks:

- configure or retrieve NFM-P network management information
- manipulate managed objects
- receive event notifications from the NFM-P server using a persistent or non-persistent JMS connection

The following figure shows the architecture of the XML API, which includes a JMS connection for the collection of near-real-time alarms and events.

Figure 1-1 XML API architecture



An XML API client sends an XML-encoded, SOAP-enabled message to a main server HTTP or HTTPS port. An XML application on the main server station parses the SOAP XML message and forwards the request to the NFM-P server for processing.

An HTTPS connection requires TLS encryption. See the *NSP Planning Guide* for the TLS and HTTPS port assignments, and the *NSP Installation and Upgrade Guide* for TLS configuration information.

The XML API is installed with the core NFM-P software components. To use the XML API, you must perform the following configuration tasks.

- Enter an NFM-P license key that activates the XML API.
- Use a user group that is assigned the OSS management scope of command role.

Note: You can create a scope of command profile to group the OSS management scope of command role with additional roles. See the section on user account and group management in the *NSP System Administrator Guide* for more information about user groups and the associated scope of command roles and permissions.

1.2.2 XML API client

The XML API client implementation can range from a CLI to a third-party application. The format for the client user interface varies with the function of the client application; for example, rolling up statistics into a third-party billing application. The XML API clients function the same way regardless of the front-end implementation: XML scripts are posted to the NFM-P server using an HTTP client. See Chapter 3, “Communicating with the NFM-P” for more information.

1.3 About this guide

1.3.1 Overview

The *NSP NFM-P XML API Developer Guide* provides information to help OSS developers create applications that perform NFM-P functions. It is part of the Developer documentation, which also includes the XML API Reference and Schema Reference.

This guide covers the following topics:

- An overview of the XML API, including the following:
  - an NFM-P and XML API high-level overview
  - redundant configurations and supported devices and technologies
  - open interfaces professional support
- Information about how an OSS via the XML API can communicate with the NFM-P and the underlying network, including the following:
  - communication with the NFM-P server
  - event monitoring
  - XML requests
  - interworking with other systems
- Information about the XML API object model that includes the following:
  - XML package descriptions, object properties, and value definitions and relationships
  - XML Reference and XML Schema type and method definitions and descriptions
  - XML message structure and request/response operation
- Information that a developer can use to help with the development of applications for the following OSS domains:
  - Fault management—provides information for developing an event and alarm management system
  - Inventory management—provides information about inventory retrieval
  - Accounting and Performance Management
  - OAM diagnostic tests
- Information about how an OSS can be used to make changes to network objects, which can involve creating, deleting, and modifying the objects in the following areas:
  - general configuration using generic method that applies to all areas

  - equipment configuration
  - routing protocol configuration
  - customer and residential subscriber configuration
  - policy configuration
  - service configuration
  - XML service template configuration
  - script management configuration
- Information about other network management products, such as:
  - CPAM
- Appendices containing information about the following:
  - best practices
  - troubleshooting client OSS application problems

The following guides, documents, or files are also relevant to an OSS developer:

- When setting up the NFM-P in your lab environment:
  - *NSP Planning Guide*—platform support information
  - *NSP Installation and Upgrade Guide*—installation information
- When starting to use the NFM-P:
  - *NSP NFM-P Classic Management User Guide*
- When developing an OSS application:
  - XML API Reference—contains javadoc-style documentation of the object model, essential for writing XML API requests
  - Schema Reference—contains XSD schema definitions for the object model
  - XML API SDK Sample Code Navigator—includes a library of XML scripts that provides configuration and network management request samples, as well as the corresponding responses, to help developers with OSS integration. The samples in this document can be used as a base on which to build OSS requests.
  - JMS Test Example Code—contains example code for writing JMS subscribers

  See the Network Developer Portal for more information.
- Other files:
  - MIBs—files are on each main server in the /opt/nsp/nfmp/server/nms/web/tomcat/work/Catalina/localhost/help/eclipse/plugins/ALU_SAM_MIB directory

1.4 Redundant network configurations

1.4.1 Overview

The NFM-P supports database and main server redundancy, which provides network visibility in the event of a failure on one or more components. If a primary component fails, the standby component automatically takes control. You can also perform a manual switch to test redundancy or to prepare for a network upgrade.

The NFM-P supports collocated and distributed redundant deployment. In a collocated system, an NFM-P main server and database are on the same station. In a distributed deployment, the NFM-P

server and the database are on separate stations. The servers and databases are separate entities, regardless of the physical or geographical deployment.

You require the following information to configure redundant OSS connections:

- the IP addresses of the primary and standby main servers
- which main server is currently primary

You can use any of the following methods to monitor the connection status between an OSS client and a main server:

- JMS exceptions, to monitor client connections to the JMS server
- monitoring near-real-time events using a JMS client connection
- XML API ping to monitor the HTTP connection to the NFM-P web server

See 3.3 “OSS client interfaces” (p. 23) for more information about monitoring the connection status between the OSS client and a main server.

See Chapter 4, “Event monitoring using JMS” for more information about JMS and redundancy.

1.5 Supported devices and technologies

1.5.1 Overview

For core network devices, the NFM-P supports the current software release and the three immediately previous major releases. Older device releases are not supported.

For devices other than core-network devices, the NFM-P release support varies. See the *NSP NFM-P Network Element Compatibility Guide* for specific device support information.

Note: The NFM-P functions and features vary for the supported devices. This guide does not describe the variations in support, or device-specific procedures for specific implementations. See the *NSP NFM-P Classic Management User Guide* for this information.

The *NSP NFM-P Network Element Compatibility Guide* lists the device releases and functions that the NFM-P supports.

See the device user documentation for information about device functions, parameters, and CLI commands that are outside the scope of the NFM-P documentation. Contact Nokia technical support for specific network or facility considerations.

1.5.2 Supported technologies

The following technologies are supported:

- XML
- SOAP
- HTTP(S)
- Java
- JMS
- JBoss

See the *NSP Architecture Guide* for more information about the standards compliance for these technologies.

1.6 Before you begin

1.6.1 Overview

Nokia recommends that you do the following before you start to work in the XML development environment.

- Review the requirements for license key configuration and platform size considerations. See the *NSP Planning Guide* for more information.
- Preconfigure the device that you want the NFM-P to manage. See the device documentation, or procedures about network management configuration in the *NSP System Administrator Guide* for more information.
- Back up the NFM-P database. Nokia recommends a database backup before you implement any major, network-wide configuration changes. A database backup utility is available in the NFM-P client GUI and through a CLI. See the section on backing up the main database in the *NSP System Administrator Guide* for more information.
- Understand OSS interfaces and standard XML schema constructions

Contact your Nokia OIPS representative for development support. See 1.7 “Open interfaces professional support” (p. 17) for more information.

1.7 Open interfaces professional support

1.7.1 Overview

Nokia provides developer support for third-party application integration with the XML API. The Nokia OIPS portfolio provides OSS developers with network management integration solutions for the NFM-P and CPAM. OSS integration initiatives include project review, design consultation, development support, and training for integration projects.

Key elements of each service offering include:

- Design consultation and high-level project reviews. Extensive software design consultation provides architectural review of a proposed design and guidance on whether the design methodology is suited to the proposed application given Nokia experience with the platform and interface.
- Software development support and technical guidance
- Sample code that provide examples of configuration and management concepts, to facilitate rapid OSS integration
- OSS product training that minimizes project down time and optimizes development efficiency
- OSS Product Certification support to ISVs identified as Connected Partners, involving extensive interoperability testing of OSS applications with the NFM-P and CPAM.

Nokia OIPS can customize the support content and contract length to meet the unique development requirements of service providers.

1.8 Recommended upgrade path

1.8.1 Overview

Nokia recommends the following upgrade path to ensure continuous availability of the network management ecosystem.

- Disconnect the OSS from the NFM-P.
- Upgrade the OSS.

  Note:

  The upgraded OSS must be backward-compatible with the current NFM-P release, including backward compatibility with the existing object model, JMS, and samOss.jar.

  For more information about object model compatibility, see 8.8 “XML schema changes” (p. 101) .

  For more information about compatibility with JMS, see 8.9 “JMS changes” (p. 102) .

  For more information about compatibility with samOss.jar, see 4.2 “JMS connections” (p. 33) .
- Upgrade the NFM-P.
- Install the new samOss.jar on the OSS.
- Connect and synchronize the OSS with the NFM-P.

  For information about the recommended startup method for an OSS client with a durable JMS subscription, see A.2 “Recommended durable JMS client operation” (p. 327) .
- Upgrade the network devices.

2 Schema and FDN changes

2.1 XML API schema changes

2.1.1 Overview

The XML API Reference includes one schema changes web page for each major NFM-P release. Each page lists the object model differences between consecutive revisions of the major release.

At the head of the page are lists of the high-level product and package changes, and a summary table of links to the class and type change lists for each package.

You can use the schema change information to:

- ensure that existing OSS applications correctly reference the current XML API object model
- track object-model changes as OSS applications are developed for different releases

See 8.10 “To view release-specific XML schema changes” (p. 102) for information about how to view the XML API schema changes.

2.2 NFM-P object FDN changes

2.2.1 Overview

The NFM-P object FDNs may change between NFM-P releases. An XML API client that uses FDNs to reference NFM-P objects must update the FDNs accordingly after an NFM-P system upgrade. The FDNs that change between NFM-P releases are listed in files in the NFM-P database installation directory. There is one log file for each consecutive release transition in which an FDN changes. For example, the following FDN mapping files are created in the /opt/nsp/nfmp/db/install directory, where *date* is the date of the upgrade:

- FDN_Mapping_11_0_R1_to_11_0_R2.*date*.log
- FDN_Mapping_11_0_R2_to_11_0_R3.*date*.log
- FDN_Mapping_11_0_R3_to_11_0_R4.*date*.log
- FDN_Mapping_11_0_R4_to_11_0_R5.*date*.log
- FDN_Mapping_11_0_R5_to_11_0_R6.*date*.log
- FDN_Mapping_11_0_R6_to_11_0_R7.*date*.log
- FDN_Mapping_11_0_R7_to_12_0_R1.*date*.log

Note: A system upgrade automatically migrates data from the starting release to the target release without the need to upgrade to each intermediate release.

Figure 2-1 Sample FDN mapping log file

```
<?xml version='1.0' ?>
<version from="A.0.R1" to="B.0.R1">
<old Fdn="network:142.165.76.50:router-1:ospf:areaSite-0.0.0.11:interface-204.83.
242.2.0" class="ospf.Interface">
        <new Fdn="network:142.165.76.50:router-1:ospf-v2:areaSite-0.0.0.11:
interface-204.83.242.2.0" class="ospf.Interface"/>
```

```
    </old>
    <old
Fdn="network:142.165.76.50:router-1:ospf:areaSite-0.0.0.11:interface-204.83.242.2.
0:neighbor-204.83.242.1.0" class="ospf.Neighbor">
        <new
Fdn="network:142.165.76.50:router-1:ospf-v2:areaSite-0.0.0.11:interface-204.83.
242.2.0:neighbor-204.83.242.1.0" class="ospf.Neighbor"/>
    </old>
    <old
Fdn="network:142.165.76.50:router-1:ospf:areaSite-0.0.0.11:interface-204.83.242.2.
0:md 5-1" class="ospf.Md5Key">
        <new
Fdn="network:142.165.76.50:router-1:ospf-v2:areaSite-0.0.0.11:interface-204.83.
242.2.0:md5-1" class="ospf.Md5Key"/>
    </old>
    <old
Fdn="network:142.165.76.50:router-1:ospf:areaSite-0.0.0.11:interface-204.83.242.
9.0" class="ospf.Interface">
        <new
Fdn="network:142.165.76.50:router-1:ospf-v2:areaSite-0.0.0.11:interface-204.83.
242.9.0" class="ospf.Interface"/>
    </old>
    <old
Fdn="network:142.165.76.50:router-1:ospf:areaSite-0.0.0.11:interface-204.83.242.9.
0:md5-1" class="ospf.Md5Key">
        <new
Fdn="network:142.165.76.50:router-1:ospf-v2:areaSite-0.0.0.11:interface-204.83.
242.9.0:md5-1" class="ospf.Md5Key"/>
    </old>
</version>
```

3 Communicating with the NFM-P

3.1 Overview

3.1.1 Overview

The following communication mechanisms are available to NFM-P OSS clients:

- NFM-P XML API and JMS event stream
- NSP REST API and Kafka notification service

The XML API receives SOAP XML requests for configuring and managing NFM-P system and network objects. The XML JMS event stream allows OSS clients to receive near-real-time notification of NFM-P system and managed-network events. The *NSP NFM-P XML API Developer Guide* focuses primarily on OSS application integration with the XML API and JMS mechanisms.

By subscribing to NSP REST APIs, an NFM-P OSS client can configure and manage objects, and receive near-real-time Kafka event notifications. See the Network Developer Portal for information.

Note: The NSP REST APIs also allow OSS clients to integrate with other NSP components.

A developer who creates an OSS application for the XML API requires knowledge of the following:

- standard XML schema constructions
- OSS interfaces
- managed devices
- NFM-P functions

3.2 NFM-P system components

3.2.1 Overview

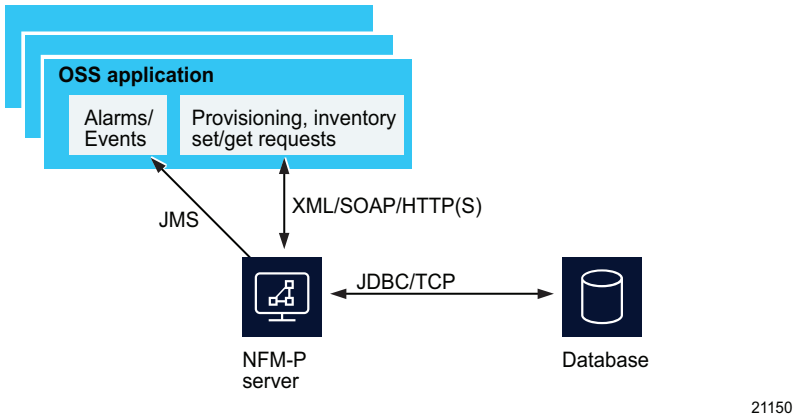
An NFM-P system has multiple components. The components are briefly described in 1.1 “NFM-P product overview” (p. 13). For a comprehensive description of NFM-P system components and relationships, see the Component communication information in the *NSP Architecture Guide*.

3.3 OSS client interfaces

3.3.1 Overview

The following figure shows the OSS client interfaces, which provide access to the NFM-P. The OSS clients use the interfaces to send configuration and information requests, and monitor network events.

Figure 3-1 OSS client interfaces



3.3.2 Event monitoring using JMS

Clients can monitor events in the NFM-P and the network by subscribing to event streams. This event-driven function allows a client to maintain a near-real-time view of events in the network. However, the function is unidirectional and does not allow clients to initiate changes in the NFM-P or the network.

To use JMS event streams to monitor events, the OSS developers need to create a JMS client. The client must subscribe to one or more event streams, which can be used to monitor events. Some applications are:

- monitoring alarms
- maintaining up-to-date inventory information by monitoring configuration changes in the network

See Chapter 4, “Event monitoring using JMS” for more information about JMS and event monitoring.

3.3.3 Request and response using the XML API

Clients can request information and initiate changes in the NFM-P and in the managed network using the XML API. This function is bidirectional. Clients send requests and receive responses that contain the information requested or the results of an operation.

The XML API uses HTTP. The OSS developers must create a client that can send HTTP requests and receive responses. Some applications are:

- retrieving network inventory information
- changing the network configuration
- provisioning

The XML API supports HTTP 1.1. Except for the following RFC 2616 standard HTTP request header fields, the other HTTP header fields are not to be set:

```
Content-Type: text/xml; charset = ISO-8859-1
```

```
User Agent: <Client User Agent>
Host: <NFM-P main server>
Content-Length: <Size of the request body>
```

The XML API does not support non-standard HTTP headers.

See Chapter 5, “XML requests” for more information about using XML API .

3.3.4 XML and SOAP

The JMS events and HTTP requests and responses are written in XML using the SOAP standard. XML is a standard format used to describe data. The Schema Reference is defined in XML style sheets, which are distributed with the documentation. API requests and responses, and JMS events are wrapped in SOAP envelopes.

For more information, see the related W3C specifications and other related material available from technical publishers and the Internet. Some suggested resources are:

- http://www.w3.org/XML
- http://www.w3.org/TR/soap
- Ray, Eric T. Learning XML (2nd Ed). O’Reilly, 2003

3.4 Secure communication

3.4.1 Overview

The XML API provides the following functions for secure communication between the OSS clients and the NFM-P:

- MD5-hashed passwords for requests
- TLS message encryption for JMS events
- HTTPS for XML requests and responses

Note: MD5-hashed passwords do not secure the communication channel.

Note: Nokia recommends enabling TLS for secure communication.

TLS is enabled by default on NFM-P server and client interfaces to provide secure communication between NFM-P components. TLS requires a security certificate that is shared among members of a network domain using TLS keystore and truststore files. The NFM-P supports using CA-supplied or self-signed certificates.

For backward compatibility, the NFM-P supports non-secure communication with OSS clients. To support non-secure clients, you must disable TLS on the XML API.

Note: Disabling TLS on the XML API disables TLS for all clients that use the XML API, and for all NFM-P GUI clients. Browser-based clients are unaffected, and must use HTTPS for application access.

See the *NSP Installation and Upgrade Guide* for information about configuring NFM-P TLS.

3.4.2 TLS communication

When establishing TLS communication from an OSS system, an NFM-P main server selects the highest security cipher that the OSS client presents. The specific set of supported security ciphers and algorithms may vary, depending on the OSS client and server software components and certificates.

You can use an NFM-P tool to specify which TLS versions and ciphers the NFM-P supports. See the procedure to update the supported NFM-P TLS versions and ciphers in the *NSP System Administrator Guide* for information about using the tool.

3.5 TLS message encryption

3.5.1 Overview

OSS clients can receive secure JMS XML messages using TLS encryption. To enable TLS for an OSS client, you must import a TLS certificate from an NFM-P main server to a TLS truststore file on the OSS client station. You must also configure Java on the OSS client station by specifying the location of the truststore file on the OSS client station. For more information, see the TLS configuration and management information in the *NSP Installation and Upgrade Guide*.

See 4.12 “To compile and connect an NFM-P JMS client” (p. 62) and the Java documentation for information about configuring Java to use TLS.

The supported Java version may change between NFM-P releases. A difference in the major or minor Java version can affect operation, for example:

- TLS 1.2 is the default security mechanism in Java 8.
- In JDK 8u31, the SSLv3 protocol is deactivated.

For details about changes to the Java Runtime Environment, see the Java release notes and security documentation.

To determine the NFM-P Java version, see 4.15 “To determine the Java version” (p. 65).

3.5.2 HTTPS communication with XML API

The XML API can send requests and receive responses using HTTPS, which requires TLS encryption. See the *NSP Planning Guide* for the system requirements associated with TLS and HTTPS. See the *NSP Installation and Upgrade Guide* for information about implementing TLS in an NFM-P system.

3.5.3 Session management

Effective session management requires the AAA functions. Authentication is the verification of a user identification and password. Authorization is the assignment of different levels of access permissions to users. Accounting is the recording of user actions. An NFM-P operator can configure AAA functions using the local security capability of the NFM-P server, a third-party authentication server, or a combination of local and third-party mechanisms.

- Local authentication on the NFM-P server is provided by a local user database and security scheme to verify login attempts and assign permission levels for command execution.

- The supported third-party authentication servers are RADIUS and TACACS+. These products run on their own platforms, with their own user lists and administration processes.

OSS user and group management

An NFM-P administrator can use a GUI or OSS client to manage user and group privileges, sessions, and authentication. Using scope of command and span of control profiles, an administrator can specify which functions are available to a user or group, and restrict the objects that users can view or configure. See the section on NFM-P user security in the *NSP System Administrator Guide* for more information.

You can configure the following OSS properties of a user account or user group; if the user and group settings differ, the user settings override the group settings:

- maximum number of allowed OSS sessions per user
- priority assigned to OSS requests
- OSS request timeout

When the number of concurrent sessions reaches the system maximum, the NFM-P uses the priority values of the outstanding requests to determine which to process next.

Note: After an upgrade, the NFM-P assigns the default priority value to each existing OSS user and user group.

An OSS request timeout value applies to a request that the NFM-P has accepted but not processed. If the request processing does not begin before the timeout period elapses, the NFM-P rejects the request and returns an error.

Client sessions

All client sessions have username and password protection.

- Each OSS client XML/SOAP message is individually authenticated using cached information from an authorization server.
- JMS messages are protected by the user name and password for the JMS connection.

You can use an MD5-hashed or a clear text password to access the NFM-P server.

MD5-hashed password format

Nokia recommends that you send the password for a request in an MD5-hashed format. You can use the md5hash password utility to generate MD5-hashed passwords to access the NFM-P server. See 3.6 “To generate an MD5-hashed password for NFM-P main server access” (p. 28) for more information about how to generate an MD5-hashed password for NFM-P access.

Note: MD5-hashed passwords do not secure the communication channel.

Note: Nokia recommends enabling TLS for secure communication. See the *NSP Installation and Upgrade Guide* for TLS configuration and management information.

Clear-text password format

OSS users can access the NFM-P using a clear-text password. A clear-text password provides no security unless HTTPS is used, but ensures greater compatibility with RADIUS and TACACS+. Nokia recommends using a clear-text password only when:

- the XML API uses an external mechanism for password authentication, for example, RADIUS or TACACS

  and

- a dedicated HTTPS channel is used for all XML/SOAP requests

See Table 9-1, “SOAP message type details” (p. 106) for more information about the SOAP XML message structure and format. The following figure shows an example of the clear-text password format.

Figure 3-2 Clear-text password format

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP:Header>
        <header xmlns="xmlapi_1.0">
            <security>
                <user>username</user>
                <password hashed="false">password</password>
            </security>
            <requestID>XML_API_client@n</requestID>
        </header>
    </SOAP:Header>
    <SOAP:Body>
        <ping xmlns="xmlapi_1.0"/>
    </SOAP:Body>
</SOAP:Envelope>
```

3.6 To generate an MD5-hashed password for NFM-P main server access

3.6.1 Steps

1

Log on to an NFM-P main server station as the nsp user.

2

Open a console window.

3

Enter the following:

```
bash$ cd /opt/nsp/nfmp/server/nms/bin ↵
```

4

Enter the following to generate an MD5-hashed password:

```
bash$ ./md5hash.bash password ↵
```

where *password* is the password string to convert to an MD5-hashed password

For example, the MD5-hashed password for the word hello is the following:

5d41402abc4b2a76b9719d11017c592

5

Add the MD5-hashed password to the password field of the SOAP header for the specified user account.

END OF STEPS

3.7 XML API security features controlled through the NFM-P client GUI

3.7.1 Overview

A system administrator can use the security management forms in the NFM-P client GUI to control the following security-related features for XML API:

- User account privileges
- Session monitoring
- Session shutdown
- SNMPv3 for secure NE polling
- Radius and TACAS+ authentication for NFM-P users
- Session logs of NFM-P client GUI and OSS activity; the XML API client ID must use the JMS client ID format described in 4.5 “JMS subscriptions” (p. 35) to be uniquely identified as a XML API session.

You must create a scope of command profile to group the OSS Management scope of command role with additional roles. Otherwise, the NFM-P returns the SOAP exception message shown in the following figure :

Figure 3-3 SOAP exception message, insufficient privileges

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <requestID>XML_API_client@n</requestID>
         <requestTime>Sep 22, 2015 3:13:08 PM</requestTime>
         <responseTime>Sep 22, 2015 3:13:08 PM</responseTime>
      </header>
   </SOAP:Header>
   <SOAP:Fault>
```

```
      <faultcode>SOAP:Client</faultcode>
      <faultstring>[security] Users require OSS Management privileges to use XML
API.</faultstring>
      <faultactor>XmlApi</faultactor>
      <detail>
         <requestID>XML_API_client@n</requestID>
      </detail>
   </SOAP:Fault>
</SOAP:Envelope>
```

See the chapter on NFM-P user security in the *NSP System Administrator Guide* for information about managing NFM-P user security.

If a user supplies an incorrect username or password, or if the username or password is missing, the NFM-P returns the SOAP exception message shown in the following figure:

Figure 3-4 SOAP exception message, login failure

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <requestID>XML_API_client@n</requestID>
         <requestTime>Sep 22, 2015 3:13:08 PM</requestTime>
         <responseTime>Sep 22, 2015 3:13:08 PM</responseTime>
      </header>
   </SOAP:Header>
   <SOAP:Fault>
      <faultcode>SOAP:Client</faultcode>
      <faultstring>[security] Login failure.</faultstring>
      <faultactor>XmlApi</faultactor>
      <detail>
         <requestID>XML_API_client@n</requestID>
      </detail>
   </SOAP:Fault>
</SOAP:Envelope>
```

If a user is not assigned the appropriate scope of command privilege for a class type, or span of control for an object, an exception is returned.

See the *NSP System Administrator Guide* for information about how listing is filtered by span, and configuration is span-enforced.

Unlike in HTTP or HTTPS where span of control rules are implicitly enforced, span of control rules are not implicitly enforced via JMS. JMS requires the administrator to explicitly specify the required span IDs within the JMS filter. For example, without the correct JMS filter, it may not be possible to listen to events from objects that are not in the users span of control.

Table 3-1 Example JMS filter entries

JMS filter	Description
ALA_span like '%:2:%''	A JMS query that uses this filter results in JMS messages being sent from objects in span 2 (for example, Default Router Span). Events from objects in other spans are not published.
ALA_span like '%:21:%'' or ALA_span like ‘%:0:%’	A JMS query that uses this filter results in JMS messages being sent from objects in span 21 or non-span objects. Events from objects in other spans are not published.
(ALA_span like ‘%:1:%’ or ALA_span like ‘%:2:%’ or ALA_span like ‘%:3:%’ or ALA_span like ‘%:4:%’ or ALA_span like ‘%:5:%’ or ALA_span like ‘%:6:%’ or ALA_span like ‘%:7:%’ or ALA_span like ‘%:8:%’ or ALA_span like '%:0:%') and (ALA_span not like ‘%:22:%’)	A JMS query that uses this filter results in JMS messages being sent from all objects (all of Default Spans listed) except those blocked in span 22.

3.8 Workflow to use the XML API

3.8.1 Stages

The following is the sequence of high-level actions required to establish a connection to the XML API.

1

Ensure that the NFM-P is operational and that the license key supports the use of the XML API.

2

Develop an understanding of the information model and the XML message structure. See Chapter 7, “XML API information model overview” .

3

Determine whether the OSS application is request-response driven, acts as a passive event listener, or both. See 3.3 “OSS client interfaces” (p. 23) for an overview of each approach.

4

To communicate using JMS, see Chapter 4, “Event monitoring using JMS” .

5

To communicate using XML, see Chapter 5, “XML requests” .

4 Event monitoring using JMS

4.1 JMS overview

4.1.1 Overview

This chapter describes the NFM-P JMS implementation, including JMS client configuration, operation, and management. The NFM-P acts as a JMS provider for OSS clients that act as JMS consumers.

An OSS client can use JMS channels to receive the following types of near-real time event notifications:

- alarm
- object change
- statistics collection
- NFM-P system status

JMS is a published industry standard that is beyond the scope of this guide. See http://www.oracle.com/technetwork/java for information about JMS.

To receive JMS event messages, a JMS client establishes a connection and subscribes to a JMS topic. See 4.5.3 “JMS topics” (p. 35) in 4.5 “JMS subscriptions” (p. 35) for NFM-P JMS topic information, and Appendix B, “JMS events” for information about specific JMS events.

4.2 JMS connections

4.2.1 Overview

A JMS client requires the following to establish an NFM-P JMS connection:

- initial JMS context configuration
- each NFM-P main server IP address or hostname
- the appropriate samOss.jar file
- NFM-P user credentials

The user account requires an assigned scope of command profile that includes the OSS management role. The user password can be MD5-hashed or plain text.

Note: MD5-hashed passwords do not secure the communication channel.

If a firewall is between a main server and an OSS client, you must ensure that the firewall allows traffic to and from the required ports. See the *NSP Planning Guide* for the list of ports required for the NFM-P JMS.

4.2.2 Initial context configuration

Note: The JNP JNDI context is deprecated and will be removed in the future. The JNDI context for looking up the JMS connection factory is now based on remoting. Nokia recommends that you update each JMS OSS application to use remoting-based JNDI context attributes.

See the /opt/nsp/nfmp/server/nms/integration/SAM_O/jmsexample/README file on an NFM-P main server for more information.

The NFM-P JMS requires a user name and password for the JNDI initial context lookup. You can set the JNDI parameters in the client environment, or in a JNDI properties file on the client station; an environment value overrides a file value.

Nokia recommends that you set the JNDI parameters in the JNDI properties file, as described in 4.11 “To configure the initial JMS context” (p. 62) , rather than in the environment, to facilitate the maintenance of JNDI parameter changes. The JNDI automatically reads the file if the file is in the client Java class path.

4.2.3 samOss.jar

WARNING

Equipment Damage

An OSS client must use the samOss.jar from the current NFM-P release.

If a different samOss.jar is used, the NFM-P system may become unstable. The NFM-P release information is in the JAR manifest file.

The samOss.jar file contains the classes required to connect to the XML API. If an OSS client must connect to multiple NFM-P systems at different releases, the OSS must use separate JVMs and the appropriate samOss.jar file for each NFM-P release. Alternatively, an OSS client can use multiple class loader instances in one JVM to access multiple samOss.jar versions.

Obtaining the samOss.jar file

4.12 “To compile and connect an NFM-P JMS client” (p. 62) describes how to obtain the samOss.jar file, and how to configure and test a JMS connection.

4.3 JMS and redundancy

4.3.1 Overview

A redundant NFM-P system implements warm-standby JMS redundancy using JNDI. If a JMS client provides the IP address of each main server to the JNDI API, JNDI resolves the current primary main server and connects the client to the server.

When a server activity switch occurs, all JMS clients are disconnected and unsubscribed, and JMS services on the former standby main server initialize as the server assumes the primary role. A JMS client must then connect to the new primary main server and resubscribe to the JMS topic. After a client successfully connects and resubscribes, the client can attempt to recover any lost events.

See A.2 “Recommended durable JMS client operation” (p. 327) for the JMS client queuing, event monitoring, and disconnection recommendations.

4.4 JMS statistics

4.4.1 Overview

The NFM-P collects JMS statistics that include the following:

- all JMS Subscriber Topic statistics for the 5620-SAM-topic-xml-filtered topic
- JMS Subscriber Session statistics for each active subscription to the 5620-SAM-topic-xml-filtered topic; you can use the statistics to monitor the count and size of the filtered event vessels
- Publisher XML Event statistics that are related to advanced JMS message filters; you can use the statistics to monitor the number of filter change events, and the count and size of the published event vessels

See the *NSP NFM-P Statistics Management Guide* for more information about JMS statistics.

4.5 JMS subscriptions

4.5.1 Overview

A JMS client must specify the following to subscribe to a JMS topic:

- client ID
- topic
- message filter
- acknowledgment mode
- persistence mode

4.5.2 Client IDs

A client ID identifies the JMS client associated with a subscription, and has the following format:

name@ID

where

name uniquely identifies the client

ID uniquely identifies a subscription of the client

4.5.3 JMS topics

A JMS client controls which event messages it receives by subscribing to the appropriate JMS topic. The following table lists the available NFM-P JMS topics. The 5620-SAM-topic-xml topic sends all events. Other topics send specific event types; for example, the 5620-SAM-topic-xml-fault topic sends only fault events.

Table 4-1 JMS topics

Topic	Description
5620-SAM-topic-xml	Subscribe to all events. This topic publishes all events with no dependency on other topics.
5620-SAM-topic-xml-filtered	Subscribe to all events using an advanced filter to limit the number of JMS messages that are sent.
5620-SAM-topic-xml-general	Subscribe to general events, such as object creation and deletion. This topic contains objects that are not in the fault, file, or statistics topics.
5620-SAM-topic-xml-fault	Subscribe to fault events.
5620-SAM-topic-xml-file	Subscribe to findToFile events.
5620-SAM-topic-xml-stats	Subscribe to events related to statistics collection.
5650-CPAM-topic-xml	Subscribe to BGP route-change notifications.

4.5.4 Filters

A JMS subscription must include a filter that, based on JMS header properties, defines the types of event messages that the client receives. A subscription requires only a simple filter, but a client that subscribes to the 5620-SAM-topic-xml-filtered topic can use advanced filters to reduce the volume of JMS traffic.

A subscription to the 5620-SAM-topic-xml-filtered topic enables additional, advanced filtering based on the event properties, or the properties of objects associated with events. See 4.6 “JMS message filtering” (p. 37) for information about creating simple and advanced JMS event filters.

4.5.5 Acknowledgment modes

A JMS client must specify one of the following acknowledgment modes:

- **AUTO_ACKNOWLEDGE**
  The client automatically acknowledges each message that it receives. After the NFM-P sends a message, it does not send a subsequent message until it receives a message acknowledgment from the client. The efficiency of this acknowledgment mode is highly dependent on network latency.
- **DUPS_OK_ACKNOWLEDGE**
  The NFM-P continues to send messages to a client before all previous messages are acknowledged. This acknowledgment mode allows for higher message throughput in high-latency networks, and reduces session overhead and message delays. The communication bandwidth is used more efficiently, and performance is not significantly affected by an increase in network latency.

The NFM-P resends all unacknowledged messages after a communication interruption. Using the AUTO_ACKNOWLEDGE mode, a client may receive one duplicate message if the connectivity failure occurs after the client receives the message and before the NFM-P receives the acknowledgment. Using the DUPS_OK_ACKNOWLEDGE mode, the client may receive several duplicate messages when communication is restored.

The following figure shows DUPS_OK__ACKNOWLEDGE as the message acknowledgment mode. The first parameter of the createTopicSession method is transacted and this must be set to false for

acknowledgement mode to be used. Transacted sessions are not supported by the NFM-P.

Figure 4-1 JMS client message acknowledgment mode

```
topicSession=topicConnection.createTopicSession(false,TopicSession.DUPS_OK_
ACKNOWLEDGE );
```

4.5.6 Persistence modes

Two subscription persistence modes are available to a JMS client: durable and non-durable. A durable subscription can persist when a client is not connected; the NFM-P continues to process and store messages for the client. The NFM-P immediately removes a non-durable subscription when the client is no longer connected, and stops processing messages for the client.

The client tolerance for missed events determines whether a durable or non-durable subscription is best. For each subscription type, the NFM-P notifies the client of missed events, after which the client can take corrective action, if required. See 4.8.5 “Missed events” (p. 58) in 4.8 “Connection monitoring and error recovery” (p. 57) for more information.

4.6 JMS message filtering

4.6.1 Overview

A JMS client subscription to an NFM-P topic must include a simple filter that specifies the event messages of interest to the client. A simple JMS message filter is based only on the JMS message header properties and uses SQL syntax. See 4.6.4 “Simple JMS message filters” (p. 40) in this section.

A JMS client that subscribes to the 5620-SAM-topic-xml-filtered topic can define advanced filters based on the JMS message header properties, and the properties of the event object. See 4.6.5 “Advanced JMS message filters” (p. 42) in this section.

An NFM-P GUI operator can apply alarm filters to active OSS clients. See 6.5 “Assigning OSS alarm filters from the NFM-P client GUI” (p. 83) for information.

Note: It is recommended that you create filters that are as restrictive as possible so that only the required events are sent. If the filter is not restrictive enough, additional events must be queued and processed, which may cause the following:

- queue overflows that result in missed events
- wasted bandwidth
- degraded NFM-P and OSS performance

The SOAP header and footer are shown in code examples only when required to provide context.

4.6.2 JMS message header properties

The following table lists and describes the JMS event message header properties that are included in all event messages

Table 4-2 Event header properties, all events

Property	Values	Description
ALA_allomorphic	Allomorphic class	The allomorphic class of the object type, if applicable
ALA_category	ACCOUNTING—accounting statistics CPAM—events related to CPAM topology management DATABASE—database state messages EQUIPMENT—physical equipment events FAULT—fault management events (alarms) GENERAL—events that are not in another category STATISTICS—statistics, except accounting statistics SERVICE—service-related events SOFTWARE—not currently used	The event category
ALA_clientId	OSS client ID, or empty	The client ID associated with the event, if applicable
ALA_isVessel	false true	Whether the message is an event vessel
ALA_OLC	0 (object has no OLC state) 1 (maintenance) 2 (in service)	OLC state. Applies to the creation, attribute changes, and deletion for objects that have an OLC state
ALA_routeManager	Application name of the route manager	Present if the message contains a managed-route change
ALA_span	List of control IDs	The OSS user span of control ID list
ALA_topic	JMS topic name	The JMS topic of the message
ALA_vesselSize	0 to 250	Size, in messages, of the event vessel; present only if ALA_isVessel value is true
eventName	See Table 4-5, “Filterable event classes” (p. 42) .	The JMS event class name, for example, ObjectDeletionEvent.
MTOSI_NTType	ALA_OTHER NT_OBJECTDELETION NT_OBJECTCREATION NT_ATTRIBUTE_VALUE_CHANGE NT_STATE_CHANGE NT_ALARM NT_HEARTBEAT ALA_RELATIONSHIPCHANGE	The type of notification
MTOSI_objectName	Object name	The name of the associated object

Table 4-2 Event header properties, all events (continued)

Property	Values	Description
MTOSI_objectType	KeepAliveEvent AlarmStatusChangeEvent DBActivityEvent DBProxyStateChangeEvent DBErrorEvent DBConnectionStateChangeEvent StateChangeEvent FileAvailableEvent DeployerEvent TerminateClientSession Any network object names in the form of *packageName.ClassName* for the following events: • AttributeValueChangeEvent • ObjectCreationEvent • ObjectDeletionEvent • RelationshipChangeEvent • StatsEvent	The type of associated object
MTOSI_osTime	Time, in ms, since January 1, 1970	The server system time at event creation

Alarm-specific header properties

Most JMS message header properties apply to all events. However, some alarm-related properties, for example, ALA_alarmType, apply only to specific events. The filter shown in the following figure includes all events except equipment alarms; the filter also excludes events that do not include the ALA_alarmType property.

Figure 4-2 Filter example, all events except equipment alarms

```
ALA_clientId in ('JMS_client@n', '') and ALA_alarmType not in ('equipmentAlarm')
```

Alarm-specific properties do not apply to all alarm events; when you create a filter on a non-fault topic using alarm-specific properties, you must include the “is null” case. The filter shown in the following figure excludes events with an ALA_alarmType of equipmentAlarm, but includes all other events:

Figure 4-3 Filter example, all events except equipment alarms

```
ALA_clientId in ('JMS_client@n','') and (ALA_alarmType is null or ALA_alarmType
not in ('equipmentAlarm'))
```

The following table lists properties that are included in some alarm-related events.

Table 4-3 Event header properties, alarm events

Property	Values	Description
ALA_alarmType	Alarm type	The type of alarm
ALA_isCorr	True, if alarm is correlated	Whether the alarm is correlated, that is, caused by another alarm. See “Correlated alarms” in the *NSP NFM-P Classic Management User Guide* for information about correlated alarms.
MTOSI_aliasNameList	Alarm name	The alarm name
MTOSI_perceivedSeverity	Severity	The alarm severity
MTOSI_probableCause	Probable cause	The probable cause associated with the alarm
MTOSI_serviceAffecting	True if service-affecting	Whether the alarm is service-affecting

The Boolean example shown in the figure below is a filter that suppresses correlated alarms, but also includes events that do not have the ALA_isCorr property. See Troubleshooting using network alarms in the *NSP Troubleshooting Guide* for information about correlated alarms.

Figure 4-4 Filter example, non-correlated alarms

```
ALA_clientId in ('JMS_client@n','') and (ALA_isCorr is null or ALA_isCorr = false)
```

4.6.3 Mandatory events

You must ensure that the following JMS events that a client must monitor are not inadvertently filtered:

- KeepAliveEvent
- StateChangeEvent
- TerminateClientSessionEvent
- JmsMissedEvent
- all events destined for the client ID

The following figure shows a filter example that includes all mandatory events.

Figure 4-5 Filter example, all mandatory events

```
ALA_clientId in ('JMS_client@n', '')
  AND (
   (<filter_for_specific_event_types>)
   OR
   MTOSI_objectType in ('KeepAliveEvent', 'StateChangeEvent', 'Terminate-
ClientSession')
   )
```

4.6.4 Simple JMS message filters

A simple JMS message filter is defined during subscription to a topic, and is a string of up to 3000 characters that uses the syntax of a WHERE clause in an SQL query. To change a filter, a client

must unsubscribe from a topic and then resubscribe to the topic using the new filter. The following figure shows a simple JMS message filter.

Figure 4-6 Simple JMS filter example

```
ALA_clientId in ('JMS_client@n', '')
```

The filter includes all of the events in the topic that are intended for the client called ossname@1, and all events in the topic that are intended for all subscribers.

Note: A subscription must filter on the client ID; otherwise, the subscription is rejected.

A simple filter can include any JMS header properties. Some common properties used for filtering include the following:

- MTOSI_NTType, the notification type
- ALA_category, the event category
- MTOSI_objectType, the object type

See Table 4-2, “Event header properties, all events” (p. 38) for a complete list of header properties, and Table 4-3, “Event header properties, alarm events” (p. 40) for header properties that are specific to alarm events. Each property is defined in the xmlApiJmsHeader.xsd file in the Schema Reference.

The following table contains simple JMS message filter examples.

Table 4-4 Simple JMS filter samples

JMS filter	Description
ALA_clientId in ('*JMS_client@n*', '')	To send general messages, and messages qualified with a client ID, directly to the specified client. This filter blocks messages sent to other clients. You must specify this information in all XML topic-based filters.
ALA_clientId in ('*JMS_client@n*', '') and ALA_category = 'FAULT' and MTOSI_perceivedSeverity = 'Warning'	To filter fault messages with a warning severity class. The filter excludes some mandatory events; see 4.6.3 “Mandatory events” (p. 40) in this section.
ALA_clientId in ('*JMS_client@n*', '') and ALA_category not in ('STATISTICS', 'ACCOUNTING')	To stop client statistics, accounting, and keep-alive messages
ALA_clientId in ('*JMS_client@n*', '') and (MTOSI_NTType in ('NT_ATTRIBUTE_VALUE_CHANGE', 'NT_OBJECTDELETION', 'NT_OBJECTCREATION') and MTOSI_objectType in ('equipment. PhysicalPort', 'equipment.DaughterCard', 'equipment.BaseCard'))	To include only object creation, object deletion, and attribute changes for cards and ports. The filter excludes some mandatory events; see 4.6.3 “Mandatory events” (p. 40) in this section.
ALA_clientId in ('*JMS_client@n*', '') and (MTOSI_NTType in ('NT_OBJECTDELETION', 'NT_OBJECTCREATION') and MTOSI_objectType like 'equipment.%')	To include only object creation and deletion for objects in the equipment package. The filter excludes some mandatory events; see 4.6.3 “Mandatory events” (p. 40) in this section.

4.6.5 Advanced JMS message filters

CAUTION

Service Disruption

The registerNotification method entry in the General Methods/Types page of the XML API Reference defines the maximum allowed number of leaf elements in a filter.

To avoid degraded system performance, Nokia recommends that you consider the current NFM-P system load, as well as this limit, before you construct a filter with a large number of leaf elements.

A JMS client that subscribes to the 5620-SAM-topic-xml-filtered topic can use advanced filters to additionally reduce the number of JMS event messages. An advanced JMS message filter specifies the following:

- the types of event notifications that are to be sent to the client
- a set of object properties as criteria for messages sent to the client

Note: If an OSS client registers a filter but does not subscribe to the 5620-SAM-topic-xml-filtered topic within 10m, the filter is deregistered.

If a JMS client unsubscribes from the 5620-SAM-topic-xml-filtered topic, the NFM-P automatically deregisters the associated filter. If a non-durable OSS client connection drops, the filter is deregistered after two minutes; durable client filters are not deregistered.

The following table lists the event classes that support advanced filtering based on the event properties, the properties of the objects associated with the event, or both.

Table 4-5 Filterable event classes

Event	Event properties	Object properties
AlarmStatusChangeEvent	✓	
AttributeValueChangeEvent	✓	✓
DeployerEvent	✓	
ExceptionEventXMLFormat	✓	✓
FileAvailableEvent	✓	
IncrementalRequestEvent	✓	
LogFileAvailableEvent	✓	
LogRemoteFileAvailableEvent	✓	
ObjectCreationEvent	✓	✓
ObjectDeletionEvent	✓	✓
RelationshipChangeEvent	✓	
ScriptExecutionEvent	✓	
StatsEvent	✓	

Event vessels

To further reduce the volume of JMS traffic, most event messages for the 5620-SAM-topic-xml-filtered topic are sent in containers called event vessels. An event vessel is a type of JMS message that contains multiple events and by default, only one JMS header.

Note: Although the JMS headers of individual events are not sent in an event vessel, you can filter the properties of the headers by specifying "class=jmsEvent" in a filter. See "Class-based filtering" (p. 48) in this section.

You can configure the NFM-P to return only specific attributes of the event bodies in the JMS message by including the resultFilter-Set tag in a registration message. See "Result filters" (p. 52) in this section.

The following events are sent on the 5620-SAM-topic-xml-filtered topic but are not sent in an event vessel:

- DBActivityEvent
- DBConnectionStateChangeEvent
- DBProxyStateChangeEvent
- EventVessel
- KeepAliveEvent
- StateChangeEvent
- TerminateClientSessionEvent
- XMLFilterChangeEvent

Figure 4-7, "Event vessel message example" (p. 43) is an example of an EventVessel message that contains the following:

- AttributeValueChangeEvent
- StatsEvent
- ExceptionEvent

Figure 4-7 Event vessel message example

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>EventVessel</eventName>
         <ALA_category>GENERAL</ALA_category>
         <ALA_vesselSize>4</ALA_vesselSize>
         <ALA_isVessel>true</ALA_isVessel>
         <MTOSI_osTime>1308318280367</MTOSI_osTime>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <ALA_eventName>EventVessel</ALA_eventName>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <eventVessel>
            <attributeValueChangeEvent>
               <fullClassName>fm.AlarmObject</fullClassName>
               <objectFullName>faultManager:network@192.168.183.13|alarm-31-4-24
```

```
</objectFullName>
               <attribute>
                  <attributeName>numberOfOccurencesSinceClear</attributeName>
                  <newValue>
                     <long>3</long>
                  </newValue>
                  <oldValue>
                     <long>2</long>
                  </oldValue>
               </attribute>
            </attributeValueChangeEvent>
            <statsEvent>
               <MTOSI_osTime>1311602579077</MTOSI_osTime>
               <state>begin</state>
               <networkElement>192.168.183.13</networkElement>
               <statsType>accounting</statsType>
               <time>1308256029287</time>
            </statsEvent>
            <statsEvent>
               <MTOSI_osTime>1311602582098</MTOSI_osTime>
               <state>end</state>
               <networkElement>192.168.183.13</networkElement>
               <statsType>accounting</statsType>
               <time>1308256029302</time>
            </statsEvent>
            <ExceptionEventXMLFormat>
               <className>aengr.Policy</className>
               <operation>distribute on node 10.168.1.91</operation>
               <objectName>Access Egress:2100</objectName>
               <requestID>JMS_client@n</requestID>
               <errorMessage>[ app: autoconfigChild ] [ class: aengr.Policy ] [
instance: network:10.168.1.91:Access Egress:2100 ] [ descr: invalid action
bitmask: 3 : Object deletion is in progress, the object cannot be configured
:Parent = network:10.168.1.91:Access Egress:2100: child = queue-1 ]
               </errorMessage>
            </ExceptionEventXMLFormat>
         </eventVessel>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

Filter elements

An advanced filter can contain leaf and composite filter elements. Leaf elements define an arithmetic or string function and do not have child elements. Composite filters use Boolean expressions to evaluate child elements, which can be leaf or composite elements. An empty filter returns all objects that match the XML request or JMS subscription criteria.

Note: Nokia recommends that you create filters that have a top-down hierarchical structure to make them as restrictive as possible. This means that a filter lists the higher-level object properties before the lower-level properties, as defined in the NFM-P object hierarchy. See the XML API Reference for object hierarchy information.

The following table lists the available leaf filter elements.

Table 4-6 Leaf filter elements

Filter element	Description	Example
equal	Return true if the value is the same as the specified value.	<equal name="localAS" value="20"/>
notEqual	Return true if the value is not the same as the specified value.	<notEqual name="localAS" value="20"/>
greater	Return true if the numeric value is greater than the specified value.	<greater name="localAS" value="20"/>
greaterOrEqual	Return true if the numeric value is greater than or equal to the specified value.	<greaterOrEqual name="localAS" value="20"/>
less	Return true if the numeric value is less than the specified value.	<less name="localAS" value="20"/>
lessOrEqual	Return true if the numeric value is less than or equal to the specified value.	<lessOrEqual name="localAS" value="20"/>
anyBit	Return true if an AND operation on the value bit and the specified value does not return 0. The filter is true if any of the "1" bits in the specified value are enabled in the property value. The function applies only to non-negative numeric types and bitmask types.	<anyBit name="parameter" value="3"/>
allBits	Return true if an AND operation on the value bit and the specified value returns the specified value. The filter is true if all of the "1" bits in the specified value are enabled in the property value. This applies only to non-negative numeric types and bitmask types.	<allBits name="parameter" value="5"/>
wildcard	Compares values using wildcards. The % character specifies 0 or more characters of any type, and the _ character specifies one character of any type.	<wildcard name="SiteId" value="1.%.%.%.%"/>
in	Return true if a numeric or string value equals one of the values in the specified comma-separated list.	<in name="parameter" value="[Edge_14, Aggregation_14, Core_14]"/>
subset	Return true if a string value is a subset of the specified string.	<subset class="service.Site" name="tier" value="123"/>
superset	Return true if a string value contains the specified string.	<superset class="service.Site" name="siteName" value="Core_"/>
notSuperset	Return true if a string value does not contain the specified string.	<notSuperset class="service.Site" name="siteName" value="Edge_"/>
between	Return true if the numeric value is between the first and second specified values.	<between name="preference" first="150" second="200" />

Table 4-6 Leaf filter elements (continued)

Filter element	Description	Example
past	Return true if the time value is between (current time) and (current time - specified time interval).	<past time="300000" /> means find all items from the past 5 minutes (300000 ms)
empty	Match everything; this is the same as specifying nothing between the <filter> and </filter> tags.	<empty />

Advanced filter registration

A client specifies a filter using the registerNotification method, which has the following parameters:

- required:
  - client ID
  - event filter criteria
- optional:
  - result filter criteria, to refine the information per event body; see "Result filters" (p. 52) in this section
  - extraTags tag, which specifies the JMS header properties to return; see "Using extraTags element to include header properties" (p. 49) in this section

Figure 4-8, "Registration message example with filter" (p. 46) shows a registration message example that contains an advanced filter. The filter elements are leaf filters that are evaluated using the Boolean OR function. The NFM-P sends only JMS events for objects that match one or more of the following criteria:

- service ID between 150 and 200
- NE with an out-of-band management address in the 192.168.101 subnet
- alarm

Figure 4-8 Registration message example with filter

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <security>
            <user>username</user>
            <password>password</password>
         </security>
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <registerNotification xmlns="xmlapi_1.0">
         <jmsClientId>JMS_client@n</jmsClientId>
         <filter-Set>
            <filter>
```

```
               <between class="service.Service" name="serviceId" first="150"
second="200"/>
            </filter>
            <filter>
               <wildcard class="netw.NetworkElement" name="outOfBandAddress"
value="192.168.101%"/>
            </filter>
            <filter>
               <equal class="jmsEvent" name="ALA_category" value="FAULT"/>
            </filter>
         </filter-Set>
      </registerNotification>
   </SOAP:Body>
</SOAP:Envelope>
```

The registrationNotification response contains the complete filter definition, which includes the filter defined in the request and any other filter applied using the GUI.

Note: To obtain all attributes of a particular class or abstract class, the following example using the service.AccessInterface abstract class shows how to define the filter:

```
<filter-Set>
    <filter class="service.AccessInterface">
        <and>
            <notEqual name="eventName" value="" class="jmsEvent"/>
        </and>
    </filter>
</filter-Set>
```

Advanced filter deregistration

To deregister a filter, a JMS client uses the deregisterNotification method, which requires only the JMS client ID. The following figure shows a deregistration message example.

Figure 4-9 Deregistration message example

```
<SOAP:Body>
   <deregisterNotification xmlns="xmlapi_1.0">
      <jmsClientId>JMS_client@n</jmsClientId>
   </deregisterNotification>
 </SOAP:Body>
```

The NFM-P returns an XMLFilterChangeEvent with an empty filter-Set element in response to a deregisterNotification message, as shown in the following figure.

Figure 4-10 XMLFilterChangeEvent message example

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>XMLFilterChangeEvent</eventName>
```

```
         <ALA_category>GENERAL</ALA_category>
         <ALA_OLC>0</ALA_OLC>
         <ALA_isVessel>false</ALA_isVessel>
         <ALA_allomorphic/>
         <MTOSI_osTime>1308255929252</MTOSI_osTime>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <MTOSI_objectName/>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <MTOSI_objectType>XMLFilterChangeEvent</MTOSI_objectType>
         <ALA_eventName>XMLFilterChangeEvent</ALA_eventName>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <xmlFilterChangeEvent>
            <filter-Set />
         </xmlFilterChangeEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

Class-based filtering

A class= tag in a filter element specifies the object type to evaluate. You can specify the jmsEvent class to indicate that the filter element is for the event. If a class value is not specified or if the class value is jmsEvent, class-based filtering is not performed. See Chapter 13, “Inventory management” for information about working with classes.

When you specify jmsEvent as the class value, you can filter on the following JMS header properties:

- ALA_alarmType
- ALA_allomorphic
- ALA_application
- ALA_category
- ALA_eventName
- ALA_isCorr
- ALA_OLC
- ALA_routeManager
- ALA_span
- eventName
- fullClassName
- MTOSI_aliasNameList
- MTOSI_NTType
- MTOSI_objectName
- MTOSI_objectType
- MTOSI_osTime
- MTOSI_perceivedSeverity
- MTOSI_probableCause
- MTOSI_serviceAffecting

Note: The OSS client user span of control defines the objects that are sent to the client. Using ALA_span in a filter is not required unless you want to restrict the returned objects to a subset of the objects within the span of control, or to a specific span such as the Edit span.

When you specify jmsEvent as the class, you can also filter on event detail properties. The following table lists the jmsEvent detail properties that are filterable.

Table 4-7 Filterable jmsEvent class detail properties

Event detail	Properties
DeployerEvent	successList failedList
ExceptionEvent	objectName operation taskName errorMessage
FileAvailableEvent	fileName
IncrementalRequestEvent	incrementalAction serviceType
LogFileAvailableEvent	serverIpAddress neId
LogRemoteFileAvailableEvent	serverIpAddress neId fileName
RelationshipChangeEvent	changeType fromObjectName fromObjectClass toObjectName toObjectClass
ScriptExecutionEvent	targetScriptResultFullName fileName failedCommands systemInvokationId
StatsEvent	networkElement statsType

Using extraTags element to include header properties

By default, JMS event vessel messages do not include the JMS header properties of the contained events. In order to return event header properties, you must specify the properties in an extraTags element during filter registration. The header properties are then included in the body of each associated event that has the header property.

The properties are specified using the following syntax:

```
<tag name="JMS_header_property" eventName="JMS_event_class" />
```

where

JMS_header_property is a JMS header property described in “Class-based filtering” (p. 48) in this section

JMS_event_class is one of the event classes listed in Table 4-5, “Filterable event classes” (p. 42)

If a JMS event does not have a specified header property, the property is excluded from the event body. If an eventName value is omitted, the specified JMS header property is returned for each event that has the property.

Figure 4-11, “Filter registration example with extraTags element” (p. 49) is an example of a filter registration message with an extraTags element that specifies the following.

- Each StatsEvent message is to include the MTOSI_osTime property.
- Each message is to include the fullClassName property; however, a StatsEvent message does not have a fullClassName property, so the property is not included in the StatsEvent message body, as shown in Figure 4-12, “Event vessel example with extraTags properties” (p. 50) .

Figure 4-11 Filter registration example with extraTags element

```
<SOAP:Body>
   <registerNotification xmlns="xmlapi_1.0">
      <jmsClientId>JMS_client@n</jmsClientId>
      <filter-Set>
         <filter>
            <or>
               <equal class="jmsEvent" name="networkElement" value="192.168.101.
145" />
               <equal class="jmsEvent" name="MTOSI_objectType" value="StatsEvent"
/>
            </or>
         </filter>
      </filter-Set>
      <extraTags>
         <tag name="MTOSI_osTime" eventName="StatsEvent" />
         <tag name="fullClassName" />
      </extraTags>
   </registerNotification>
</SOAP:Body>
```

Figure 4-12 Event vessel example with extraTags properties

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>EventVessel</eventName>
         <ALA_category>GENERAL</ALA_category>
         <ALA_vesselSize>3</ALA_vesselSize>
         <ALA_isVessel>true</ALA_isVessel>
         <MTOSI_osTime>1308318280367</MTOSI_osTime>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <ALA_eventName>EventVessel</ALA_eventName>
      </header>
```

```
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <eventVessel>
            <attributeValueChangeEvent>
               <fullClassName>fm.AlarmObject</fullClassName>
               <objectFullName>faultManager:network@192.168.101.145|alarm-31-4-24
</objectFullName>
               <attribute>
                  <attributeName>numberOfOccurencesSinceClear</attributeName>
                  <newValue>
                     <long>3</long>
                  </newValue>
                  <oldValue>
                     <long>2</long>
                  </oldValue>
               </attribute>
               <attribute>
                  <attributeName>lastTimeDetected</attributeName>
                  <newValue>
                     <long>1308318277307</long>
                  </newValue>
                  <oldValue>
                     <long>1308318115170</long>
                  </oldValue>
               </attribute>
               <attribute>
                  <attributeName>numberOfOccurences</attributeName>
                  <newValue>
                     <long>3</long>
                  </newValue>
                  <oldValue>
                     <long>2</long>
                  </oldValue>
               </attribute>
            </attributeValueChangeEvent>
            <statsEvent>
               <MTOSI_osTime>1311602579077</MTOSI_osTime>
               <state>begin</state>
               <networkElement>192.168.183.13</networkElement>
               <statsType>accounting</statsType>
               <time>1308256029287</time>
            </statsEvent>
            <statsEvent>
               <MTOSI_osTime>1311602582098</MTOSI_osTime>
               <state>end</state>
               <networkElement>192.168.183.13</networkElement>
               <statsType>accounting</statsType>
               <time>1308256029302</time>
```

```
            </statsEvent>
         </eventVessel>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

Result filters

Figure 4-13, “registerNotification message example with result filter” (p. 51) shows a registerNotification example using a result filter. The result filter specifies that only ObjectCreation events are sent to JMS_CLIENT@4. The resultFilter elements define the following filter criteria:

- the objectFullName, operationalState, and administrativeState properties of each ObjectCreation event associated with an equipmentPort object
- the affectedObjectFullName, severity, operatorAssignedUrgency, and additionalText properties of each ObjectCreation event associated with an fm.AlarmObject object
- the objectFullName property of all other ObjectCreation events

Figure 4-13 registerNotification message example with result filter

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <security>
            <user>username</user>
            <password>password</password>
         </security>
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <registerNotification xmlns="xmlapi_1.0">
         <jmsClientId>JMS_client@n</jmsClientId>
         <filter-Set>
            <filter>
               <equal class="jmsEvent" name="ALA_eventName" value="ObjectCreation"
/>
            </filter>
         </filter-Set>
         <resultFilter-Set>
            <resultFilter>
               <attribute>objectFullName</attribute>
            </resultFilter>
            <resultFilter class="equipment.Port">
               <attribute>objectFullName</attribute>
               <attribute>operationalState</attribute>
               <attribute>administrativeState</attribute>
            </resultFilter>
            <resultFilter class="fm.AlarmObject">
```

```
               <attribute>operatorAssignedUrgency</attribute>
               <attribute>additionalText</attribute>
               <attribute>severity</attribute>
               <attribute>affectedObjectFullName</attribute>
            </resultFilter>
         </resultFilter-Set>
      </registerNotification>
   </SOAP:Body>
</SOAP:Envelope>
```

After a successful registration, the XMLFilterChangeEvent message shown in the following figure is sent to the JMS client.

Figure 4-14 XMLFilterChangeEvent message example

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>XMLFilterChangeEvent</eventName>
         <ALA_category>GENERAL</ALA_category>
         <ALA_OLC>0</ALA_OLC>
         <ALA_isVessel>false</ALA_isVessel>
         <ALA_allomorphic/>
         <MTOSI_osTime>1335291140845</MTOSI_osTime>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <MTOSI_objectName/>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <MTOSI_objectType>XMLFilterChangeEvent</MTOSI_objectType>
         <ALA_eventName>XMLFilterChangeEvent</ALA_eventName>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <xmlFilterChangeEvent>
            <filter-Set>
               <filter>
                  <equal name="ALA_eventName" value="ObjectCreation" class=
"jmsEvent"/>
               </filter>
            </filter-Set>
            <resultFilter-Set>
               <resultFilter>
                  <attribute>objectFullName</attribute>
               </resultFilter>
               <resultFilter class="equipment.Port">
                  <attribute>administrativeState</attribute>
                  <attribute>operationalState</attribute>
                  <attribute>objectFullName</attribute>
```

```
               </resultFilter>
               <resultFilter class="fm.AlarmObject">
                  <attribute>operatorAssignedUrgency</attribute>
                  <attribute>additionalText</attribute>
                  <attribute>severity</attribute>
                  <attribute>affectedObjectFullName</attribute>
               </resultFilter>
            </resultFilter-Set>
         </xmlFilterChangeEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

The following three figures show ObjectCreationEvent event vessel message examples based on the result filters specified in the previous figure.

Figure 4-15 Event vessel example, physical port

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>EventVessel</eventName>
         <ALA_category>GENERAL</ALA_category>
         <ALA_vesselSize>50</ALA_vesselSize>
         <ALA_isVessel>true</ALA_isVessel>
         <MTOSI_osTime>1334951029044</MTOSI_osTime>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <ALA_eventName>EventVessel</ALA_eventName>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <eventVessel>
            <objectCreationEvent>
               <equipment.PhysicalPort>
                  <operationalState>portInService</operationalState>
                  <administrativeState>portInService</administrativeState>
                  <objectFullName>network:198.51.100.24:shelf-1:cardSlot-1:card:
daughterCardSlot-1:daughterCard:port-1</objectFullName>
               </equipment.PhysicalPort>
            </objectCreationEvent>
            <objectCreationEvent>
               <equipment.PhysicalPort>
                  <operationalState>portInService</operationalState>
                  <administrativeState>portInService</administrativeState>
                  <objectFullName>network:198.51.100.24:shelf-1:cardSlot-1:card:
daughterCardSlot-1:daughterCard:port-2</objectFullName>
```

```
            </equipment.PhysicalPort>
         </objectCreationEvent>
         <objectCreationEvent>
            <equipment.PhysicalPort>
               <operationalState>portInService</operationalState>
               <administrativeState>portInService</administrativeState>
               <objectFullName>network:198.51.100.24:shelf-1:cardSlot-1:card:
daughterCardSlot-1:daughterCard:port-3</objectFullName>
            </equipment.PhysicalPort>
         </objectCreationEvent>
      </eventVessel>
   </jms>
</SOAP:Body>
</SOAP:Envelope>
```

Figure 4-16 Event vessel example, alarm

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>EventVessel</eventName>
         <ALA_category>GENERAL</ALA_category>
         <ALA_vesselSize>1</ALA_vesselSize>
         <ALA_isVessel>true</ALA_isVessel>
         <MTOSI_osTime>1335291256665</MTOSI_osTime>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <ALA_eventName>EventVessel</ALA_eventName>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <eventVessel>
            <objectCreationEvent>
               <fm.AlarmInfo>
                  <severity>critical</severity>
                  <affectedObjectFullName>network:198.51.100.24</affectedObjectFullName>

                  <additionalText>N/A</additionalText>
                  <operatorAssignedUrgency>indeterminate</operatorAssignedUrgency>
               </fm.AlarmInfo>
            </objectCreationEvent>
         </eventVessel>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

Figure 4-17 Event vessel example returning only objectFullName

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>EventVessel</eventName>
         <ALA_category>GENERAL</ALA_category>
         <ALA_vesselSize>1</ALA_vesselSize>
         <ALA_isVessel>true</ALA_isVessel>
         <MTOSI_osTime>1335291361323</MTOSI_osTime>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <ALA_eventName>EventVessel</ALA_eventName>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <eventVessel>
            <objectCreationEvent>
               <rtr.VirtualRouterIpAddress>
                  <objectFullName>network:198.51.100.24:router-1:ip-interface-2:
TP-1</objectFullName>
               </rtr.VirtualRouterIpAddress>
            </objectCreationEvent>
         </eventVessel>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

4.7 Managing and monitoring client sessions

4.7.1 Overview

You can use the NFM-P GUI to manage and monitor JMS client sessions. You can view the session status, and close or remove sessions. You can also use the stopOssConnection method in the security.SamJmsTopicManager class over HTTP to close a JMS session. You must be an admin user, or the user that initiates a session, to close the session using this method.

When you close a session that has a non-durable subscription, the NFM-P sends a TerminateClientSession event to the client, disconnects the client, and removes the client subscription.

When you close a session that has a durable subscription, the NFM-P sends a TerminateClientSession event to the client and disconnects the client, but the subscription remains active. The NFM-P continues to process and store messages for the client, and the GUI shows the session status as Disconnected. 4.14 “To close a JMS client session, or remove a durable subscription” (p. 65) describes how to remove a disconnected session.

The NFM-P raises alarms for events that affect JMS sessions; the alarms are listed in Table 4-8, “JMS client alarms” (p. 59). The NFM-P also logs information about each JMS session and the associated JMS filters.

4.8 Connection monitoring and error recovery

4.8.1 Overview

A JMS client must monitor and, if required, recover from the following:

- connectivity loss
- event loss

The scenarios can occur separately or in conjunction. If a client has a durable subscription, a loss of connectivity may occur without causing events to be lost. Event loss can occur under heavy load without a connectivity loss; the client is notified using a jmsMissedEvents message. A subscription removal can accompany a connectivity loss, for example, after a main server activity switch or restart.

The manner in which a client manages each scenario depends on the specific needs and the expectations of their customers. For example, for some applications, a loss of connectivity should always result in the OSS application reconnecting to the NFM-P. In other situations, the desired behavior may depend on the cause of the connectivity loss—for example, if an NFM-P administrator intentionally disconnects an OSS, it may not be desirable to immediately reconnect.

How to recover from lost events also depends on the OSS. If an OSS uses JMS events to maintain a local information store that mirrors some data set in the NFM-P—such as a network inventory or a list of current alarms—then a loss of events results in the OSS being out of sync with the NFM-P.

The following scenarios could be used to determine whether the OSS needs to resync the database of inventory and alarm information:

- If a durable OSS disconnects but remains subscribed to the NFM-P, on reconnect if a JmsMissedEvent message is not received and the sysStartTime is not changed, it is not necessary to do a resync of inventory and alarm information because all events are queued on the NFM-P and await OSS reconnection.
- If a disconnected durable subscription is removed from an NFM-P main server, a JmsMissedEvents message is generated when the client reconnects using the same client ID. This indicates to the OSS that a resync of inventory and alarm information needs to be performed because events have been missed.

When an OSS detects a lack of synchronization, the recovery scenarios include:

- immediately resynchronizing with the NFM-P, such as by retrieving inventory information or the latest alarm list through the XML API
- notifying the OSS administrator of the problem, and allowing the administrator to select a recovery approach—for example, an immediate resync or a scheduled resync for a later time

When you implement a recovery procedure, you must consider that events will continue to occur in the network while an OSS is busy resynchronizing or populating its database for the first time. For this reason, an OSS must process events while they are resynchronizing, and may need to recover from error conditions that require them to reconnect or restart the resync.

4.8.2 JMS exceptions

You must use JMS exceptions if you have a JMS connection to an NFM-P server. JMS internally monitors client connections to the JMS server and throws an exception if a connection between the JMS server and a client is lost. You must implement the javax.jms.ExceptionListener interface on the JMS client to enable the monitoring of exceptions. The interface contains a call-back method that is invoked for all exceptions. A typical implementation of the call-back method attempts to reconnect and possibly take another action, such as generating an event.

4.8.3 Monitoring for incoming events

In addition to handling exceptions, an OSS must monitor incoming events to ensure that the JMS connection is active. A KeepAliveEvent is published at approximately 30-second intervals to each of the JMS topics even when no other messages are sent.

KeepAliveEvents may be received ahead of other event types.

If no events are received within a reasonable time period, you must investigate the status of the NFM-P server.

4.8.4 XML API session termination

You can use the NFM-P GUI client to close and remove a durable subscription when you no longer require a durable client. See the procedure to disconnect an XML API JMS client connection or remove a durable subscription in the *NSP System Administrator Guide* for more information about removing durable subscriptions.

The XML API uses the following JMS event to notify OSS client applications of a session termination:

```
TerminateClientSession
```

The TerminateClientSession event indicates that a client JMS session is about to be closed. The client must clean up the disconnected session when the message is received. Additional session termination behavior is dependent on the requirements of your OSS client application. For example, you can also configure the requirement to close the OSS client application.

4.8.5 Missed events

Both durable and non-durable subscriptions can be used by OSS clients that cannot tolerate losing events without taking corrective action. However, in certain situations, events can be missed and subscribers are notified with a JmsMissedEvents message.

JMS messages are queued by the NFM-P until they are acknowledged by the JMS client. If the JMS message queue overflows, the following occurs for both durable and non-durable JMS subscriptions:

- Connected non-durable subscribers
  - The NFM-P stops queuing messages until there is capacity in the queues.
  - A JmsMissedEvents message may or may not be sent, depending on the internal cause of the overflow and whether there is capacity in the queues.
- Connected durable subscribers
  - The queued messages are removed.

  - A JmsMissedEvents message is added to the queue. See Figure 4-18, “JmsMissedEvents message example” (p. 59) for an example.
  - New events continue to be queued or sent.
- Non-connected non-durable subscribers
  - Nothing occurs; a non-connected, non-durable subscriber is unsubscribed when it disconnects from the NFM-P.
- Non-connected durable subscribers
  - The queued messages are removed.
  - The client is unsubscribed.
  - No new events are queued.

The following are example scenarios in which messages may be missed:

- The OSS client is slow, the message rate is high, or there is high network latency.
- A client with an active subscription disconnects, resulting in messages being queued until the client reconnects.

The following table lists and describes the alarms that are raised against JMS clients.

Table 4-8 JMS client alarms

Alarm	Description
JMSDurableClientReset	Raised when a durable JMS client is reset as the result of a JMS server restart or activity switch
JMSClientMessagesRemoved	Raised when a JMS client has messages removed after exceeding the configured message limit. This applies to OSS durable subscribers only.
JMSDurableClientUnsubscribed	Raised when a durable JMS client is automatically unsubscribed. This occurs when a disconnected durable client exceeds the configured message limit.

JmsMissedEvents message

The NFM-P sends a JmsMissedEvents message to indicate that events have been lost, allowing subscribers to detect missed events. The JmsMissedEvents message is a StateChange Event, as shown in the following figure.

Figure 4-18 JmsMissedEvents message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>JmsMissedEvents</eventName>
         <MTOSI_osTime>1222891102168</MTOSI_osTime>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <MTOSI_objectType>StateChangeEvent</MTOSI_objectType>
```

```
            <ALA_category>GENERAL</ALA_category>
            <ALA_isVessel>false</ALA_isVessel>
            <ALA_allomorphic/>
            <ALA_eventName>JmsMissedEvents</ALA_eventName>
            <MTOSI_objectName/>
            <ALA_OLC>0</ALA_OLC>
         </header>
      </SOAP:Header>
      <SOAP:Body>
         <jms xmlns="xmlapi_1.0">
            <stateChangeEvent>
               <eventName>JmsMissedEvents</eventName>
               <state>jmsMissedEvents</state>
            </stateChangeEvent>
         </jms>
      </SOAP:Body>
</SOAP:Envelope>
```

Recovery from missed events

Clients must be able to recognize when events are missed and take the appropriate recovery action. For example, in an OSS application for which events are being used to maintain inventory information, the recovery action could include:

- an immediate resync of inventory information
- notifications sent to an administrator, who then performs or schedules a resync

Although an OSS application must have measures in place to recover from missed events, the OSS application can implement prevention methods. The following are examples of ways an OSS application can prevent missed events:

- Use restrictive filtering wherever possible. See 4.6 “JMS message filtering” (p. 37) for more information.
- Increase the OSS event-processing efficiency.
- Minimize the time that an OSS is subscribed but disconnected from the NFM-P, in which case events are queued but not processed.
- Queue messages internally to facilitate the handling of message bursts.
- Use the DUPS_OK_ACKNOWLEDGE acknowledgment mode to facilitate the handling of network latency. See 4.5.5 “Acknowledgment modes” (p. 36) in 4.5 “JMS subscriptions” (p. 35) for more information.

4.9 Workflow to establish an NFM-P JMS connection

4.9.1 Stages

The following is the sequence of high-level actions required to create and test an NFM-P JMS client connection using JMS sample code to monitor NFM-P events.

1

See 3.8 “Workflow to use the XML API” (p. 31) to set up and operate the XML API.

2

Perform 4.11 “To configure the initial JMS context” (p. 62) to establish the initial JMS context.

3

Perform 4.12 “To compile and connect an NFM-P JMS client” (p. 62) to create a JMS client and monitor an event stream.

4

As required, perform 4.13 “To record and replay a JMS message stream” (p. 64) to record an event stream and replay the stream to test an OSS application.

5

If required, perform 4.14 “To close a JMS client session, or remove a durable subscription” (p. 65) to close a JMS client session.

4.10 NFM-P JMS client configuration and testing

4.10.1 Overview

CAUTION

Service Disruption

The README file for the example code contains important information about implementing a JMS connection.

Ensure that you read the file before you attempt to create a JMS connection.

4.11 “To configure the initial JMS context” (p. 62) describes how to configure the initial context for a JMS client. After you configure the initial JMS context, you can use 4.12 “To compile and connect an NFM-P JMS client” (p. 62) to create a simple JMS client and test a JMS connection using the NFM-P JMS test example code, which is available from the /opt/nsp/nfmp/server/nms/integration/SAM_O/jmsexample directory on a main server.

4.10.2 JMS event recording and replay

You can use an NFM-P utility to record NFM-P JMS messages in a file, and then replay the messages as input for OSS application testing. See 4.13 “To record and replay a JMS message stream” (p. 64) .

4.11 To configure the initial JMS context

4.11.1 Steps

1

Copy the following file from a main server station to the JMS client station:
/opt/nsp/nfmp/server/nms/integration/SAM_O/jmsexample/jndi.properties

2

Open the jndi.properties file on the client station using a plain-text editor.

3

Locate the following line:

```
java.naming.provider.url=remote://YourServerAddress1:1099,remote://YourServerAddress2:1099
```

4

Perform one of the following.

a. For a standalone NFM-P system, edit the line to read:

```
java.naming.provider.url=remote://server_address:1099
```

where *server_address* is the IP address or hostname of the standalone main server

b. For a redundant NFM-P system, edit the line to read:

```
java.naming.provider.url=remote://server_address_1:1099,remote://server_address_2:1099
```

where *server_address_1* and *server_address_2* are the IP addresses or hostnames of the primary and standby main servers

5

Save and close the file.

6

Add the jndi.properties file to the Java class path of the OSS client.

END OF STEPS

4.12 To compile and connect an NFM-P JMS client

4.12.1 Steps

1

Open a console window on the JMS client station.

2

Copy the following file from an NFM-P main server station to the current working directory on the JMS client station:

- /opt/nsp/nfmp/server/nms/integration/SAM_O/jmsexample/JmsTest.java

3

Copy the samOss.jar file from the main server to the client station:

- file system location:
  /opt/nsp/nfmp/server/nms/integration/SAM_O/samOss.jar
- URL:
  TLS enabled—https://*server_address*:8443/integration/samOss.jar
  TLS disabled—http://*server_address*:8080/integration/samOss.jar
  where *server_address* is the main server IP address or hostname

4

Enter the following to compile the JmsTest.java file:

```
javac -classpath .:samOss.jar JmsTest.java
```

5

Enter one of the following to open a JMS connection to the NFM-P system:

Note: The connection is made to the standalone main server, or to the current primary main server in a redundant system. The server addresses are specified in the initial JMS context.
The -persistent parameter specifies a durable subscription, and is optional.

- If TLS is not enabled on the NFM-P XML API:

```
java -classpath path:samOss.
jar JmsTest -t topic -u user -p pass -f "filter" -persistent -c ID
```

- If TLS is enabled on the NFM-P XML API:

```
java -classpath path:samOss.jar -Djavax.net.ssl.trustStore=
truststore JmsTest -t topic -u user -p pass -f "filter" -persistent -c ID
```

where

path is the relative path to the samOss.jar file on the client station, such as ./ or .\

truststore is the absolute path and name of the TLS truststore file on the client station

topic is the JMS subscription topic

user is the NFM-P user name

pass is the NFM-P user password

filter is a JMS filter; see 4.5.4 “Filters” (p. 36) in 4.6 “JMS message filtering” (p. 37)

ID is a JMS client ID that you provide

END OF STEPS

4.13 To record and replay a JMS message stream

WARNING

Equipment Damage

The replay function may affect the NFM-P main server, GUI client, or OSS client performance.

Note: The replay function is a JMS exercise and there is no interaction with the NFM-P system or network configuration.

The tool does not monitor disk capacity; you must stop recording messages to avoid excessive disk consumption.

You can modify the messages and delays by adding or removing attributes.

The delay is measured in milliseconds.

Only the syntax is validated. Invalid syntax prevents replay.

4.13.1 Steps

1

Log in to an NFM-P main server as the nsp user.

2

Open a console window.

3

Enter the following:

```
bash$ cd /opt/nsp/nfmp/server/nms/bin/unsupported ↵
```

4

To start recording JMS messages, enter the following:

```
bash$ ./jmsTools.bash jmsRecorder start filename ↵
```

where *filename* is the absolute path and name of the file in which to store the messages

5

To stop recording messages, enter the following:

```
bash$ ./jmsTools.bash jmsRecorder stop ↵
```

6

To replay the recorded messages, enter the following:

```
bash$ ./jmsTools.bash jmsReplayer filename delay numberOfLoops ↵
```

where

filename is the absolute path and name of the file that contains the recorded messages

delay is Yes or No, to specify whether to include the delays captured during the recording

numberOfLoops is an optional integer value that specifies the number of times to play the stream; if not specified, the stream plays once

END OF STEPS

4.14 To close a JMS client session, or remove a durable subscription

4.14.1 Steps

1

Using an account with an assigned security scope of command role, choose Administration→Security→NFM-P User Security from the NFM-P main menu. The NFM-P User Security - Security Management (Edit) form opens.

2

Click on the Sessions tab.

3

Click Search to list the current user sessions.

4

To close a session, select the session and click Close Session. The NFM-P terminates the session.

If the client subscription is durable, the subscription remains active and the NFM-P continues to process and store JMS messages for the client.

If the client subscription is non-durable, the subscription is removed.

5

To close a session and remove the associated durable subscription, select the session and click Remove Session. The NFM-P stops processing JMS messages for the client.

6

Close the NFM-P User Security - Security Management (Edit) form.

END OF STEPS

4.15 To determine the Java version

4.15.1 Steps

1

Log on to an NFM-P main server station as the nsp user.

2

Open a console window.

3

Enter the following:

```
bash$ cd/opt/nsp/nfmp/server/nms/bin ↵
```

4

Enter the following:

```
bash$ ./nmsserver.bash jvm_version ↵
```

END OF STEPS

5 XML requests

5.1 HTTP communication with the NFM-P

5.1.1 Overview

The XML API is accessible using a servlet that runs on an NFM-P main server. The XML API uses HTTP or HTTPS to transport the XML requests between the client application and the server.

To establish communication with the NFM-P, you must develop HTTP client request messages that include the following components:

- The address of the NFM-P main server, which is the server hostname or IP address, a colon, and the HTTP or HTTPS TCP port.
- A SOAP envelope, which contains one or more XML requests.

After the NFM-P executes a request, it returns a response with a status code that indicates the request success or failure, and the information defined in the request. See 9.1 “XML message structure ” (p. 105) for information about SOAP request, response, and fault messages.

5.2 HTTP POST method

5.2.1 Overview

The XML API supports only the HTTP POST method for client requests. SOAP messages between the OSS and XML API are transmitted in the HTTP message body. For more information about HTTP, see RFC 2616.

5.3 To post an XML request to the NFM-P

Note: You must use a user account with OSS Mgmt permission to access the XML API. See the chapter on NFM-P user security in the *NSP System Administrator Guide* for more information about the NFM-P user accounts and privileges.

You must have an OSS script or application that uses the required HTTP client methods, including POST that is required to send the XML API requests.

You must always close and release the connection regardless of whether the response to your XML request is a success or not. Make sure the connection is not stuck in a CLOSE-WAIT state; use the appropriate HTTP client methods to properly close it.

5.3.1 Steps

1

Specify the HTTP or HTTPS access address.

a. For HTTP access, use:

`http://server_name:port/xmlapi/invoke`

where

server_name is the host name or IP address of the NFM-P main server

port is the TCP port on the main server that is configured for HTTP

b. For HTTPS access, use:

`https://server_name:port/xmlapi/invoke`

where

server_name is the host name or IP address of the NFM-P server

port is the TCP port on the main server that is configured for HTTPS

The XML application servlet is running and available from the appropriate port. The SOAP-enabled XML can now be sent . For more information about the communication between an OSS and a main server, contact Nokia technical support.

2

Create an XML request.

3

Create a script to post the request.

4

Post the request to the XML API using the following CLI syntax:

`OSS_application xml_request.xml`

where

OSS_application is the name of your OSS application that uses the POST method

xml_request is the name of your XML request

The NFM-P sends a response that includes the request execution status and the requested information. See Table 9-3, “HTTP response codes” (p. 116) for information about the HTTP response codes and the corresponding execution status.

END OF STEPS

5.4 Monitoring connection status using XML API ping

5.4.1 Overview

You can use XML API ping to monitor the HTTP connection to the NFM-P.

Note: See Chapter 4, “Event monitoring using JMS” for more information about using the following methods to monitor the status of the connection:

- JMS exception to monitor client connections to the JMS server

- monitoring near-real-time events using JMS to monitor client connections to the JMS server

5.5 XML API ping

5.5.1 Overview

NOTICE

Service-disruption hazard

The following sample message is an example of the request format.

Use the sample as a base to build your request. Ensure that you test your request before network deployment.

The XML API ping method is used to monitor the HTTP connection to the NFM-P.

Figure 5-1, “XML interface check request” (p. 68) shows an XML API ping that you can use to test the ability of the OSS application to access the NFM-P. Information in the SOAP/XML request includes the:

- standard SOAP user and password encoding
- ping command to look for the release of XML on the server, in this case version 1.0

Figure 5-1 XML interface check request

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <security>
            <user>username</user>
            <password hashed="false">password</password>
         </security>
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <ping xmlns="xmlapi_1.0"/>
   </SOAP:Body>
</SOAP:Envelope>
```

The following figure is a code example of a successful response message to the ping request. The response indicates that the correct server XML version, 1.0, responded to the ping.

Figure 5-2 XML interface check response

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
```

```
      <header xmlns="xmlapi_1.0">
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <pingResponse xmlns="xmlapi_1.0"/>
   </SOAP:Body>
</SOAP:Envelope>
```

An exception response to a failed ping may result in numerous types of return messages; for example:

- socket timeouts
- HTTP errors
- connection exceptions

5.6 Workflow to set up and operate an HTTP XML request-response connection

5.6.1 Stages

The following workflow lists the high-level steps required to set up and operate an HTTP XML request-response connection.

1

See 3.8 “Workflow to use the XML API” (p. 31) to set up and operate the XML API.

2

Develop the application that requests data from, or sends data to, the XML API. See 5.3 “To post an XML request to the NFM-P” (p. 67) .

1. Construct the request by specifying:
   - methods that define the executed function
   - package classes used that define the kinds of objects, and the values of the objects (parameters) requested
   - filters to limit or customize the scope of the request
   - information structure and data types
2. Send the request. See 5.3 “To post an XML request to the NFM-P” (p. 67) .

3

Manage the request responses.

a. View the error or success messages.

b. Retrieve the data from the response.

4

Close the connection and release it.

5

Handle the retrieved information appropriately.

5.7 Viewing XML requests ignored by the NFM-P

5.7.1 Overview

If an OSS application includes an element that is not in the schema in the XML request, the element is ignored and the XML API processes the request. All XML requests with matching start and end tags are accepted by the XML API, including spelling and case errors. The NFM-P logs a message to the serverLogFile.

Note: Enabling all logging for the XML API layer is only suitable for development environments.

5.8 XML API system commands

5.8.1 Overview

The XML API provides commands that allow you to do the following:

- obtain the local server time of the request
- identify the NFM-P software release and patch level

5.9 Obtaining the local server time of a request

5.9.1 Overview

You can include the <timeStamp xmlns="xmlapi_1.0"/> element in your request to obtain the processing time for the request in milliseconds. Figure 5-3, “Timestamp and software load request” (p. 72) shows a sample of the <timeStamp xmlns="xmlapi_1.0"/> element and NFM-P-specific version commands used in a request.

Note: The server time for the request in the SOAP response header indicates when a request stream was opened. It does not indicate when a request completed.

You can insert multiple <timeStamp xmlns="xmlapi_1.0"/> elements for a SOAP message that contains multiple message requests. You can use the delta between the timestamp commands to determine the milliseconds required to execute the intermediate commands. See 9.1.3 “Requests” (p. 107) in 9.1 “XML message structure ” (p. 105) for more information about multiple requests.

5.10 Identifying the NFM-P software release and patch level

5.10.1 Overview

You can include the <version xmlns="xmlapi_1.0"/> element within the body of your XML request to determine the software load installed on the NFM-P. The NFM-P returns the software load information in an XML response that has the following format:

```
NFM-P Version <version>.<major.minor>.<build_number>.<patch_number>
```

The NFM-P does not return patch information in the XML response if there are no issued patches for the software.

Figure 5-3 Timestamp and software load request

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <security>
            <user>username</user>
            <password hashed="false">password</password>
         </security>
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <timeStamp xmlns="xmlapi_1.0"/>
      <ping xmlns="xmlapi_1.0"/>
      <timeStamp xmlns="xmlapi_1.0"/>
      <version xmlns="xmlapi_1.0"/>
      <timeStamp xmlns="xmlapi_1.0"/>
   </SOAP:Body>
</SOAP:Envelope>
```

The following figure shows a sample of the response to the request in the previous figure.

Figure 5-4 Timestamp and software load response

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <requestID>XML_API_client@n</requestID>
         <requestTime>Jun 9, 2014 3:44:23 PM</requestTime>
         <responseTime>Jun 9, 2014 3:44:23 PM</responseTime>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <timeStampResponse xmlns="xmlapi_1.0">
         <millis>1402343063516</millis>
      </timeStampResponse>
      <pingResponse xmlns="xmlapi_1.0" />
      <timeStampResponse xmlns="xmlapi_1.0">
         <millis>1402343063517</millis>
      </timeStampResponse>
      <versionResponse xmlns="xmlapi_1.0">
         <version>NFM-P Version R.r.Rx.y</version>
```

```
         <baseVersion>R.r</baseVersion>
         <build>Rx</build>
         <patch>y</patch>
      </versionResponse>
      <timeStampResponse xmlns="xmlapi_1.0">
         <millis>1402343063518</millis>
      </timeStampResponse>
   </SOAP:Body>
</SOAP:Envelope>
```

6 XML API filtering

6.1 XML API filter types

6.1.1 Overview

You can include a filter in an XML request to limit the returned objects to only those that you require. The following filter and result filtering types describe the fundamental elements of filtering objects requested through the XML API. These types are applicable to a variety of methods that perform functions such as inventory requests, alarm retrieval, statistics data retrieval, and object configuration.

The xmlApiTypes.xsd file in the *Schema Reference* describes the 6.1.2 “FilterHolder type” (p. 75) and 6.1.3 “ChildrenFilterHolder type” (p. 76) .

Note: Nokia recommends that you create filters that have a top-down hierarchical structure to make them as restrictive as possible. This means that a filter lists the higher-level object properties before the lower-level properties, as defined in the NFM-P object hierarchy. See the XML API Reference for object hierarchy information.

The SOAP header and footer are shown in code examples only if required to provide context.

6.1.2 FilterHolder type

The FilterHolder type limits or refines the objects returned by an XML API request. The following figure shows a filter example for service.Site objects that have a displayedName value equal to Some special site.

Figure 6-1 Filter example using equal operator

```
<filter class="service.Site">
   <equal name="displayedName" value="Some special site"/>
</filter>
```

The following figure shows a filter example that contains leaf and composite filter elements that use multiple operators to compare numeric and string values.

Figure 6-2 Filter example using multiple operators

```
<filter>
   <and>
      <greater name="localAS" value="0"/>
      <or>
         <equal name="domain" value="network"/>
         <between name="preference" first="150" second="200" />
      </or>
   </and>
</filter>
```

6.1.3 ChildrenFilterHolder type

The ChildrenFilterHolder type is similar to the FilterHolder type; however, the ChildrenFilterHolder type can contain nested children filters that are applied to the children of an object. This type is used for find and findToFile methods.

The attribute “childClass” is required if the attributes used in the children filter are specific to a child class.

Setting the attribute “withChildrenOnly” to “true” returns only objects with children present. Setting the attribute to “false” returns objects of the specified class regardless of child presence. This attribute is optional and is set to “false” by default.

Figure 6-3, “Nested filter example” (p. 76) returns only IES services, IES sites, L3 access interfaces and L3 anti-spoofing object if all filter level criteria are met. If at any level of the nested filter all the children are filtered out, none of parent objects are returned due to the withChildrenOnly equal true.

The first level filter applies to IES services which have a description that starts with “Static”.

The second level filter applies to only the IES sites because of the class specified in childClass attribute. If there exists an IES site that has a siteId in the 198.51.100.0/24 subnet, the IES service is returned.

The third level filter is applied to only L3 access interfaces because of the class specified in childClass attribute. If there exists L3 access interfaces that have a primary IPv4 address in the 10.1.2.0/24 subnet, the IES site is returned.

The fourth level filter applies to the L3 anti-spoofing object because of the class specified in childClass attribute. If there exists L3 anti-spoofing objects with arpPopulate attribute equal disabled, the L3 access interface is returned.

Figure 6-3 Nested filter example

```
<filter class="ies.Ies" >
   <wildcard name="description" value="Static%"/>
   <children>
      <filter class="ies.Ies" childClass="ies.Site" withChildrenOnly="true">
         <wildcard name="siteId" value="198.51.100.%" />
         <children>
            <filter class="ies.Site" childClass="ies.L3AccessInterface"
withChildrenOnly="true" >
               <wildcard name="primaryIPv4Address" value="10.1.2.%"/>
               <children>
                  <filter class="ies.L3AccessInterface" childClass="antispoof.
L3AntiSpoofing" withChildrenOnly="true">
                     <equal name="arpPopulate" value="disabled"/>
                  </filter>
               </children>
            </filter>
         </children>
      </filter>
   </children>
</filter>
```

6.2 Result filtering

6.2.1 Overview

Result filters limit the attributes, objects, and child objects that are returned in a result. If no attribute tags are defined in a result filter, then all properties at the level of resultFilter are returned. For more information, see the xmlApiTypes.xsd file in the Schema Reference.

The <attribute>*propertyName*</attribute> tag specifies that the property is to be returned. When using multiple levels of nested resultFilter, a single empty attribute tag <attribute/> returns no attributes for that object level.

The <children> list of nested resultFilters</children> tag specifies that the given types of children be returned. The nested resultFilters can contain the class attributes specifying the package qualified class name of the child to be returned.

If the children tag is empty, no children are returned. If the children tag is not defined, then all children of this object are returned.

The class attribute on nested child resultFilters allows a class of ManageObject to be specified, which is the base class for all classes.

If the optional recursive attribute is set to yes in the children specification, then the children are filtered according to the same rules as the parent; nested resultFilters are not required or allowed. The recursive attribute default is no. See Figure 6-5, “resultFilter example using recursive children attribute” (p. 78) for an example.

Within a nested class resultFilter, by specifying an empty <attribute/> tag, no attributes are returned for the specified class object. However, children and grandchildren of that class object can be returned by including attribute tags with values. See Figure 6-7, “Nested resultFilter example specifying only child attributes” (p. 79) for an example.

The following table lists some commonly used methods that implement the resultFilter type as a filter input parameter. See the Schema Reference and XML API Reference for information about other methods.

Table 6-1 Commonly used methods type definitions

Filter type	ResultFilter type	Common method name
ChildrenFilterHolder	ResultFilter	find
ChildrenFilterHolder	ResultFilter	findToFile
ChildrenFilterHolder	ResultFilter	registerLogToFile
—	ResultFilter	registerSasLogToFile
Filter-Set	ResultFilter-Set ExtraTags	registerNotification
FilterHolder	ResultFilter	fm.FaultManager.findFault
FilterHolder	ResultFilter	fm.FaultManager.findFaults
—	ResultFilter	generic.GenericObject.configureInstanceWithResult

Table 6-1 Commonly used methods type definitions (continued)

Filter type	ResultFilter type	Common method name
—	ResultFilter	generic.GenericObject.configureChildInstanceWithResult
—	ResultFilter	generic.GenericObject.getDeployer
FilterHolder	ResultFilter	generic.GenericObject.getDeployers

The following figure shows a nested resultFilter that returns the objectFullName and status attributes from the top-level object. The top-level object may have many children but the nested resultFilter specifies only objects of equipment.Port class are returned. Therefore, only the objectFullName attribute of equipment.Port object is returned with no children of its own.

Figure 6-4 Nested resultFilter example

```
<resultFilter>
   <attribute>objectFullName</attribute>
   <attribute>status</attribute>
   <children>
      <resultFilter class="equipment.Port">
         <attribute>objectFullName</attribute>
         <children/>
      </resultFilter>
   </children>
</resultFilter>
```

The following figure shows the same result filter rule applied to returned objects and their children, where only objectFullName is returned for all children.

Figure 6-5 resultFilter example using recursive children attribute

```
<resultFilter>
   <attribute>objectFullName</attribute>
   <children recursive="yes"/>
</resultFilter>
```

The following figure shows a nested resultFilter that returns four attributes from the top-level object along with attributes from two specific children classes.

Figure 6-6 Nested resultFilter example for multiple child classes

```
<resultFilter>
   <attribute>objectFullName</attribute>
   <attribute>displayedName</attribute>
   <attribute>name</attribute>
   <attribute>description</attribute>
   <children>
      <resultFilter class="nqueue.Entry">
         <attribute>objectFullName</attribute>
         <attribute>id</attribute>
```

```
            <attribute>cir</attribute>
            <attribute>pir</attribute>
         </resultFilter>
         <resultFilter class="nqueue.ForwardingClass">
            <attribute>objectFullName</attribute>
            <attribute>forwardingClass</attribute>
            <attribute>queueId</attribute>
         </resultFilter>
      </children>
   </resultFilter>
```

The following figure shows a nested resultFilter with a class definition and no attribute defined. This returns two attributes from the top-level object and attributes from the CardSlot class, but nothing from the Shelf class.

Figure 6-7 Nested resultFilter example specifying only child attributes

```
<resultFilter>
   <attribute>activeManagementIp</attribute>
   <attribute>ipAddress</attribute>
   <children>
      <resultFilter class="equipment.Shelf">
         <attribute/>
         <children>
            <resultFilter class="equipment.CardSlot">
               <attribute>assignedCardSubType</attribute>
               <attribute>cardProtectionRowStatus</attribute>
               <attribute>slotId</attribute>
               <children/>
            </resultFilter>
         </children>
      </resultFilter>
   </children>
</resultFilter>
```

The following figure shows an example of a resultFilter in an object configuration request that uses the generic.GenericObject.configureInstanceWithResult method. See 15.2 “Configuration methods” (p. 202) for information about the generic methods that apply to all object types.

Figure 6-8 resultFilter example using generic method

```
<SOAP:Body>
   <generic.GenericObject.configureInstanceWithResult xmlns="xmlapi_1.0">
      <deployer>immediate</deployer>
      <!-- Network Interface -->
      <distinguishedName>network:10.1.1.223:router-1:ip-interface-5</distinguishedName>

      <includeChildren>false</includeChildren>
      <configInfo>
         <rtr.NetworkInterface>
```

```
            <actionMask>
               <bit>modify</bit>
            </actionMask>
            <!-- vRtrIfUp/vRtrIfDown -->
            <administrativeState>vRtrIfUp</administrativeState>
         </rtr.NetworkInterface>
      </configInfo>
      <resultFilter>
         <attribute>objectFullName</attribute>
         <attribute>displayedName</attribute>
         <children recursive="yes" />
      </resultFilter>
   </generic.GenericObject.configureInstanceWithResult>
</SOAP:Body>
```

6.3 JMS simple message filters

6.3.1 Overview

An NFM-P JMS subscription requires a filter using SQL syntax based on the JMS header properties. See 4.6.4 “Simple JMS message filters” (p. 40) in 4.6 “JMS message filtering” (p. 37) for information.

6.4 JMS advanced message filters

6.4.1 Overview

The following types, which are specific to the registerNotification method, provide advanced filtering for clients that subscribe to the 5620-SAM-topic-xml-filtered topic. The filtering reduces the number of event messages that the NFM-P sends to a client. See 4.6.5 “Advanced JMS message filters” (p. 42) for more information.

6.4.2 filter-set

A filter-Set element can contain multiple filter elements that are processed using the Boolean OR function. If no filters are specified, no events are returned.

Figure 6-9, “Advanced filter example with leaf and composite elements” (p. 81) is an example of an advanced message filter that has two filters in the filter set. Each filter contains leaf and composite filter elements.

The first filter specifies the abstract service.Service class to filter service events, and the filter elements specify events that meet the following criteria:

- subscriber ID associated with the service is 22
- AND
- service ID is greater than 10
- AND
  - Administrative State parameter value is 2 or Partially Down

- OR
- Aggregated Operational State value is 3 or Down
- OR
- Service Tier value is 2

The second filter specifies the forwarding of events associated with any port on card 1 of NE 10.20.40.218 that has a compositeEquipmentState other than:

- 5—equipmentMissing
- 6—equipmentMismatch
- 7—equipmentAdministrativelyDown
- 9—containingEquipmentInTest

Figure 6-9 Advanced filter example with leaf and composite elements

```
<filter-Set>
   <filter class="service.Service">
      <and>
         <equal name="subscriberId" value="22"/>
         <greater name="serviceId" value="10"/>
         <or>
            <equal name="administrativeState" value="2"/>
            <equal name="aggrOperationalState" value="3"/>
            <equal name="tier" value="2"/>
         </or>
      </and>
   </filter>
   <filter>
      <and>
         <wildcard name="objectFullName" value="network:10.20.40.218:shelf-1:
cardSlot-1:card:daughterCardSlot-1:daughterCard:%"/>
         <not>
            <in class="equipment.Equipment" name="compositeEquipmentState"
value="5,6,7,9"/>
         </not>
      </and>
   </filter>
</filter-Set>
```

6.4.3 resultFilter-Set

Result filters are applied to the class properties of the events to restrict the events that are included in an event vessel. A result filter is specified using the resultFilter-Set type in the registerNotification method.

The class tag allows a class of ManagedObject, which is the base class for all classes, to be specified. The class value must be unique in the resultFilter-Set element.

Note: The children element is not valid in a resultFilter-Set element because child attributes are not sent in a JMS message.

Only AttributeValueChangeEvent, ObjectCreationEvent, or ObjectDeletionEvent attributes of an AlarmInfo object may be suppressed in an event vessel when a result filter is defined.

If no AttributeValueChangeEvent, ObjectCreationEvent, or ObjectDeletionEvent attributes of an AlarmInfo object match the result filter, then the entire event is suppressed and is not included in the event vessel. Including the following in a result filter ensures that events are not suppressed because of unmatched attributes:

```
<resultFilter>
    <attribute>objectFullName</attribute>
</resultFilter>
```

The following figure shows the first resultFilter element applied to all ManagedObject classes except for equipment.Port and fm.AlarmObject; only the objectFullName class property is to be returned. The second resultFilter element applies to equipment.Port objects only. The third resultFilter applies to fm.AlarmObject objects only.

Figure 6-10 resultFilter-Set element example

```
<resultFilter-Set>
   <resultFilter>
      <attribute>objectFullName</attribute>
   </resultFilter>
   <resultFilter class="equipment.Port">
      <attribute>objectFullName</attribute>
      <attribute>operationalState</attribute>
      <attribute>administrativeState</attribute>
   </resultFilter>
   <resultFilter class="fm.AlarmObject">
      <attribute>affectedObjectFullName</attribute>
      <attribute>severity</attribute>
      <attribute>operatorAssignedUrgency</attribute>
      <attribute>additionalText</attribute>
   </resultFilter>
</resultFilter-Set>
```

6.4.4 extraTags

By default, JMS event vessels do not include the JMS header properties of the contained events. In order to return event header properties, you must specify the properties using the extraTags type during filter registration. The header properties are then included in the body of each associated event that has the header property.

If a JMS event does not have a specified header property, the property is excluded from the event body. If an eventName value is omitted, the specified JMS header property is returned for each event that has the property.

See Table 4-5, “Filterable event classes” (p. 42) for a list of filterable event classes.

The following figure shows each StatsEvent message is to include the MTOSI_osTime JMS header property in its body. All events are to include the fullClassName property. However, since a StatsEvent message does not have a fullClassName property, it is not included in the StatsEvent message body

Figure 6-11 extraTags element example, registerNotification method

```
<extraTags>
   <tag name="MTOSI_osTime" eventName="StatsEvent" />
   <tag name="fullClassName" />
</extraTags>
```

6.5 Assigning OSS alarm filters from the NFM-P client GUI

6.5.1 Overview

An NFM-P GUI operator with the required user privileges can directly apply public alarm filters created in the NFM-P GUI to OSS clients that subscribe to the 5620-SAM-topic-xml-fault or 5620-SAM-topic-xml-filtered topic. When a filter is created in the NFM-P GUI and applied to an OSS client, that filter immediately applies to the current connected JMS sessions and disconnected durable sessions of the client. See “Alarm management” in the *NSP NFM-P Classic Management User Guide* for more information.

Note: Nokia recommends that JMS clients subscribe to the 5620-SAM-topic-xml-fault or 5620-SAM-topic-xml-filtered topic and listen for XMLFilterChangeEvent notifications. The notification provides information about filters applied using an NFM-P GUI client.

6.6 OAM test result filtering

6.6.1 Overview

OAM test results can be retrieved using the findToFile or registerSasLogToFile method, where Filters and resultFilters can both be applied. Use filters to reduce the volume of retrieved statistics data and resultFilter to specify the returning properties of the test result.

See the previous sections for the filter and resultFilter types.

See 12.11 “Retrieving results” (p. 155) for information about retrieving results.

6.7 Inventory filtering

6.7.1 Overview

The XML API provides methods that allow OSS applications to retrieve required information about network objects. The find and findToFile methods are recommended for retrieving inventory information; each method supports the filter and resultFilter types.

See 13.3 “Inventory retrieval methods” (p. 162) for information about inventory retrieval. See 13.6 “Request filters” (p. 170) for information about equipment filtering.

6.8 Statistics filtering

6.8.1 Overview

Accounting and performance statistics retrieval from the NFM-P can be performed using the findToFile or registerLogToFile method. Each method supports the filter and resultFilter input parameters. Nokia recommends that the filter and resultFilter parameters be used to retrieve required statistics log records and log record properties.

See 14.8 “Statistics retrieval using registerLogToFile” (p. 183) for information about statistics retrieval using registerLogToFile. See 14.9 “Statistics retrieval using findToFile” (p. 189) for information about statistics retrieval using findToFile.

6.9 Configuration management result filtering

6.9.1 Overview

There are two configuration methods that can return all children and grandchildren objects of the configured object that has resultFilter as an input parameter. The configureChildInstanceWithResult method is used to create an object and the configureInstanceWithResult method is used to modify an existing object. Each method support the resultFilter tag, which can be used to return only a subset of attributes for the configured object.

See 15.2 “Configuration methods” (p. 202) for information about configuration methods.

7 XML API information model overview

7.1 XML API information model overview

7.1.1 Overview

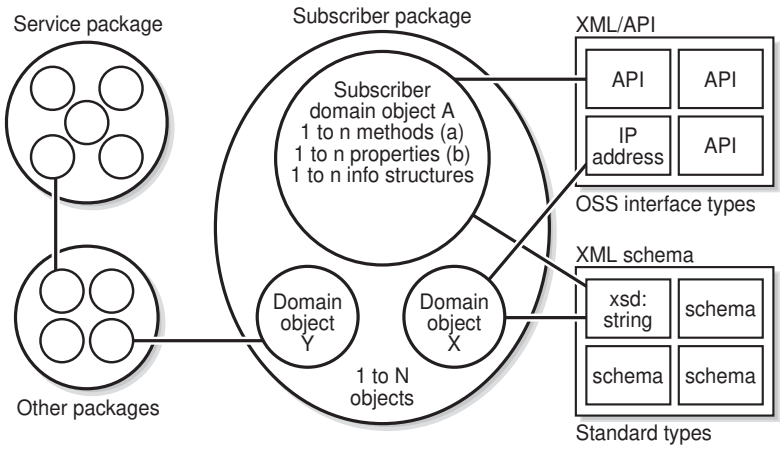
The XML API information model maps to the NFM-P Java code base. The model is defined by an XML schema that is organized in packages.

The information model consists of:

- packages with one or more groups of related domain objects (classes), data types, bitmasks, and enumerations
- classes that contain:
  - properties of the class
  - information structures for both the domain objects (classes) and properties of the information, which are called elements in the XML API but are known as parameters in CLI and the NFM-P GUI
  - NE-specific properties
- relationships between domain objects that indicate the inheritance and containment of these objects
- standard and specific XML schema definitions of types and methods, as well as XML schema definitions specific to NFM-P functions.

The information model provides a type of object model of the NFM-P-managed network. The following figure shows an example of the information model in terms of subscriber configuration.

Figure 7-1 Information model - subscriber package



a) specify method
b) specify properties <inparam... >
Notes: Packages organize related objects under the same name.
Lines represent the relationship between objects.
Circles represent an object or package, the squares represent the types.

17303

More details about the information model can be found in the following:

- XML API Reference—Provides information about the XML interface, including package descriptions, object property values and descriptions, UML inheritance drawings, domain objects, or classes, parent relationships, inherited properties and methods, and supported NE types. The XML API Reference is downloadable from the Documentation Center.
- Schema Reference—Provides a definition of the XML schema for each package, and includes a list of the methods and types that are associated with the package. See the Network Developer Portal to access the Schema Reference.

7.2 Packages

7.2.1 Overview

Each package maps to a functional domain of the NFM-P, for example:

- equipment—fr, netw, equipment, ethernetequipment
- policies—file, qos, acl, slope, accounting
- routing protocols—bgp, ospf, rip, isis
- services—service, subscr, ies, vpls, vll, vprn
- generic—generic, user
- statistics—statistics, log
- diagnostics—sas, ethernetoam

- faults—fm
- database—db
- CPAM—topology, rca, monpath

To view the relationship and inheritance drawing of the package, click on the package name in XML API Reference.

8 XML API Reference

8.1 XML API Reference

8.1.1 Overview

The XML API Reference provides information about the XML API specification and provides information about all of the packages that define the functions on various NFM-P domains that an OSS application can access. The XML API Reference includes the following:

- package descriptions
- object/class property values and descriptions
- UML inheritance drawings
- general methods and types
- deprecated packages, properties, methods, classes, and enumerations
- lists of supported NEs and NE releases
- NE-specific information about supported properties per NE release

Package, class, type, UML inheritance diagrams, property descriptions, and value details are presented on HTML pages and include the following:

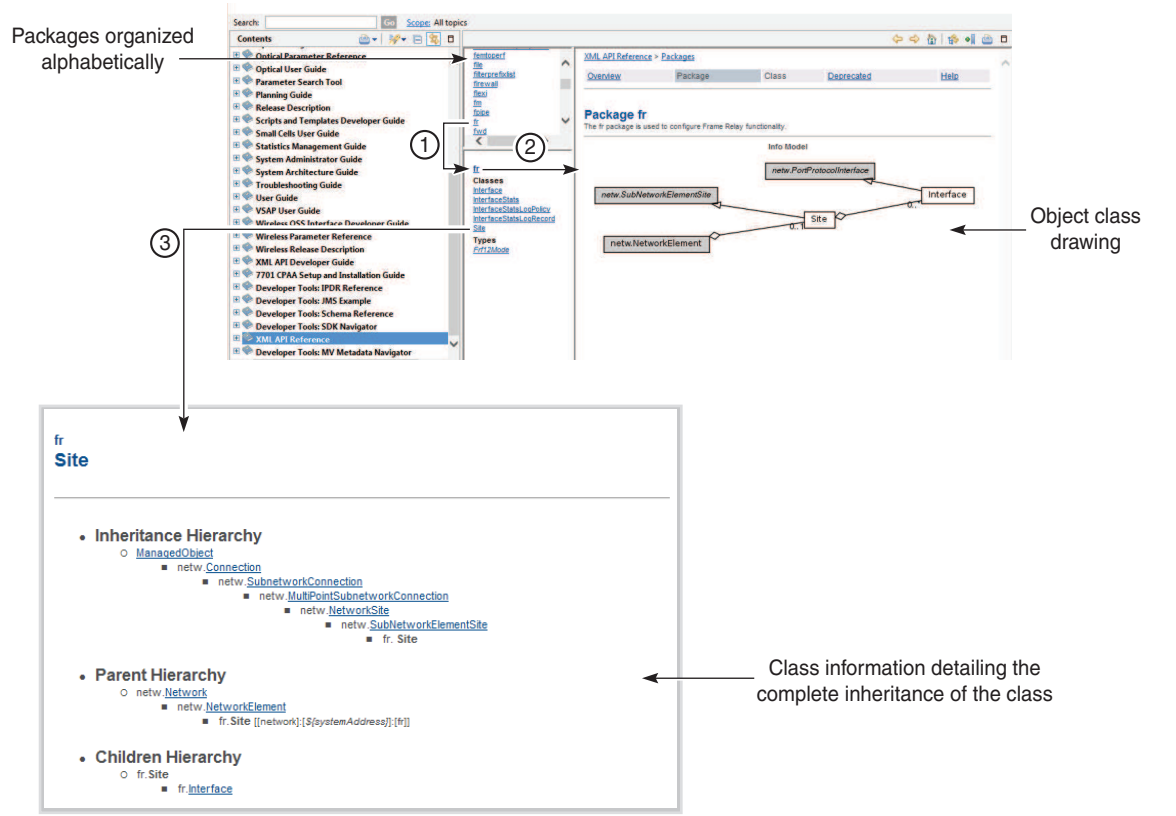
- package list—displays the packages that are available to the XML interface (upper left panel)
- class list—displays the classes, structs, and types that are defined for each package (lower left panel)
- class details—contains information about each class inheritance, relationships, class type, properties, methods, and supported NE types and releases (right panel)

Deprecated properties, general methods/types, and supported products are all available from the upper left panel of the XML API Reference. See 8.12 “To view deprecated schema items” (p. 103) , 8.13 “To view the general methods and types” (p. 104) , and 8.14 “To view the supported device types” (p. 104) for more information.

The XML API Reference is downloadable from the Documentation Center.

The following figure shows the XML API Reference with an XML sample object model.

Figure 8-1 Service object model



8.2 Main menu

8.2.1 Overview

The main menu located at the top of the right panel provides hyperlinks to information in the following areas:

- Overview—lists all of the packages and their descriptions
- Package—provides information about the selected package and its information model diagram
- Class—provides information about the selected class
- Deprecated—lists all of the specifications that are deprecated for a specific NFM-P release
- Help—provides information about how to use and navigate the XML API Reference

8.3 Package list

8.3.1 Overview

After you access the XML API Reference, a table appears. The table includes the full package list and corresponding descriptions of all of the packages that are available using the XML interface. Click on a package link in the table for a general description of the package and the information model diagram.

The package list is also available from the upper left panel of the HTML document. When you choose All Classes/Structs/Types, or a specific package name from the list of packages, the content of the class, struct, and types list area in the lower left panel changes. Each package may contain classes, structs, types, or all.

The information model is a UML drawing that shows the relationships amongst the classes in the selected package and other extended classes from which some properties, methods, and naming are inherited. The classes in the package are represented by white rectangles; the extended classes are represented by gray rectangles. When you click on one of the rectangles, the class information about the selected class is displayed.

Most packages have only one information model drawing. However, some of the larger packages (such as service, equipment, and sas) group member classes into separate information models that represent specific functional areas.

For example, the service package provides specifications for all of the areas that affect service creation, maintenance, configuration, and testing. As a result, the information model for the package is divided into more than one information model to address the following functional areas:

- Access Interface
- Connector and Composite Service
- GNE Info
- Base Service
- Logical Interface
- Operational Group and Encap Group
- STM Service Action
- STM Service Ping
- STM Service Trace
- SAP Policer and Policy Override

The $core package, which is a root or base of all of the packages, defines the base classes and types that all of the other classes use to inherit from. Perform 8.10 “To view release-specific XML schema changes” (p. 102) to view package information in the XML API Reference.

8.4 Classes, structs, and types list

8.4.1 Overview

The class list in the lower left panel lists the classes, structs, and types that are defined in a package. The package name is displayed at the top of the list. Choose the package name to display an overview of the package in the right panel. Choose a class name, struct, or type to display information about each object in the right panel.

8.5 Class details

8.5.1 Overview

Each class has a page that may include the following information:

8.5.2 Description

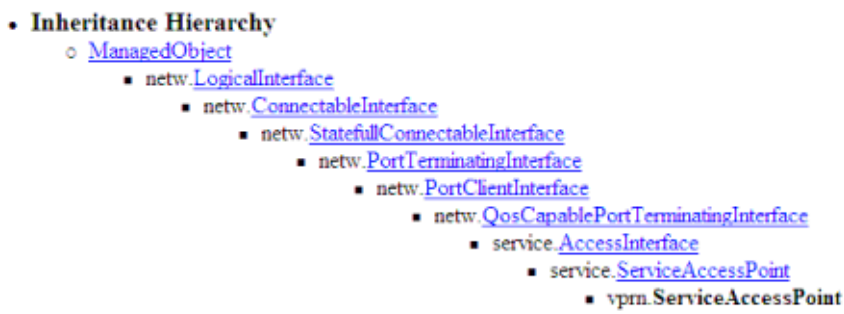
The description provides information about the selected class. For example:

```
vprn
ServiceAccessPoint
A Service Access Point identifies the customer interface point in a Routed CO VPRN
solution.
```

8.5.3 Inheritance hierarchy

The inheritance hierarchy outlines the inheritance tree with the list of all classes that are extended to define a class, as shown in the following figure.

Figure 8-2 Sample inheritance hierarchy



8.5.4 Direct subclass

The direct subclass lists all classes that inherit directly from the selected class, as shown in the following figure.

Figure 8-3 Sample direct subclass

8.5.5 Parent hierarchy

The parent hierarchy indicates where an instance of the class may be located in the containment hierarchy, as shown in Figure 8-4, “Sample parent hierarchy” (p. 93) .

The fully distinguished name, or objectFullName, specifies the NFM-P identifier for the specific object. See 10.1 “Schema Reference ” (p. 127) for more information about object full name.

Figure 8-4 Sample parent hierarchy

8.5.6 Children hierarchy

The children hierarchy lists the types of classes that may be contained in the hierarchy, for example, directly as children or indirectly as descendents, as shown in the following figure.

Figure 8-5 Sample children hierarchy

8.5.7 Relationships

The format of relationships displays the relationship between the current class and other classes. The textual description is of the form:

```
<current-class> may be <relationship-description> <cardinality> <other-class>
(<relationship-type>)
```

where

<relationship-description> is one of the following: associated with, direct parent of, direct children of, contained under, or contain

<cardinality> describes one or many

<relationships-type> is one of the following: parent-child, child-parent, implied, ancestor-descendent, or descendent-ancestor

The following figure shows an example of relationships.

Figure 8-6 Sample relationships

- **Relationships**
  - *vprn.ServiceAccessPoint* may be associated with many *arp.ArpHost*s (implied relationship)
  - one *vprn.ServiceAccessPoint* may be a direct parent of many *clear.ClearRequest*s (parent-child relationship)
  - *vprn.ServiceAccessPoint* may be associated with many *pppoe.PPPoASession*s (implied relationship)
  - *vprn.ServiceAccessPoint* may be associated with many *pppoe.PPPoESession*s (implied relationship)
  - one *vprn.ServiceAccessPoint* may be a direct parent of many *ressubscr.DbInfoSubscriberHost*s (parent-child relationship)
  - one *vprn.ServiceAccessPoint* may be a direct parent of many *ressubscr.HostTrackingInfoPerSap*s (parent-child relationship)
  - one *vprn.ServiceAccessPoint* may be a direct parent of one *ressubscr.SapSubMgmtCfg* (parent-child relationship)
  - *vprn.ServiceAccessPoint* may be associated with many *ressubscr.SubscriberHost*s (implied relationship)
  - many *vprn.ServiceAccessPoint*s may be direct children of one *vprn.GroupInterface* (child-parent relationship)
  - one *vprn.ServiceAccessPoint* may be a direct parent of one *vprn.SapIgmpHostTracking* (parent-child relationship)
  - many *vprn.ServiceAccessPoint*s may be contained under one *vprn.Site* (descendent-ancestor relationship)
  - many *vprn.ServiceAccessPoint*s may be contained under one *vprn.Vprn* (descendent-ancestor relationship)

8.5.8 Class type

The class type is an abstract or a non-abstract class, which is defined as:

```
public [abstract] class name
```

Abstract classes are generally used to provide common property and method inheritance for concrete classes; for example, public abstract class VII. The following is an example of a non-abstract class: public class serviceAccessPoint.

8.5.9 Naming

Objects are named by concatenating the parent and relative names. The relative name of an instance may be defined as: Relative Name=[*name*]. For example:

```
Relative Name=[svc-mgr]
```

Class properties present in the name are identified as: ${propertyName}. For example:

```
Relative Name=[ospf-${version}-instance-${instanceIndex}]
```

Auto-generated IDs are identified with an "*", such as in ${*propertyName}.

8.5.10 Methods that return children

Some methods return children and some methods do not return children. These methods are denoted with the description; for example:

```
methodName() does not return children
```

For example, for netw.NetworkElement:

- configureChildInstanceWithResult() does not return children
- configureInstanceWithResult() does not return children
- findMultipleFilteredInstances() does not return children

8.5.11 Stats

The Stats section lists the set of classes that monitor the class, including inherited statistics classes. The following figure shows an example.

Figure 8-7 Sample stats

Stats for

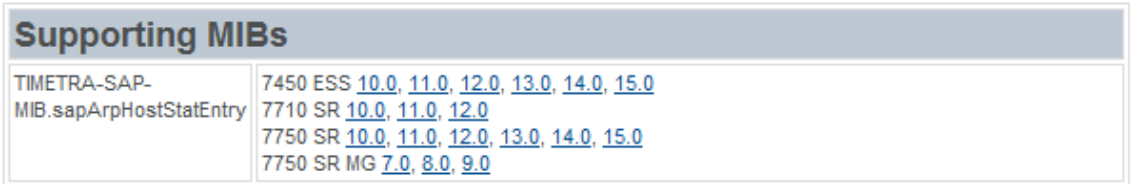
The Stats for section lists the set of classes that the class monitors, as shown in the following figure.

Figure 8-8 Sample stats for

Supporting MIBs

The supporting MIBs section lists the MIB entries that the class reads and the supported device releases for each, as shown in the following figure.

Figure 8-9 Sample supporting MIBs



8.5.12 Properties

The properties are listed alphabetically list of properties displayed for each class. The property definitions may include:

- comment
- type=type-specification— may be a fundamental type or link to another defined type. Children-Set indicates that objects of the specified type may be children of this class.
- default=value
- access=access-type—may be read-only, write-only, read-create, or read-write, which is the default. Read-create values can be set at creation time only.
- Mandatory on create (Indicates the value is required when an object is created)
- minimum=value
- maximum=value
- units=value
- SNMP Deploy/Resync: MIB.entry—defined for read-write access to the NE
- Modification may shutdown object—changing the value may shut down the object. The GUI displays a dialog box for these items.
- Displayed (tab/group)=Displayed (tab/group)—displays the property name and tab location in the GUI. When a property does not have a tab or group information, the property is displayed in the general area.
- enums=enumeration—lists the enumerations when they are defined in a property
- bits—lists the bits when they are defined in a property

The following figure shows an example.

Figure 8-10 Sample properties

Properties	
clear.ClearRequest-Set	*type*=Children-Set
mcLagPropHoldTimeRemain	"The value of mcLagPropHoldTimeRemain indicates the remaining time, in seconds, until MEPs on this SAP will react to a Multi-Chassis LAG protocol or port change. The value zero (0) indicates there are no pending events, or the SAP is not a MC-LAG SAP. This object corresponds to the global configuration timer: TIMETRA-IEEE8021-CFM-MIB::tmnxDot1agCfmMcLagPropHoldTime." *type*=long *access*=read-only *Displayed(tab/group)*=MC-LAG Hold Time (/Facility Meps)
ressubscr.DbInfoSubscriberHost-Set	*type*=Children-Set
ressubscr.HostTrackingInfoPerSap-Set	*type*=Children-Set
ressubscr.SapSubMgmtCfg-Set	*type*=Children-Set
tunnelFaultNotification	The value of tunnelFaultNotification specifies whether the SAP will 'accept' CFM fault notifications from a Tunnel MEP and process the notifications (i.e. do fault handling and/or fault propagation), or 'ignore' the notification. Both TIMETRA-SERV-MIB::svcEthCfmTunnelFaultNotification and this object MUST be set to 'accept' for the SAP to process the notification. The value 'notApplicable' is used by the system to represent a SAP which supports ETH-CFM, but not this object. *type*=service.TunnelFaultNotificationType *default*=accept *Displayed(tab/group)*=Tunnel Fault Notification (/Facility Meps)
vprn.SapIgmpHostTracking-Set	*type*=Children-Set

Overridden properties

A list of all properties that are overridden in this class. A property may be partly overridden and partly inherited through several classes.

Properties inherited from

Inherited and overridden properties may exist with links to definitions in other classes. Properties that are overridden by a specific class are displayed in italics. The following figure shows an example.

Figure 8-11 Sample properties inherited from

Properties inherited from service.ServiceAccessPoint
aaApplicationProfile aaGrpId aaPartId ancp.AncpStaticMap-Set antiSpoofing antispoof.AntiSpoofingStaticHosts-Set callingStationId cpmProtEthCfmMonitorFlags *description displayedName* id *innerEncapValue* macMonitoring memberOperGroupName monitorOperGroupName *outerEncapValue portPointer* sapSubType service.OperGrpBindingMember-Set service.OperGrpBindingMonitor-Set

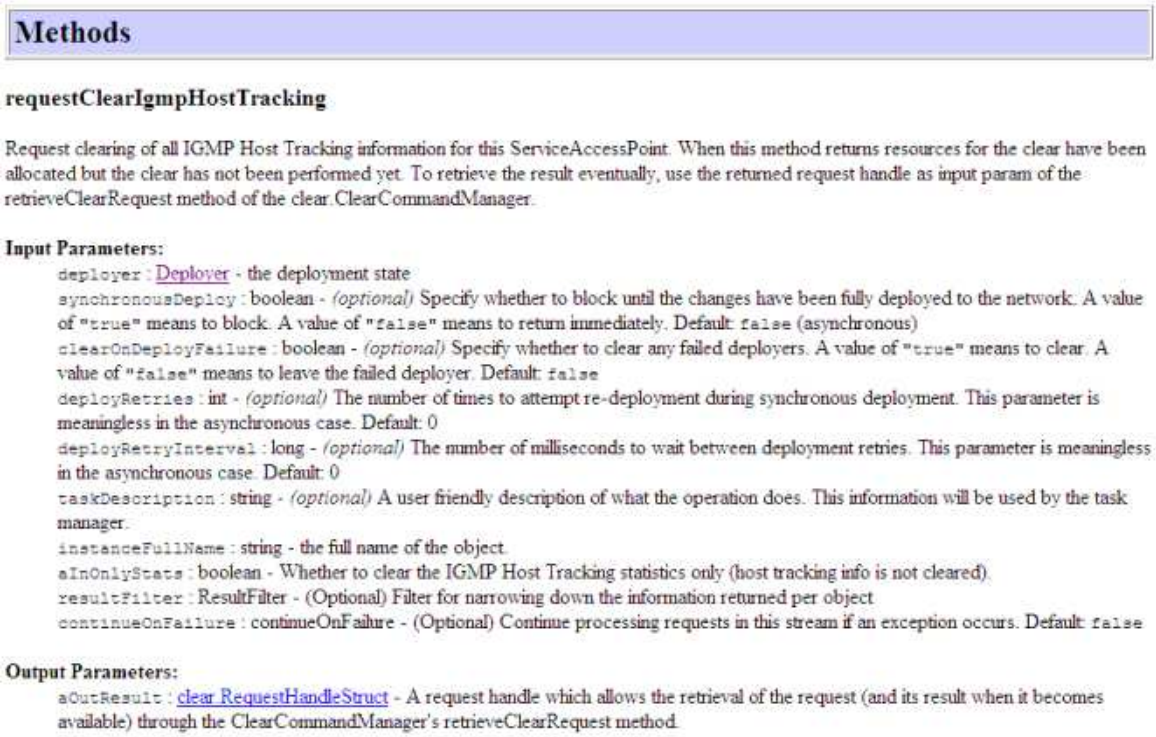
8.5.13 Methods

A list of methods is displayed for each class. The method definitions may include:

- comment
- Input Parameters: name : type [comment]
- Output Parameters: name : type [comment]
- Exceptions: name [comment]

The following figure shows an example.

Figure 8-12 Sample methods



Methods

requestClearIgmpHostTracking

Request clearing of all IGMP Host Tracking information for this ServiceAccessPoint. When this method returns resources for the clear have been allocated but the clear has not been performed yet. To retrieve the result eventually, use the returned request handle as input param of the retrieveClearRequest method of the clear.ClearCommandManager.

Input Parameters:

`deployer` : Deployer - the deployment state
`synchronousDeploy` : boolean - *(optional)* Specify whether to block until the changes have been fully deployed to the network. A value of `"true"` means to block. A value of `"false"` means to return immediately. Default: `false` (asynchronous)
`clearOnDeployFailure` : boolean - *(optional)* Specify whether to clear any failed deployers. A value of `"true"` means to clear. A value of `"false"` means to leave the failed deployer. Default: `false`
`deployRetries` : int - *(optional)* The number of times to attempt re-deployment during synchronous deployment. This parameter is meaningless in the asynchronous case. Default: 0
`deployRetryInterval` : long - *(optional)* The number of milliseconds to wait between deployment retries. This parameter is meaningless in the asynchronous case. Default: 0
`taskDescription` : string - *(optional)* A user friendly description of what the operation does. This information will be used by the task manager.
`instanceFullName` : string - the full name of the object.
`aInOnlyStats` : boolean - Whether to clear the IGMP Host Tracking statistics only (host tracking info is not cleared).
`resultFilter` : ResultFilter - (Optional) Filter for narrowing down the information returned per object
`continueOnFailure` : continueOnFailure - (Optional) Continue processing requests in this stream if an exception occurs. Default: `false`

Output Parameters:

`aOutResult` : clear.RequestHandleStruct - A request handle which allows the retrieval of the request (and its result when it becomes available) through the ClearCommandManager's retrieveClearRequest method.

Methods inherited from

There may be a list of inherited methods with links to definitions in other classes. The following figure shows an example.

Figure 8-13 Sample methods inherited from

Methods inherited from service.AccessInterface

findSitesFor

8.5.14 Supported network elements

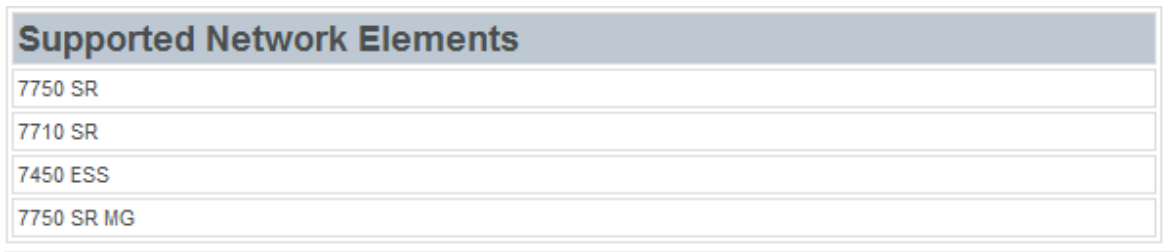
Supported network elements is a list of products and releases that support this class. Specifications may include:

- excluded chassis types chassis-list—not supported for the specified variants of this product
- supported from version—supported for this device for the specified and later releases; there can be multiple entries of this type
- supported for all releases—supported for all device releases

If the supported network elements list does not appear, the class may be supported for any NE.

The following figure shows an example.

Figure 8-14 Sample supported network elements



Product specifics

Product specifics is a list of device- and release-specific properties, such as the following:

- applicable=true or false—whether the property applies
- Excluded Chassis Types=chassis-list—list of chassis that do not support the property
- Supported from=version—supported for the product for the specified and later device releases
- default=value—default value
- access=access type—allowed configuration access, such as read-only, write-only, or read-create
- write-access=yes | no—overrides the write access set using the access attribute in the base property
- minimum=value—minimum value allowed
- maximum=value—maximum value allowed
- suppressedEnums=enumeration—the suppressed enumerations

8.6 Type details

8.6.1 Overview

Each package can also contain complex types. Each complex type has a page that contains the following list of values:

- Enumeration—includes name, value, display-name, whether selectable on GUI, and applicable list of classes
- BitMask—includes bit_name, value, display_name, and whether selectable on the GUI
- Deployer—specifies how and when deployment occurs
- DeploymentState—identifies whether the object is deployed or failed
- List—collection type, ordered collection of values, and duplicates allowed
- Pointer—object reference, expressed using a fully distinguished name
- Map—collection type, unordered key-value mapping, and duplicate keys are not allowed
- Set—collection type, unordered collection of values, and duplicates not allowed
- Children-Set—set of directly contained children for a specific object

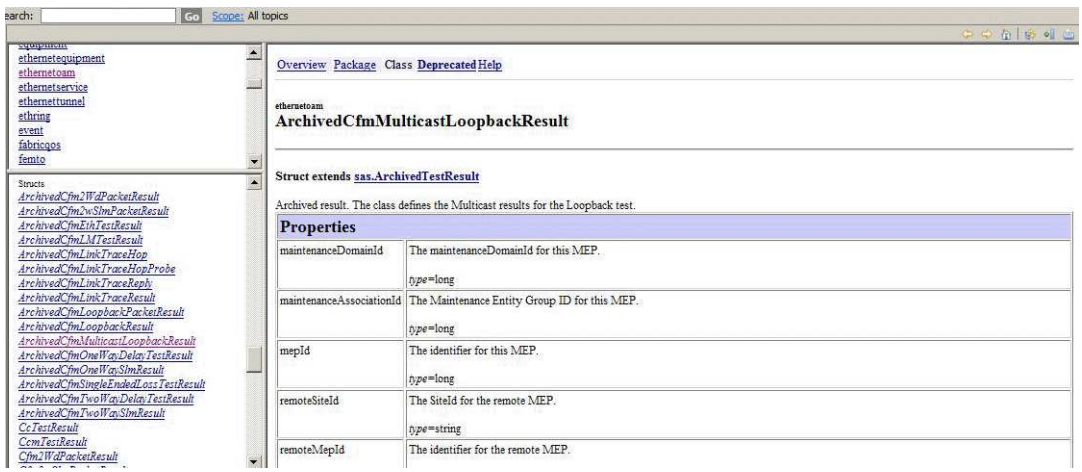
Deployer, DeploymentState, and Children-Set are included in the $core package and can be inherited from other classes.

8.7 Struct details

8.7.1 Overview

Each package can also contain Structs. Each struct has a page that contains struct extension information, description, and a list of property value definitions. The following figure shows an example.

Figure 8-15 Sample struct details



8.8 XML schema changes

8.8.1 Overview

The XML API has evolved over several releases and expanded and changed with the introduction of various network management and network element functions.

8.8.2 Deprecated properties and methods

Some deprecated objects of the XML API are subject to change or to be replaced, are no longer in use, or were never supported. Although deprecated objects remain in the software, Nokia recommends that OSS developers not use them because the deprecated object may not be included in a future release.

8.8.3 Schema changes for releases

The differences between releases of the XML API, including additions, deletions, and changes, are listed at a high level. Changes in the schema can be reviewed by examining details of the interface documentation in each release. Differences can be compared to the methods and objects used in OSS applications to determine whether application updates are required.

See 8.10 “To view release-specific XML schema changes” (p. 102) for information about viewing schema changes between releases.

Note: If you are planning to use an OSS client that was tested in a previous release of the NFM-P, you must review your usage with the list of schema changes before you start development, in order to identify changes that must be made to the packages, classes, types, methods, or properties that are used by the OSS client.

8.8.4 Schema changes for network element versions

Various packages, classes, and properties may apply to a subset of network element products and versions. Values and applicability of properties may change after a network element upgrade or downgrade. OSS applications should be designed to process XML API such that extra or missing properties do not cause unexpected OSS behavior. For example, OSS applications should not expect specific assigned values for properties that are not applicable to a particular product version.

It is recommended that product-specific differences be compared to the methods and objects used in OSS applications to determine whether application updates are required.

See 8.5.14 “Supported network elements” (p. 99) for information about products and releases that support a class. Product-specific details may also apply to various properties.

Note: If you are planning to use an OSS client that was tested in a previous release of the NFM-P or are planning to use different network element product versions, you must review your usage with the list of schema changes before you start development in order to identify changes that must be made to the packages, classes, types, methods, or properties that are used by the OSS client.

8.9 JMS changes

8.9.1 Overview

This section provides a list of hyperlinks to the NFM-P JMS changes over multiple releases. Each hyperlink opens a page that identifies changes to the JMS interface in that release. Changes are divided into two sections: event changes, and interface changes. See 8.11 “To view release-specific JMS changes” (p. 103) for more information about how to view JMS changes between releases.

Note: If you are planning to use an existing OSS client that has been tested in a previous release of the NFM-P, you must review your usage with the list of schema changes before you start development, in order to identify changes that must be made to packages, classes, types, methods, or properties that are used by the OSS client.

8.10 To view release-specific XML schema changes

8.10.1 Steps

1

Open the XML API Reference (it is downloadable from the Documentation Center).

2

Scroll down in the right pane to the Schema Reference Changes section.

3

Click on the “Schema changes for release *R.r*” link, where *R.r* is the release descriptor. The Schema Reference Changes page for the release opens.

The page displays the NFM-P object model changes between the consecutive revisions of the release.

4

As required, click on a link in the Objects Removed/Added/Changed table to view the class or type change information for a specific package.

5

WARNING

Equipment Damage

A schema change may affect an OSS application.

After an NFM-P system upgrade, you must ensure that each OSS application complies with the current object model before you deploy the application. Otherwise, serious system or network damage may result.

Review the information.

6

Close the browser window.

End of steps

8.11 To view release-specific JMS changes

8.11.1 Steps

1

Open the XML API Reference (it is downloadable from the Documentation Center).

2

Scroll down to the JMS Changes section in the right pane.

3

Click on the JMS changes for release *R.r* link, where *R.r* is the release descriptor. The JMS Changes page opens. The page displays the JMS changes between the revisions of the major release.

4

WARNING

Equipment Damage

A JMS change may affect an OSS application.

After an NFM-P system upgrade, you must ensure that each OSS application complies with the current JMS model before you deploy the application, otherwise serious system or network damage may result.

Review the information.

5

Close the browser window.

End of steps

8.12 To view deprecated schema items

8.12.1 Steps

1

Open the XML API Reference (it is downloadable from the Documentation Center).

2

Click on the deprecated properties link in the upper center panel. The Deprecated Packages / Properties / Methods / Classes / Enumerations page opens and displays a list of the deprecated items.

END OF STEPS

8.13 To view the general methods and types

8.13.1 Steps

1

Open the XML API Reference (it is downloadable from the Documentation Center).

2

Click on the general methods/types link in the upper center panel. The General Methods / Types page opens and displays a list of the methods and types that are for general use.

END OF STEPS

8.14 To view the supported device types

8.14.1 Steps

1

Open the XML API Reference (it is downloadable from the Documentation Center).

2

Click on the supported products link in the upper center panel. The General Methods / Types page opens and displays a list of the supported device types for the current release.

END OF STEPS

9 XML message structure

9.1 XML message structure

9.1.1 Overview

The XML API uses request, response, and fault messages to handle the XML message data related to network objects. Messages are sent to the XML API using an RPC pattern. The messages are constructed and wrapped in a SOAP envelope. The XML API then returns a success response message or a failure fault message.

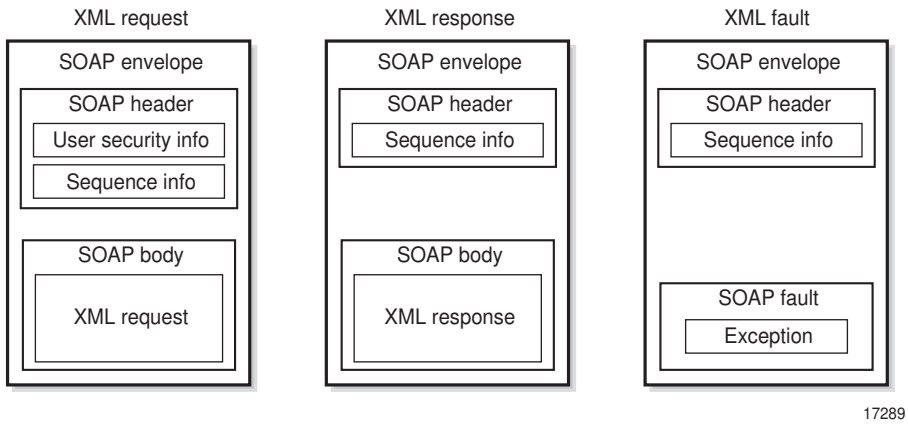
The SOAP encoding transports the XML data. Three types of SOAP messages are used:

- requests
- responses for a successful or unsuccessful execution
- faults for SOAP messages that are not well-formed

9.1.2 Message structure

The following figure shows the structure of the three SOAP message types.

Figure 9-1 SOAP message types



The following table describes the SOAP message types.

Table 9-1 SOAP message type details

SOAP message type	Contents	Details
Request	SOAP envelope	Contains all message data
	SOAP header	The header contains: • user security details: - user IDs in plain-text format - passwords in plain text[1] or an MD5-hashed format. A utility to convert plain text passwords to an MD5-hashed format is available. See 3.4 “Secure communication” (p. 25) for information. • sequence information in the form of a requestID to associate requests with responses and faults. You can use a string for the requestID. The requestID must be unique per HTTP request. Do not use the following formats: - packageName-packageName.ObjectName-[instanceFullNameIfApplicable].methodName. For example: netw-netw.Topology-[].configure - Areq*ObjectNameMethodName*-CLIENT-*userId-sequenceNumber* For example: AreqVirtualInterfaceConfigureIpAddresses-CLIENT-admin-2 - Sreq*ObjectNameMethodName*-CLIENT-*userId-sequenceNumber* For example: SreqTopologyConfigure-CLIENT-operator-12
	SOAP body	Contains the XML request in the general format: <package.object.method> or <internal_method> <inParam1>value</inParam1>[1] <inParamX>value</inParamX>[1] </package.object.method> or </internal_method>
Response	SOAP envelope	Contains all message data
	SOAP header	Contains sequence information in the form of the request ID of the original request
	SOAP body	Contains the XML response in the general format: <package.object.methodResponse> or <package.object.methodException> or <XMLException> or <IMException> <outParam1>value</outParam1> <outParamX>value</outParamX>
Fault	SOAP envelope	Contains all message data
	SOAP header	Contains sequence information in the form of the request ID of the original request
	SOAP fault	Contains: • SOAP fault information • exception details See 9.1.5 “Faults and exceptions” (p. 116) for more information about the fault details.

Notes:

1. For XML string values, there are five special characters that should not be used in plain text. Instead, it is recommended they be replaced with their entity references. See Table 9-2, “Entity references for special characters” (p. 109) for more information.

9.1.3 Requests

The XML API supports the following request types:

- synchronous
- asynchronous (default)

There are three scenarios for a message request:

- The OSS requests a database interaction that causes network deployments, for example, provisioning a service. See Figure 9-2, “Database interaction with synchronous network deployment” (p. 107) and Figure 9-3, “Database interaction with asynchronous network deployment” (p. 108) .
- The OSS requests a database interaction that does not cause network deployments. See Figure 9-4, “Database interaction without network deployment” (p. 108) .
- The OSS requests read information, for example, inventory information or an alarm feed. See Figure 9-5, “Read from database interaction” (p. 109) .

Note: The NFM-P maps the object FDNs to a corresponding database index. Nokia recommends using FDN properties in queries to maximize the processing speed of the query. It is recommended that XML API queries also use concrete classes rather than an abstract class to optimize performance.

Figure 9-2 Database interaction with synchronous network deployment

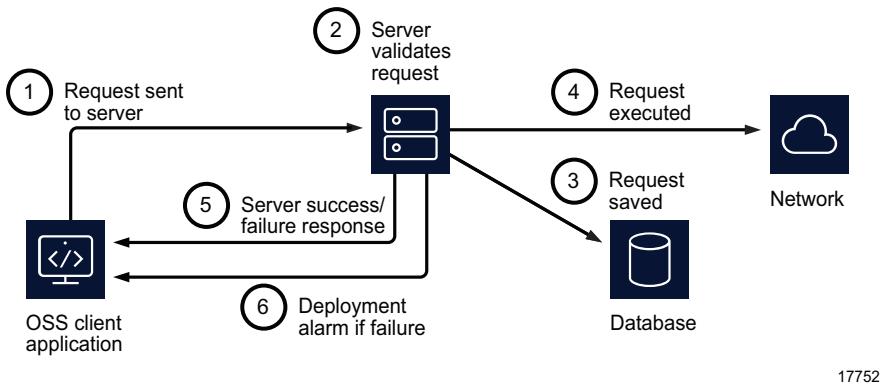


Figure 9-3 Database interaction with asynchronous network deployment

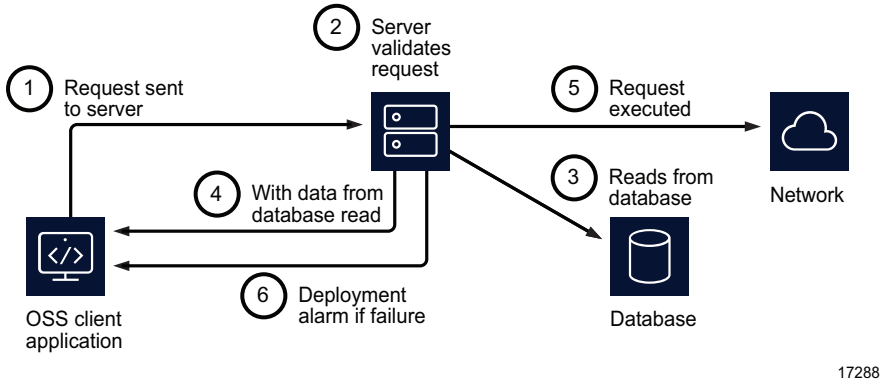


Figure 9-4 Database interaction without network deployment

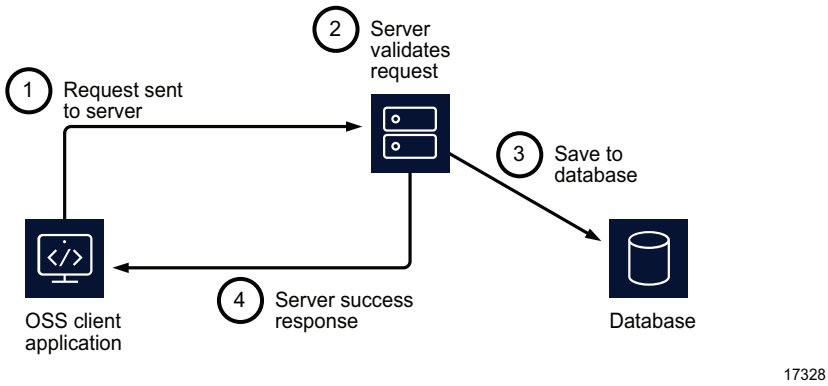
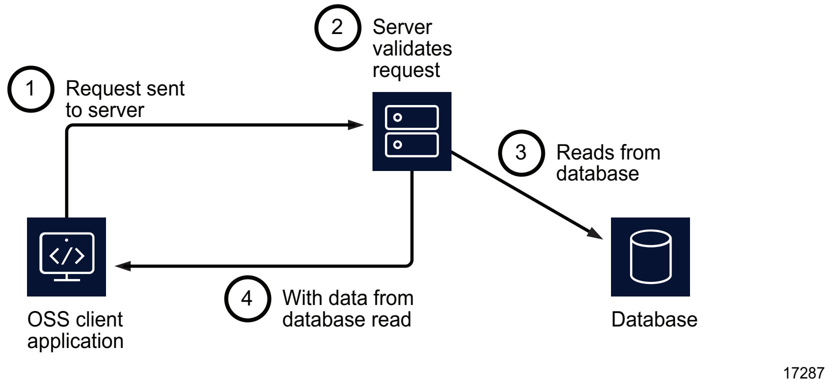


Figure 9-5 Read from database interaction



Network interactions are performed using deployments. When a request fails before being committed to the database, a fault message is sent to the OSS client. For a synchronous request, the XML response indicates the success or failure of deployments. A single request may result in multiple deployments. The success or failure of multiple deployments is returned. OSS clients must be designed to accept more than one deployment in a response.

See 9.1.5 “Faults and exceptions” (p. 116) in this section for more information about deployment failure recovery. See 15.3 “Deployments” (p. 209) for more information about deployment failure JMS alarms and deployment request configurations.

Special characters

There are five special characters used in the XML language that should not be used in plain text. Instead, it is recommended they be replaced with their entity references. The following table lists the special characters and their entity references. The characters < and & are illegal in the XML language, but it is good practice to replace all other characters.

Table 9-2 Entity references for special characters

Special character	Entity reference
<	<
>	>
&	&
‘	'
“	"

Request example

CAUTION

Service Disruption

The request examples are provided to show the request format only, and are intended for use as a template for your requests.

Ensure that you test each request that you create before you deploy the request in a live network.

The XML/SOAP request examples in this guide are intended as a base on which to build your own requests. The sample requests may contain:

- Methods that are deprecated by the XML API
- Configuration information that is not applicable to your network architecture

The request example in the following figure changes the configuration of port 1/1/3 on NE 10.1.202.93 to be access, dot1q, and administratively up.

Figure 9-6 XML request example

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <security>
            <user>username</user>
            <password hashed="false">password</password>
         </security>
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <generic.GenericObject.configureInstanceWithResult xmlns="xmlapi_1.0">
         <deployer>immediate</deployer>
         <distinguishedName>network:10.1.202.93:shelf-1:cardSlot-2:card:daughterCardSlot-
1:daughterCard:port-3</distinguishedName>
         <includeChildren>true</includeChildren>
         <configInfo>
            <equipment.PhysicalPort>
               <actionMask>
                  <bit>modify</bit>
               </actionMask>
               <administrativeState>portInService</administrativeState>
               <mode>access</mode>
               <encapType>qEncap</encapType>
               <children-Set/>
            </equipment.PhysicalPort>
         </configInfo>
      </generic.GenericObject.configureInstanceWithResult>
```

```
   </SOAP:Body>
</SOAP:Envelope>
```

The request includes the following:

- generic.GenericObject.configureInstanceWithResult, which identifies:
  - the use of the generic package
  - the configureInstanceWithResult method defined in the GenericObject class
- the standards-based schema and version identifiers
- the distinguishedName tag, which contains the FDN of the NFM-P object
- the configInfo object, which can contain one or more objects

The configureInstanceWithResult method, as indicated in the request, is defined in the genericMethods.xsd. The equipmentTypes.xsd schema file defines the valid parameters available for configuration on equipment.PhysicalPort. The request is executed by the server according to the parameters specified in the configureInstanceWithResult method.

Note: The actionMask object is of type bitmask.
A bitmask is a small set of Booleans. Each bit is by default false and needs to be explicitly enabled in a single operation. Specifying only a single bit will result in the remaining bits being reset to false even if they were previously enabled.

XML request grouping

You can concatenate multiple message requests in the body of a single SOAP message. The response contains the result of each request.

The following figure shows an example of a SOAP message with multiple message requests.

Figure 9-7 Example of multiple requests in one SOAP message

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <security>
            <user>username</user>
            <password hashed="false">password</password>
         </security>
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <find xmlns="xmlapi_1.0">
         <fullClassName>equipment.DaughterCard</fullClassName>
         <filter>
            <equal name="siteName" value="sim202_93"/>
         </filter>
         <resultFilter>
            <attribute>objectFullName</attribute>
```

```
            <attribute>operationalState</attribute>
            <attribute>administrativeState</attribute>
            <attribute>specificType</attribute>
            <children/>
         </resultFilter>
      </find>
      <find xmlns="xmlapi_1.0">
         <fullClassName>equipment.PhysicalPort</fullClassName>
         <filter>
            <equal name="siteName" value="sim202_93"/>
         </filter>
         <resultFilter>
            <attribute>objectFullName</attribute>
            <attribute>operationalState</attribute>
            <attribute>administrativeState</attribute>
            <attribute>mode</attribute>
            <children/>
         </resultFilter>
      </find>
   </SOAP:Body>
</SOAP:Envelope>
```

Password format

The NFM-P server accepts XML requests with plaintext or MD5-hashed passwords. See 3.4 “Secure communication” (p. 25) for more information.

Action on failure

The <continueOnFailure> and <onFailure> options help manage failed requests.

```
<continueOnFailure>false</continueOnFailure>
```

When the <continueOnFailure> option is set to false and a request in a concatenated message fails, the NFM-P:

- ignores the remaining requests in the concatenated message
- returns responses for successful requests up to the failed request
- generates an exception message for the failure

When the <continueOnFailure> option is set to true, the execution of the requests continues even if a failure occurs. Responses and exceptions are returned for each success or failure.

Note: Execution does not continue if there is a parsing error in the XML request. For example, incorrectly formatted XML or specifying a className or attribute that does not exist results in a parsing error.

The <execute> and <onFailure> options allow requests to recover from a failure. Multiple message requests can be placed within <execute> and <onFailure> blocks. The execution starts with the first request in the <execute> block. If there is a failure, the execution branches to the start of the <onFailure> block. The results are for the requests up to the failed request in the <execute> block, and the requests up to the failed request, if any, in the <onFailure> block. Only one level of nesting

is allowed in an <onFailure> block.

Each <execute> block can have a timeout parameter, as shown in Figure 9-9, “VPLS site creation request example” (p. 112) :

Figure 9-8 execute and onFailure options

```
<execute timeout="500000" xmlns="xmlapi_1.0">
.
.
</execute>
<onFailure>
.
.
</onFailure>
```

Elapsed time is verified between requests in the <execute> block. If the timeout is exceeded, the execution is terminated and a request timed out failure is returned in the result.

The following figure shows a request to create a VPLS site.

Figure 9-9 VPLS site creation request example

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <security>
            <user>username</user>
            <password hashed="false">password</password>
         </security>
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
         <distinguishedName>svc-mgr:service-67</distinguishedName>
         <childConfigInfo>
            <vpls.Site>
               <actionMask>
                  <bit>create</bit>
                  <bit>modify</bit>
               </actionMask>
               <siteId>10.1.241.69</siteId>
               <displayedName>Customer_12:VPLS:100020:site_14</displayedName>
               <administrativeState>1</administrativeState>
               <mtu>0</mtu>
            </vpls.Site>
         </childConfigInfo>
      </generic.GenericObject.configureChildInstance>
```

```
   </SOAP:Body>
</SOAP:Envelope>
```

The following figure shows a deletion request that can be used after a failure occurs when creating the VPLS site.

Figure 9-10 VPLS site deletion request example

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <security>
            <user>username</user>
            <password hashed="false">password</password>
         </security>
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <generic.GenericObject.deleteInstance xmlns="xmlapi_1.0">
         <distinguishedName>svc-mgr:service-67:10.1.241.69
         </distinguishedName>
      </generic.GenericObject.deleteInstance>
   </SOAP:Body>
</SOAP:Envelope>
```

You can combine both requests in Figure 9-9, “VPLS site creation request example” (p. 113) and Figure 9-10, “VPLS site deletion request example” (p. 114) with the execute and onFailure blocks, as shown in the following figure.

Figure 9-11 Request example with execute and onFailure elements

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <security>
            <user>username</user>
            <password hashed="false">password</password>
         </security>
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <execute>
         <generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
            <distinguishedName>svc-mgr:service-67</distinguishedName>
            <childConfigInfo>
               <vpls.Site>
```

```
                  <actionMask>
                     <bit>create</bit>
                     <bit>modify</bit>
                  </actionMask>
                  <siteId>10.1.241.69</siteId>
                  <displayedName>Customer_12:VPLS:100020:site_14</displayedName>
                  <administrativeState>1</administrativeState>
                  <mtu>0</mtu>
               </vpls.Site>
            </childConfigInfo>
         </generic.GenericObject.configureChildInstance>
      </execute>
      <onFailure>
         <generic.GenericObject.deleteInstance xmlns="xmlapi_1.0">
            <distinguishedName>svc-mgr:service-67:10.1.241.69</distinguishedName>
         </generic.GenericObject.deleteInstance>
      </onFailure>
   </SOAP:Body>
</SOAP:Envelope>
```

9.1.4 Responses

Responses identify the execution of a method.

The following figure shows a successful response to the request to configure a network interface.

Figure 9-12 Request success response example

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <requestID>XML_API_client@n</requestID>
      </header xmlns="xmlapi_1.0">
   </SOAP:Header>
   <SOAP:Body>
      <rtr.RoutingInstanceSite.configureResponse xmlns="xmlapi_1.0">
         <objectFullName>network:10.1.202.95:router-1:ip-interface-24</objectFullName>

      </rtr.RoutingInstanceSite.configureResponse>
   </SOAP:Body>
</SOAP:Envelope>
```

You can also include the content type information in the HTTP response header, using the <xmlapi> element in the main server configuration. The xmlContentType parameter defines the content; for example, using the default value of text/xml; charset=ISO-8859-1. You can use any valid string to configure the parameter.

The customProperty fields that can be configured for the network element object, support UTF-8 with the following limitations:

- 4-byte UTF-8 characters are supported for customProperty fields in HTTP requests via the XML

API and also supported in the NFM-P, however, 4-byte UTF-8 characters are not displayed in HTTP responses.

- Certain 4-byte characters are in the supplementary range of the UTF-8 character set and are handled differently. Specifically, any characters above the 0x0FFFF mark are represented as pairs of two 2-byte characters which are not able to be inserted into XML without using escape characters and code point values. These supplementary 4-byte characters are not supported in requests or responses.

HTTP response codes

The following table defines the HTTP response codes based on the RFC 2616 standard. The XML API does not use all responses defined by the RFC 2616 standard. The HTTP response codes are associated with the successful or unsuccessful transmission of the XML response.

Table 9-3 HTTP response codes

Code Range	Definition
1xx [1]	Informational - Request received, continuing process.
2xx	Success - The action was successfully received, understood, and accepted.
3xx [1]	Redirection - Further action must be taken in order to complete the request.
4xx	Client Error - The request contains bad syntax or cannot be fulfilled.
5xx	Server Error - The server failed to fulfill an apparently valid request.

Notes:

1. The XML API does not send 1xx and 3xx messages.

The XML API client must check the HTTP response code before it attempts to parse the XML API results. An HTTP response code between 200 and 299 identifies successful execution of the request. The HTTP body for a 2xx response contains a valid XML SOAP response. All other response codes may not contain well-formed XML.

9.1.5 Faults and exceptions

The XML API produces a SOAP fault message when there is a problem processing a SOAP header or empty SOAP body. The subsequent errors result in an exception within the SOAP body.

SOAP exception messages can contain the following fault-related details:

- <faultcode> provides some standard SOAP information, including:
  - version mismatch between SOAP versions
  - server problems
  - client problems, for example, sending a request to execute a method that does not exist on the server
- <faultstring> error message
- <faultactor> is the URL of the server if the server is the problem

The XML API uses the following format for the <faultstring> element:

```
<faultstring>[error_string] message</faultstring>
```

The following table lists the error types that are associated with the <faultstring> element.

Table 9-4 <faultstring> error types for SOAP fault messages

[error_string]	Details
license	Not licensed to use the XML API
soap	Incomplete or incorrect SOAP construction
xml-header	XML API header is missing, or is missing necessary information, such as the user ID or the MD5-hashed password
security	Authentication error, for example, invalid user ID

The following table describes the faults and exceptions that can appear in an XML response.

Table 9-5 Summary of XML response faults and exceptions

Fault or exception	Format	Description

SOAPFault	`<SOAP:Fault>`
	`<faultcode>value</faultcode>`
	`<faultstring>value</faultstring>`
	`<faultactor>value</faultactor><detail>`
	`<requestID>value</requestID></detail>`
	`</SOAP:Fault>`
		Errors in a SOAP header.
		The <faultstring> in a <SOAP:Fault> response provides details on the fault.
		The strings may change over different releases of the NFM-P and therefore it is recommended they not be parsed by the OSS application.

XMLException	`<XMLException xmlns="xmlapi_1.0">`
	`<description>value</description>`
	`<line>value</line>`
	`<column>value</column>`
	`</XMLException>`
		Errors in the XML syntax or because of a non-existent method or attribute in the XML API.
		The XMLException <description> field provides details on the XML errors.
		The strings may change over different releases of the NFM-P and therefore it is recommended they not be parsed by the OSS application.

IMException	`<IMException>`
	`<cause>value</cause>`
	`<description>value</description>`
	`</IMException`
		Errors in the object model. For example, binding a SAP to a non-existent tunnel.
		The <description> field provides details about the error.
		The strings may change over different releases of the NFM-P and therefore it is recommended they not be parsed by the OSS application.

BaseException	`<BaseException xmlns="xmlapi_1.0">`
	`<description>value</description>`
	`</BaseException>`
		Operation-related exceptions, such as configurationException and creationException. The BaseException appears in the XML response in these exceptions.
		The <description> field in the XML response provides details about the error.
		The strings may change over different releases of the NFM-P and therefore it is recommended they not be parsed by the OSS application.

Table 9-6, “Sample faults and exceptions” (p. 117) lists sample faults and exceptions that can appear in an XML response, as described in Table 9-5, “Summary of XML response faults and exceptions” (p. 117) .

Note: The code examples in Table 9-6, “Sample faults and exceptions” (p. 117) do not contain the SOAP envelope or header information.

Table 9-6 Sample faults and exceptions

Fault or exception	Example fault or exception

SOAPFault	`<SOAP:Fault>`
	`<faultcode>SOAP:Client</faultcode>`
	`<faultstring>[security] Users require OSS Management privileges to use XML API</faultstring>`
	`<faultactor>XmlApi</faultactor>`
	`<detail>`
	`<requestID>XML_API_client@n</requestID>`
	`</detail>`
	`</SOAP:Fault>`

XMLException	`<SOAP:Body>`
	`<XMLException xmlns="xmlapi_1.0">`
	`<description>Command 'fm.FaultManager.findFault2s' is not defined</description>`
	`<line>4</line>`
	`<column>57</column>`
	`</XMLException>`
	`</SOAP:Body>`

IMException	`<SOAP:Body>`
	`<IMException xmlns="xmlapi_1.0">`
	`<cause>java.lang.SecurityException</cause>`
	`<description>SecurityManager:Security Breach: No such user: 'oss_client' </description>`
	`</IMException>`
	`</SOAP:Body>`

BaseException	`<SOAP:Body>`
	`<script.ScriptManager.configureException xmlns="xmlapi_1.0">`
	`<description>[app: script] [class: script.Script] [instance: -unknown-] [descr: the NE type is invalid type,{neType=craig}] </description>`
	`</script.ScriptManager.configureException>`
	`</SOAP:Body>`

Figure 9-13, “Exception message example, configuration failure” (p. 119) shows an exception message generated when a user does not have the required OSS management privileges to use XML API. The sample exception provides the following details:

- <faultcode> element indicates that the server could not execute the request because of a fault of the client
- <faultstring> element provides the security error string, which indicates that the Client user specified does not have the required OSS management privileges to use XML API
- <faultactor> element identifies the XmlApi as the fault factor

Figure 9-13 Exception message example, configuration failure

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <requestID>XML_API_client@n</requestID>
         <requestTime>Jan 2, 2007 1:18:53 PM</requestTime>
         <responseTime>Jan 2, 2007 1:18:53 PM</responseTime>
      </header>
   </SOAP:Header>
   <SOAP:Fault>
      <faultcode>SOAP:Client</faultcode>
      <faultstring>[security] Users require OSS Management privileges to use XML
API</faultstring>
      <faultactor>XmlApi</faultactor>
      <detail>
         <requestID>XML_API_client@n</requestID>
      </detail>
   </SOAP:Fault>
</SOAP:Envelope>
```

Note: The <responseTime> tag in an XML API header is the time at which the response stream is opened.

The <detail> element in the SOAP exception message identifies the command that caused the request failure.

The XML API validates the XML request when it is sent to the server. If the validation fails, an exception message is generated and sent, as shown in Figure 9-13, “Exception message example, configuration failure” (p. 119) . If the request requires changes to the managed network, deployments are created and queued to send the changes to the routers. If the deployment fails, an alarm is raised which includes details for the particular deployment ID. The network can be in an indeterminate state because another configuration request may have succeeded. See 15.3 “Deployments” (p. 209) for more information about creating and viewing deployment requests and alarms.

To recover from request errors, the OSS application may need to check the affected objects in the deployment and issue appropriate requests to undo the action that caused the error. Another option is to manually maintain a list of failures for resolution at a later date.

Note: See 15.4 “Workflow to handle deployment failures” (p. 216) in Chapter 15, “Configuration management overview” for more information about how to handle deployment errors that may cause the NFM-P database to be out of synchronization with the managed network.

Logged exception messages

The EmsServer log files on an NFM-P main server contain additional exception message information. You can view the file contents using a text editor or the LogViewer application. See the

NSP Troubleshooting Guide for information about using LogViewer. See 9.3 “Mapping XML methods to GUI operations” (p. 123) for information about enabling debug logging and viewing EmsServer log files.

9.1.6 Logging XML requests, responses, and exceptions

CAUTION

Service Disruption

The prolonged use of XML message logging in a busy network environment may have network management performance impacts.

Use the log functions for a limited time to debug a specific problem.

You can enable the logging of XML API request, response, and exception messages on an NFM-P main server. The entries in the message logs can assist you with the following:

- identifying the user associated with an XML request
- debugging the OSS requests from a user

You can enable XML message logging for an individual user or multiple users. See C.7 “Identifying XML messages from specific users” (p. 363) for more information. The NFM-P creates log files for each HTTP request and response associated with the user defined by the <systemOssiLog> element in the NFM-P main server configuration. An HTTP request and response can contain multiple XML requests and responses.

The NFM-P stores the log file in the directory defined by the <systemOssiLog> element in the main server configuration, typically .../log/. The NFM-P uses the following naming convention for log files:

- ossi*user*Request*n*.log
- ossi*user*Response*n*.log

where

user is the NFM-P user name

n is the incremental request number

The request log contains the body of the SOAP message. The response log contains the entire SOAP envelope of the response.

The following figure shows a sample request for an XML message log file.

Figure 9-14 Request example, ossiuserRequest1.log file

```
<?xml version="1.0" encoding="UTF-8"?>
<Request user="username" requestId="XML_API_client@n">
   <find>
      <fullClassName>...</fullClassName>
      <filter>
.
.
.
```

```
      </filter>
   </find>
</Request>
```

The following figure shows an example of a response to an XML message log file request.

Figure 9-15 Response example, ossiuserResponse1.log file

```
<?xml version="1.0" encoding="UTF-8"?>
<Response user="user" requestId="XML_API_client@n">
   <SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP:Header>
         <header xmlns="xmlapi_1.0">
            <requestID>XML_API_client@n</requestID>
            <requestTime>Jan 2, 2007 3:10:41 PM</requestTime>
            <responseTime>Jan 2, 2007 3:10:41 PM</responseTime>
         </header>
      </SOAP:Header>
      <SOAP:Body>
         <findResponse xmlns="xmlapi_1.0">
            <result>
.
.
.
            </result>
         </findResponse>
      </SOAP:Body>
   </SOAP:Envelope>
</Response>
```

Note: The <responseTime> tag in the header is the time at which the response stream is opened.

The XML message logging parameters are defined in the NFM-P main server configuration. The system administrator is responsible for the maintenance and backup of log files.

To modify the main server configuration, contact Nokia technical support.

As part of debug logging policy configuration, the system administrator can configure the maximum disk space for storing log messages. The NFM-P disables the log option and raises an alarm if the size of the log file exceeds the storage allocation. The default log file retention period is 24 hours. See C.7 “Identifying XML messages from specific users” (p. 363) for information about how to modify the log file retention period.

In a redundant system, the NFM-P logs the activities for the primary main server.

See C.7 “Identifying XML messages from specific users” (p. 363) for information about how to enable the logging of NFM-P XML message transactions.

The User Activity form in the NFM-P client GUI displays user activity. See the section on user activity logging in the *NSP System Administrator Guide* for more information.

9.2 CLI command methods

9.2.1 Overview

An OSS client can send CLI commands to an NE using the methods listed in Table 9-7, “CLI command methods” (p. 121) . The NFM-P uses the CLI credentials in the associated mediation policy to gain CLI access. For some devices, you can use a site user profile to restrict the CLI commands that are available to a user.

Before executing the commands in a CLI script, the NFM-P executes the “environment no more” command to disable CLI output pagination. You must ensure that the “environment more” command is permitted in the site user profile.

Table 9-7 CLI command methods

Method	Description
executeCli	Sends a CLI command to an NE; see Figure 9-16, “Single CLI command execution request example” (p. 122) .
executeMultiCli	Sends multiple CLI commands to an NE; see Figure 9-17, “Multiple CLI command execution request example” (p. 122) .

Figure 9-16 Single CLI command execution request example

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <security>
            <user>username</user>
            <password hashed="false">password</password>
         </security>
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <netw.NetworkElement.executeCli xmlns="xmlapi_1.0">
         <instanceFullName>network:10.1.186.183</instanceFullName>
         <command>ping 192.168.186.185</command>
      </netw.NetworkElement.executeCli>
   </SOAP:Body>
</SOAP:Envelope>
```

Figure 9-17 Multiple CLI command execution request example

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <security>
```

```
            <user>username</user>
            <password hashed="false">password</password>
         </security>
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <netw.NetworkElement.executeMultiCli xmlns="xmlapi_1.0">
         <instanceFullName>network:35.121.20.56</instanceFullName>
         <commands>  <string>environment no more</string> <string>show system
info</string>
         </commands>
      </netw.NetworkElement.executeMultiCli>
   </SOAP:Body>
</SOAP:Envelope>
```

9.3 Mapping XML methods to GUI operations

9.3.1 Overview

Mapping XML methods to GUI actions helps developers understand which classes and methods are required to integrate an OSS application. In development environments, XML methods can be mapped to GUI actions in the following ways:

- by viewing the User Activity logs in the GUI to identify class and method information in XML format
- by configuring the server log file to record classes and methods called when a GUI action is performed

9.4 To view GUI operations in the NFM-P User Activity log

9.4.1 Steps

1

Perform an action using the GUI.

2

Choose Adiminstration→ Security→User Activity from the NFM-P main menu. The User Activity window opens.

3

Select the Activity tab and click on the Search button. A list of activity logs is displayed.

4

Locate the activity log entry that is relevant to the action performed, based on the time or sub-type, and double-click on the log entry to view it.

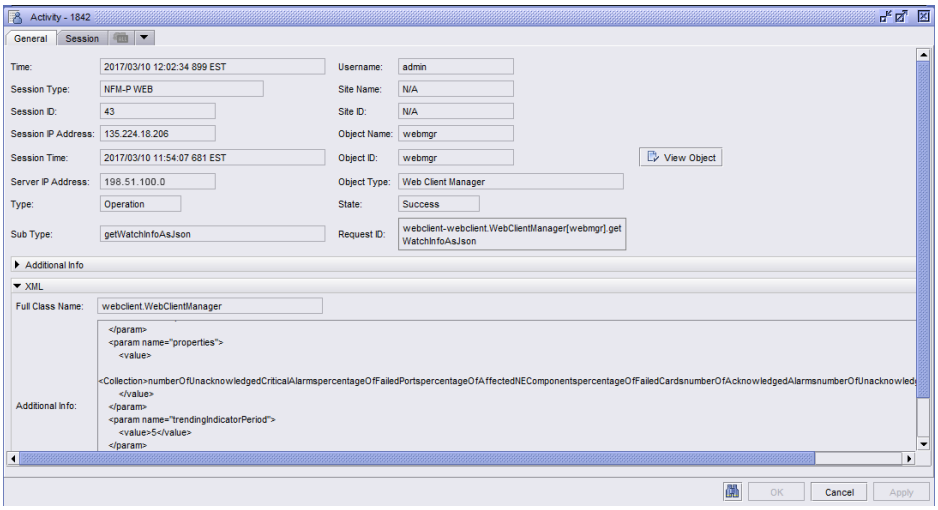
END OF STEPS

9.5 User activity log fields

9.5.1 Overview

The following figure shows the information in an activity log entry for a user configuring an accounting policy through the GUI. The XML and Additional Info panels show object class and attribute details for the activity. XML details, for example, the distinguishedName value, can help developers compose SOAP requests for object creation, modification, and deletion.

Figure 9-18 User activity log fields



9.6 To enable logging of GUI operations on the NFM-P server

CAUTION

Service Disruption

It is recommended that the logging of GUI operations be enabled only on an NFM-P system in a lab or development environment.

Enabling logging on an NFM-P system in a live network can seriously degrade server performance.

Contact your Nokia technical support representative before you attempt to modify the nms-server.xml file. Modifying the nms-server.xml file can have serious consequences that can include service disruption.

9.6.1 Steps

1

Log in to the NFM-P main server station as the nsp user.

2

Open the /opt/nsp/nfmp/server/nms/config/nms-server.xml file using a plain-text editor.

3

Locate the <Filter> tag in the <systemLog> section of the file.

4

Add the following line directly after the line that begins with <!-- Don’t change this section END -->.

```
<include severity="debug" class=".*IFGBase" method="printDebug.*"/>
```

5

Save and close the nms-server.xml file.

6

Open a console window.

7

Navigate to the /opt/nsp/nfmp/server/nms/bin directory.

8

Enter the following at the console prompt to restart the NFM-P server:

```
bash$ ./nmsserver.bash force_restart ↵
```

The NFM-P main server restarts, and all subsequent GUI and OSS actions are logged in the /opt/nsp/nfmp/server/nms/log/server EmsServer.log file.

END OF STEPS

9.7 To view the GUI operation in the server log

9.7.1 Steps

1

Complete 9.6 “To enable logging of GUI operations on the NFM-P server” (p. 124) .

2

Open the server log file.

3

Perform an action on the GUI.

4

View the logged information for that action in the server log file to determine the class and method used.

END OF STEPS

9.8 Log file entries for GUI operations

9.8.1 Overview

The following is an example of a server log entry for the deletion of an IP interface using the client GUI:

```
<server.core.IFGBase.printDebug> IFG[generic]:([generic.GenericObject].deleteInstance() : BEGIN
<server.core.NullDeployer.issueDatabaseLog> From RIWorker [8](1301414107317),Datab
aseLogRuleInvokationCapsule User Id: securityManager:user-admin Request Id:
 AreqGenericObjectDeleteInstance-CLIENT-admin-NFM-P@6-181 Target Class: rtr.
DirectInterfaceCtp Target Object Id: network:198.51.100.59:router-1:ip-interface-2:TP Operation: Obje
ctDeletion Result: true
<server.core.IFGBase.printDebug> IFG[generic]:([generic.GenericObject].deleteInstance() : END ::: suc
cessfuly executed : true
```

The log entry contains the following information:

- the class and method that corresponds to the GUI action, which in this example is generic.GenericObject.deleteInstance. You can use the identified class and method to create an OSS application that performs the same task.
- the specific instance of the action that was on the managed router IP interface with the unique ID network:198.51.100.59:router-1:ip-interface-2:TP. The unique ID identifies that the router has an IP address of 198.51.100.59 and a name of router-1.
- acknowledgement of successful completion of the configuration action

Note: In some cases, a GUI action may use custom methods that are not directly applicable for use by an OSS. For example, when configuring network objects using the GUI, the server log output could log the mpls.LspPath.configure method. In this case an OSS should use the generic method generic.GenericObject.configureInstance. See 15.2 “Configuration methods” (p. 202) for more information about using generic methods for configuring network objects.

In some cases, no specific instance is returned, for example, when you perform a find action.

10 Schema Reference

10.1 *Schema Reference*

10.1.1 Overview

The Schema Reference describes the structure and allowed elements of an XML document sent to the NFM-P, such as a configuration or information request. A developer can use the schema to determine the valid elements, data types, and parameters of an NFM-P functional area, or package.

10.1.2 Package-specific schema files

The following schema files are specific to a package:

- *package*Methods.xsd—describes the methods that are available to the package
- *package*Types.xsd—describes the properties of each object, or class, in the package, the package information structure, and the values assigned to complex types such as bitmask and enumeration

10.1.3 Viewing the schema of a specific XML package

See the Network Developer Portal to access the Schema Reference. The schema page lists the method and data type files for each package.

Click on the *package*Methods.xsd or *package*Types.xsd link, as required, where *package* is the package of interest. The schema information is displayed.

10.2 General schema files

10.2.1 Overview

The following general schema files describe the top level of the XML API hierarchy:

- xmlApi.xsd—lists the schema files for all packages
- xmlApiMethods.xsd—describes the general methods, such as find and findToFile, that are not associated with a specific package
- xmlApiTypes.xsd—describes the data types, which range from simple types like Date, Char, and IpAddress, to complex structures like Any, List, and Filter

  Note:

  The Filterholder, ChildrenFilterHolder, and ResultFilter types are also defined in the “general methods/types” section of the XML API Reference. See 8.13 “To view the general methods and types” (p. 104) for information about viewing the general methods and types.

The XML schema files for NFM-P subsystems such as script management and the JMS are also available.

The following table describes the core methods that a client can use to perform specific actions. The methods are defined in xmlApiMethods.xsd.

Table 10-1 Core XML methods

Method	Purpose	Input parameters	Output parameters
deregister-LogToFile	To stop the creation of accounting statistics files	• **fullClassName (optional)** Package qualified class name in dot-separated format. If this parameter is not specified, the method deregisters all accounting statistics registrations (string) • **jmsClientId** The JMS Client ID (string)	—
deregister-SasLogToFile	To stop the creation of OAM test result files	• **fullClassName (optional)** A comma-separated list of package qualified class names in dot-separated format to deregister. the method deregisters all test results registrations. If a class is not an instance of sas.TestResult or sas.TraceHop, an exception occurs, no classes are deregistered, and the request processing terminates unless continueOnFailure is true. (string) • **jmsClientId** The JMS Client ID (string) • **continueOnFailure (optional)** Continue processing requests in the stream if an exception occurs, unless the request is invalid. The default is false. (Boolean)	—
deregisterNo-tification	To deregister an XML filter and stop advanced JMS message filtering for the client	• **jmsClientId** The JMS Client ID (string)	—
find	To return the set of objects of a specified type that match the specified filter criteria	• **fullClassName** Package qualified class name in dot-separated format • **filter (strongly recommended)** Filter which objects are returned based on the properties of the class specified by fullClassName • **timeout (optional)** Specifies the time, in milliseconds, after which the find operation times out. The request is aborted and an exception is returned. • **resultFilter (strongly recommended)** Filter which attributes are returned for each object	• **result** (generic.CommonManagedEntityInfor-mation) The list of objects which match the given filter criteria.

Table 10-1 Core XML methods (continued)

Method	Purpose	Input parameters	Output parameters
findToFile	To find the set of objects of the specified type that match the filter criteria, and store the result in the specified file. Upon completion of the request, a fileAvailable event is generated.	See Table 14-3, “findToFile method input parameters” (p. 191) .	—
ping	To check the connectivity of the NFM-P server.	—	—
registerLogTo-File	To create accounting statistics files based on the specified class type. Accounting statistics files are created and are placed in the specified directory on the server. Each time a file is created, a LogFileAvail-ableEvent event is generated. If there is no JMS client subscribed within a given time, the NFM-P server deregisters the request.	See Table 14-2, “registerLogToFile method input parameters” (p. 185) .	—
register-SasLogToFile	To create OAM test results file based on specified class types. Files are created and are placed in the specified directory on the server. Every time that a file is created, a LogFileAvail-ableEvent event is generated. If there is no JMS client subscribed within some time, the NFM-P server deregisters the request.	See Table 12-1, “registerSasLogToFile method input parameters” (p. 156) .	—
registerNotifi-cation	To register an advanced JMS message filter and begin using the filter for messages sent to the OSS client	• **jmsClientId** The JMS Client ID (string) • List of filter criteria, in XML format; see Chapter 4, “Event monitoring using JMS” for information about advanced JMS message filtering	—
sleep	To halt request processing for the given period of time (in milliseconds) before proceeding to the next request element.	• **milliseconds** The length of time in milliseconds to wait before executing the next request element (long). The maximum is 60 000 milliseconds.	—

Table 10-1 Core XML methods (continued)

Method	Purpose	Input parameters	Output parameters
timeStamp	To view the current Java time on the NFM-P server, in milliseconds. The execution time of a command or group of commands can be determined by surrounding them with timeStamp commands and taking the delta of their return values.	• **includeFormatted (optional)** Include formatted timestamp in the response (Boolean)	• **millis** The difference, in milliseconds, between the current time and midnight, Jan 1, 1970 UTC (long) • **timeStamp** The time stamp (string)
version	To return NFM-P software release information	—	• **version** The official software release identifier (string) • **baseVersion** The software release ID in *major.minor* format (string) • **build** The software build identifier (string) • **patch** The software patch level (string)

10.3 XML method schema files

10.3.1 Overview

XML methods are defined in PackageNameMethods.xsd files. All method definitions begin with a header containing:

- method name
- function—one of the following:
  - MODIFIER
  - RETRIEVER
- scope—one of the following:
  - Instance—requires instanceFullName to be specified
  - Class—requires className to be specified
- input parameters
- output parameters
- exceptions

The following figure is an example of a method definition header for the clearFaults method of the FaultManager class in the fm package.

Figure 10-1 Sample fm.FaultManager.clearFaultsOnObject method definition header

```
<!--
     = fm.FaultManager.clearFaultsOnObject
     = [MODIFIER]
     = [SCOPE: CLASS]
     =
     = This method clears the faults raised against the base instance that matches
the faultFilter criteria.
     =
     =
     = [IN]
     =   deployer - Deployer
     =       The deployment state.
     =
     =   synchronousDeploy - xsd:boolean
     =       (optional) Specify whether to block until the changes have been
     =       fully deployed to network. A value of "true" means to block.
     =       A value of "false" means to return immediately.
     =       Default: false (asynchronous)
     =
     =   clearOnDeployFailure - xsd:boolean
     =       (optional) Specify whether clear the deployer if a failure occurs.
     =       A value of "true" means to clear.
     =       A value of "false" means to leave the failed deployer.
     =       Default: false
     =
     =   deployRetries - xsd:int
     =       (optional) The number of times to attempt re-deployment during
     =       synchronous deployment. This parameter is meaningless in the
     =       asynchronous case.
     =       Default: 0
     =
     =   deployRetryInterval - xsd:long
     =       (optional) The number of milliseconds to wait between deployment
     =       retries. This parameter is meaningless in the asynchronous case.
     =       Default: 0
     =
     =   taskDescription - xsd:string
     =       (optional) A user friendly description of what the operation does.
     =       This information will be used by the task manager.
     =
     =   baseInstanceFullName - xsd:string
     =       Full name of the affected object on which the alarms should be
cleared.
     =
     =   faultFilter - FilterHolder
     =       Filter applied to alarms.
     =
```

```
     =   continueOnFailure - xsd:boolean
     =      (Optional) Continue processing requests in this stream if an exception
occurs, unless the request is invalid
     =
     =
     = [EXCEPTIONS]
     =   fm.FaultManager.clearFaultsOnObjectException
     -->
```

10.3.2 Common elements of methods

The following table describes elements that are common to all NFM-P methods.

Table 10-2 Common elements of methods

Element	Description	Options
deployer	Specifies the deployment state.	—
synchronousDeploy	Specifies whether synchronous or asynchronous deployment is used.	• false (default) Asynchronous deployment • true Synchronous deployment
clearOnDeployFailure	Specifies whether to clear the deployment if a failure occurs.	• false (default) The deployment is not cleared • true (recommended) Clear the deployment
deployRetries	Specifies the number of times that deployment is retried during a synchronous deployment. This is an optional parameter.	0 (default and recommended)
deployRetryInterval	Specifies the time, in milliseconds, between deployment attempts in synchronous deployments. This parameter is optional.	0 (default)
instanceFullName	Specifies the full name of the object.	—
configInfo	Specifies any class type.	

10.4 XML type schema files

10.4.1 Overview

XML object classes are defined in *PackageNameTypes*.xsd files. *PackageName* is the name of the package. All object class definitions begin with a header containing:

- Name
- Type - class
- Abstract - yes/no
- Singleton - yes/not

- Category - one of service, topology, general
- Deployable - yes/no
- Read access - list of users or all
- Write access - list of users or all
- Inheritance Hierarchy - inheritance tree

The following figure is an example of an Object Class Type definition header for the AdditionalTextAttribute class in the fm package.

Figure 10-2 Object Class definition header example

```
<!--
      =
      =  Name:          fm.AdditionalTextAttribute
      =  Type:          Class
      =  Abstract:      no
      =  Singleton:     no
      =
      =  Category:      fault
      =  Deployable:    no
      =  Read Access:   all
      =  Write Access:  admin,fault
      =
      =  Inheritance Hierarchy:
      =
      =   <root>
      =      |
      =   fm.AdditionalTextAttribute
      =
      -->
```

10.5 Standard properties of classes

10.5.1 Overview

All object classes in the schema definition files contain the following standard property elements:

- actionMask—configuration operation to perform on an object
- children-Set—set of contained children of an object
- objectFullName—NFM-P identifier of an object instance
- selfAlarmed—whether an object raises alarms

The following figure shows the standard properties definition of the fm.AdditionalTextAttribute class.

Figure 10-3 Standard properties definition for a class

```
<xsd:complexType name="fm.AdditionalTextAttribute">\
    <xsd:all>
            <!-- standard properties -->
            <xsd:element name="objectFullName" minOccurs="0">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:maxLength value="252"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:element>
            <xsd:element name="actionMask" type="generic.ModifierActionMask"
minOccurs="0"/>
            <xsd:element name="children-Set" type="Children-Set" minOccurs="0"/>
            <xsd:element name="selfAlarmed" type="xsd:boolean" minOccurs="0"/>
```

10.5.2 actionMask element

Each class definition contains an actionMask element that specifies what operation is to be performed on the object. The actionMask element is used only for object configuration. You can use the genericTypes.xsd file to specify the actionMask element definition (generic.ModifierActionMask).

The following table lists and describes the actionMask element values.

Table 10-3 actionMask element values

Action type	Numeric ID	Description
create	1	Creates an object based on the specified parameters. Only read-write and read-create attributes are allowed. The read-create attribute is mandatory.
modify	2	Modifies an existing object based on the specified parameters. Only read-write attributes are allowed.
delete	4	Deletes an object.
reset	8	Resets the parameters of an existing object to the default values.

Note: Not all actionMask values apply to all classes.

Nokia recommends that you use one action type for each request. However, you can combine action types. For example, when you want to create and modify an object that does not exist, you can specify the create and modify actions in the same request. The following figure shows an example of a combined request

Figure 10-4 Combined create and modify request example

```
<actionMask>
   <bit>create</bit>
   <bit>modify</bit>
</actionMask>
```

10.5.3 children-Set element

The objects in the XML API model are organized in a hierarchy. The parent-child relationships of the hierarchy are defined by the children-Set element of an object.

The following figure is an example that shows how the children-Set element is used in an XML request to set the base card in slot 7 of an NE to In Service, and to provision the card as a card_iom2_20g.

Figure 10-5 children-Set example

```
<equipment.CardSlot>
   <actionMask>
      <bit>modify</bit>
   </actionMask>
   <children-Set>
      <equipment.BaseCard>
         <actionMask>
      <administrativeState>inService</administrativeState>
      <slotId>7</slotId>
         <specificType>
            <bit>card_iom2_20g</bit>
         </specificType>
      </equipment.BaseCard>
   </children-Set>
</equipment.CardSlot>
```

10.5.4 objectFullName element

The objectFullName element specifies the fully distinguished name, or FDN, that uniquely identifies an object. The objectFullName format depends on the object type. Table 10-4, “Object FDN examples” (p. 135) lists the FDN formats of various object types.

Note: There is no documented list of object full name formats. The samples in Table 10-4, “Object FDN examples” (p. 135) are not guaranteed to remain valid between NFM-P releases. Nokia recommends that XML APIrequest writers find FDNs to use in their requests by performing a find request for the class of the object they are looking for. See 13.3 “Inventory retrieval methods” (p. 162) for information about the find method.

For a class that has no associated objects, a find request does not return a result from which you can obtain an FDN. In such a scenario, you must create the object using an NFM-P GUI client, and then perform the find request.

You can also monitor the NFM-P logs for the FDN of an object that you create or modify using the GUI. See 9.1.6 “Logging XML requests, responses, and exceptions” (p. 120) in 9.1 “XML message structure ” (p. 105) for information about logging.

Table 10-4 Object FDN examples

Network object (class)	Format	Example
Network element		

Table 10-4 Object FDN examples (continued)

Network object (class)	Format	Example
Network element	Created for each managed NE and forms the base of site-specific objects. network:*siteID*	network:10.1.1.88
Policies		
Policy manager	A policy manager exists for each type of policy at the network and element level and contains sets of policies of a type. For example, access and slope. *policy Manager Type* for network level or network:*siteID*:*policyManagerType* for element level	Access Egress or network:10.1.1.88:Access Egress
Access egress policy (aengr.Policy)	policy Manager Type:*ID* or *network:siteID*:*policyManagerType*:*policyID* for element level	Access Egress:2 or network:10.1.1.88:Access Egress:2
Access egress queue (aengr.Queue)	*network:siteID*:*policyManagerType*:*policyID*:queue-*ID*	network:10.1.1.88:Access Egress:2:queue-1
Access ingress policy (aingr.Policy)	*policyManagerType*:*ID* or *network:siteID*:*policyManagerType*:*policyID* for element level	Access Ingress:2 or network:10.1.1.88:Access Ingress:2
Access ingress forwarding class (aingr.ForwardingClass)	*policyManagerType*:*forwarding-class-type* or *network:siteID*:*policyManagerType*:*forwarding-class-type* for element level	Access Engress:2:forwarding-class-be or network:10.1.202.94:Access Engress:2:forwarding-class-be
Access ingress queue (aingr.Queue)	network:*siteID*:*policyManagerType*:*policyID*:queue-*ID*	network:10.1.1.88:Access Ingress:2:queue-1
Network policy (niegr.Policy)	Network:*ID* or *network:siteID*:Network:*ID* for element level	Network:2 network:10.1.1.88:Network:2
Network forwarding class (niegr.ForwardingClass)	Network*ID*:*forwarding-class-type* or *network:siteID*:Network*ID*:*forwarding-class-type* for element level	network:10.1.1.88:Network:3:forwarding-class-af
File policy (file.Policy)	File:*policyID* or *network:siteID*:File:*ID* for element level	File:2 or network:10.1.1.88:File:2
Slope policy (slope.Policy)	*slopeName* or *network:siteID*:Slope:default	network:10.1.1.88:Slope:default

Table 10-4 Object FDN examples (continued)

Network object (class)	Format	Example
IP filter policy (aclFilter.IPFilter)	IP Filter:*policyID* or *network:siteID*:IP Fitler:*policyID* for element level	IP Filter:5 or network:10.1.1.88:IP Filter:5
Mediation security policy (security.MediationPolicy)	pollerManager:mediation-security-*policyID*	pollerManager:mediation-security-1
Deployment policy (mediation.DeploymentPolicy)	Default.DeployerBank:Policy	Default.DeployerBank:Policy
Deployment (generic.DeployerInfo)	Default.DeployerBank:depl-*ID*	Default.DeployerBank:depl-322
Router		
Router (rtr.VirtualRouter)	There is only one VR for each NE. *network:siteID*:router-1 for element level	network:10.1.1.88:router-1
Network Interface (rtr.NetworkInterface)	*network:siteID*:router-1:ip-interface-*ID* for element level	network:10.1.1.88:router-1:ip-interface-2
Static route (rtr.StaticRoute)	*network:siteID*:router-1:SR-*ID-DestinationsiteID- IP mask* for element level	network:10.1.1.88:router-1:SR-2-12.119.60.0-255.255.255.0
OSPF		
OSPF (ospf.area)	ospf:area-*siteID*	ospf:area-2.2.2.2
OSPF site (ospf.Site)	*network:siteID*:router-1:ospf for element level	network:10.11.1.219:router-1:ospf
OSPF area (ospf.AreaSite)	*network:siteID*:router-1:ospf:areaSite-*siteID* for element level	network:10.11.1.219:router-1:ospf:areaSite-0.0.0.0
OSPF interface (ospf.interface)	*network:siteID*:router-1:ospf:areaSite-*siteID*:interface-*siteID* for element level	network:10.11.1.219:router-1:ospf:areaSite-0.0.0.0:interface-3.3.3.3.0
OSPF export policy (ospf.ExportPolicy)	*network:siteID*:router-1:ospf:exportPolicy for element level	network:10.11.1.219:router-1:ospf:exportPolicy
BGP		
BGP (bpg.Site)	*network:siteID*:router-1:bgp for element level	network:10.11.1.219:router-1:bgp
BGP peer group (bpg.PeerGroup)	*network:siteID*:router-1:bgp:group-*Group name* for element level	network:10.11.1.219:router-1:bgp:group-Sample-Client
BGP group import policy (bgp.GroupImportPolicy)	*network:siteID*:router-1:bgp:group-*Group name*:importPolicy for element level	network:10.11.1.219:router-1:bgp:group-Sample-Client:importPolicy
BGP group export policy (bgp.GroupExportPolicy)	*network:siteID*:router-1:bgp:group-*Group name*:exportPolicy for element level	network:10.11.1.219:router-1:bgp:group-Sample-Client:exportPolicy
BGP site export policy (bgp.SiteExportPolicy)	*network:siteID*:router-1:bgp:exportPolicy for element level	network:10.11.1.219:router-1:bgp:exportPolicy
BGP site import policy (bgp.SiteExportPolicy)	*network:siteID*:router-1:bgp:importPolicy for element level	network:10.11.1.219:router-1:bgp:importPolicy

Table 10-4 Object FDN examples (continued)

Network object (class)	Format	Example
BGP confederation (bgp.Confederation)	*network:siteID*:router-1:confederation-*ID* for element level	network:10.11.1.219:router-1:confederation-100
BGP confederation member (bgp.ConfederationMember)	*network:siteID*:router-1:confederation-*ID*:member-*ID* for element level	network:110.11.1.219:router-1:confederation-100:member-10
LDP		
LDP (ldp.Site)	*network:siteID*:router-1:ldp for element level	network:10.11.1.219:router-1:ldp
MPLS		
MPLS (mpls.Site)	*network:siteID*:router-1:mpls for element level	network:10.1.1.219:router-1:mpls
MPLS interface (mpls.interface)	*network:siteID*:router-1:mpls:interface-*ID* for element level	network:10.1.1.219:router-1:mpls:interface-1
MPLS dynamic LSP (mpls.DynamicLSP)	lsp:from-*Source siteID*-id-*ID*	lsp:from-10.1.1.88-id-1
MPLS LSP path (mpsl.LspPath)	lsp:from-*Source siteID*-id-*ID*:lsppath-*ID*	lsp:from-10.1.1.88-id-1:lsppath-2
MPLS provisioned path (mpls.ProvisionedPath)	provisionedMplsTePath:from-*Source siteID*-id-*ID*	provisionedMplsTePath:from-10.1.1.88-id-1
MPLS provisioned hop (mpls.ProvisionedHop)	provisionedMplsTePath:from-*Source siteID*-id-*ID*:hop-*ID*	provisionedMplsTePath:from-10.1.1.88-id-1:hop-2
MPLS tunnel (mpls.Tunnel)	mplsTunnel:from-*Source siteID*-id-*ID*	mplsTunnel:from-10.1.1.88-id-1
Subscribers		
Subscriber (subscr.Subscriber)	subscriber:*ID*	subscriber:21
Services		
Service (service.Service)	serviceManager:service:*ID*	svc-mgr:service-12
Service site (service.Site)	serviceManager:service:*ID*:*siteID*	svc-mgr:service-5:10.1.1.91
Service access interface (service.AccessInterface)	serviceManager:service*ID*:*site ID*:ip-interface-*ID*	svc-mgr:service-5:10.1.1.91:interface-1/1/4-inner-tag-0-outer-tag-11 or svc-mgr:service-52:10.1.202.93:ip-interface-2051
Channels		
STS192 Channel (sonetequipment.Sts192Channel)	network:*siteID*:shelf-*ID*:cardSlot-*ID*:card:daughterCardSlot-*ID*:daughterCard:port-*ID*:*Sts192Channel*	network:10.1.1.218:shelf-1:cardSlot-2:card:daughterCardSlot-1:daughterCard:port-1:sts192-1
DS3/E3 Channel (tdmequipment.DS3E3Channel)	network:*siteID*:shelf-*ID*:cardSlot-*ID*:card:daughterCardSlot-*ID*:daughterCard:port-*ID*:*DS3E3Channel-ID*	network:10.1.1.218:shelf-1:cardSlot-3:card:daughterCardSlot-1:daughterCard:port-1:ds3e3-1
DS0 Channel Group (tdmequipment.DS0ChannelGroup)	network:*siteID*:shelf-*ID*:cardSlot-*ID*:card:daughterCardSlot-*ID*:daughterCard:port-*ID*:*DS1E1Channel-ID*:*DS0ChannelGroup-ID*	network:10.1.1.218:shelf-1:cardSlot-3:card:daughterCardSlot-1:daughterCard:port-1:ds1e1-3:ds0Grp-3

10.5.5 Non-standard parameter elements

A class definition may include one or more parameter elements in addition to the standard elements. Each element definition has a header that contains some or all of the following:

- Name—parameter name
- Defined in—parent class
- Min/Max—minimum and maximum values
- Default—default value
- Access—allowed access type

The following figure shows the EncapValue parameter definition in the antispoof.AntiSpoofingFilter class.

Figure 10-6 Parameter element definition example

```
<!--
  -   Name:       EncapValue
  -  Defined in: antispoof.AntiSpoofingFilter
  -  Default: 0
  -  Access: Read-Write   -->
<xsd:element name="encapVal" minOccurs="0">
   <xsd:simpleType>
      <xsd:restriction base="xsd:long">
         <xsd:minInclusive value="0"/>
         <xsd:maxInclusive value="4294967295"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
```

10.5.6 Accessibility types

Each parameter element has one of the following accessibility types:

- read-only—the element is read-only and cannot be used in a configuration request
- read-write—the element can be modified using a create or modify request
- read-create—the element can be modified using only a create request

11 Fault management

11.1 Fault management

11.1.1 Overview

The OSS applications that focus on fault management use events and alarms to detect, log, notify users of, and automatically correct network problems to keep the network running effectively.

11.2 Communicating with the NFM-P

11.2.1 Overview

An OSS fault management application can use a JMS and/or an XML/SOAP interface to communicate with the NFM-P server. For alarm management:

- The NFM-P sends network event/alarm notifications using the JMS interface in near-real-time to the OSS. An OSS uses the JMS event stream to monitor new alarms and status changes on existing alarms such as clearing or escalation. See Chapter 4, “Event monitoring using JMS” for information.
- The OSS uses XML/SOAP to send requests for retrieving information about objects with alarms, historical alarm information, and configuring alarms. See Chapter 5, “XML requests” for information.

The following sections provide information about JMS messages and API calls that are relevant to a fault management OSS developer.

11.3 Alarm sources

11.3.1 Overview

The sources that initiate NFM-P alarm notifications are:

- network traps
- NFM-P alarms

11.3.2 Network traps

The NFM-P monitors network health by listening for traps from the managed network elements. These SNMP traps may indicate conditions that include configuration or operational state changes, security breach attempts, and equipment faults.

When the NFM-P detects a fault in the network, one or more alarms are generated, depending on the cause of the fault. When an alarm is created, modified or deleted the JMS messages that contain the alarm information are sent to interested listeners.

Note: Although the NFM-P receives network information in traps and generating alarms about fault conditions in the network, the traps are not directly mapped to alarms.

11.3.3 NFM-P alarms

The NFM-P generates alarms based on network conditions; the alarms do not apply to specific NEs. The alarms provide information including: faults and changes to the NFM-P server system, database, redundancy, users, and logs.

11.4 Alarm definitions

11.4.1 Overview

An alarm in the NFM-P is represented by an instance of fm.AlarmObject. The attributes of the AlarmObject are described in the XML API Reference, and in 11.5 “Alarm messages” (p. 142) .

Information about the alarm types are also available. See the NSP NFM-P Alarm Search Tool for more information.

When an OSS receives an alarm from the NFM-P, the OSS can map to the corresponding alarmDetails.csv using the fields described in the following table.

Table 11-1 Alarm mapping details

fm.AlarmObject attribute	Corresponding alarmDetails field	Comments
alarmClassTag	alarmClassTag	alarmClassTag is unique in the alarmDetails list
alarmName	alarmName	The combination of alarmName and alarmType is unique in the alarmDetails list
type	alarmType	

11.5 Alarm messages

11.5.1 Overview

An OSS needs to subscribe to JMS in order to receive the notification of new alarms or changes to existing alarms; for example, escalation, de-escalation, acknowledgement, or recurrence. See “Alarm-specific header properties” (p. 39) in 4.6 “JMS message filtering” (p. 37) for information about properties that are common to all JMS events. See Table 4-3, “Event header properties, alarm events” (p. 40) for properties that are specific to fault messages.

When an alarm is raised, the NFM-P sends a JMS message with an fm.AlarmInfo structure. The following figure shows a message example for a new alarm. The fields of the AlarmInfo correspond to the properties of the new fm.AlarmObject. See the XML API Reference for fm.AlarmInfo and fm.AlarmObject information.

Figure 11-1 New alarm message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>ObjectCreation</eventName>
         <ALA_category>FAULT</ALA_category>
         <ALA_isCorr>false</ALA_isCorr>
```

```
         <ALA_isVessel>false</ALA_isVessel>
         <ALA_alarmType>equipmentAlarm</ALA_alarmType>
         <ALA_allomorphic>fm.AlarmObject</ALA_allomorphic>
         <MTOSI_NTType>NT_ALARM</MTOSI_NTType>
         <MTOSI_serviceAffecting>false</MTOSI_serviceAffecting>
         <MTOSI_probableCause>inoperableEquipment</MTOSI_probableCause>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <MTOSI_aliasNameList>EquipmentDown</MTOSI_aliasNameList>
         <ALA_OLC>2</ALA_OLC>
         <MTOSI_osTime>1315841370394</MTOSI_osTime>
         <MTOSI_objectName>faultManager:network@198.51.100.47@shelf-1@cardSlot-
1@card@daughterCardSlot-1@daughterCard@port-32|alarm-10-3-8</MTOSI_objectName>
         <MTOSI_perceivedSeverity>major</MTOSI_perceivedSeverity>
         <ALA_span>:2:</ALA_span>
         <MTOSI_objectType>fm.AlarmObject</MTOSI_objectType>
         <ALA_eventName>ObjectCreation</ALA_eventName>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <objectCreationEvent>
            <fm.AlarmInfo>
               <severity>major</severity>
               <previousSeverity>indeterminate</previousSeverity>
               <originalSeverity>major</originalSeverity>
               <highestSeverity>major</highestSeverity>
               <probableCause>8</probableCause>
               <alarmName>10</alarmName>
               <type>3</type>
               <affectedObjectFullName>network:198.51.100.47:shelf-1:cardSlot-1:card:
daughterCardSlot-1:daughterCard:port-32</affectedObjectFullName>
               <affectedObjectClassName>equipment.PhysicalPort</affectedObjectClassName>

               <isAcknowledged>false</isAcknowledged>
               <wasAcknowledged>false</wasAcknowledged>
               <acknowldegedBy>N/A</acknowldegedBy>
               <clearedBy>N/A</clearedBy>
               <deletedBy>N/A</deletedBy>
               <firstTimeDetected>1315841369120</firstTimeDetected>
               <lastTimeDetected>1315841369120</lastTimeDetected>
               <lastTimeSeverityChanged>0</lastTimeSeverityChanged>
               <lastTimeCleared>0</lastTimeCleared>
               <lastTimePromoted>0</lastTimePromoted>
               <lastTimeDemoted>0</lastTimeDemoted>
               <lastTimeEscalated>0</lastTimeEscalated>
               <lastTimeDeEscalated>0</lastTimeDeEscalated>
               <lastTimeAcknowledged>0</lastTimeAcknowledged>
               <frequency>0</frequency>
               <occurences>N/A</occurences>
```

```
               <numberOfOccurences>1</numberOfOccurences>
               <numberOfOccurencesSinceClear>1</numberOfOccurencesSinceClear>
               <numberOfOccurencesSinceAck>0</numberOfOccurencesSinceAck>
               <isServiceAffecting>false</isServiceAffecting>
               <additionalText>N/A</additionalText>
               <operatorAssignedUrgency>indeterminate</operatorAssignedUrgency>
               <urgencyAssignedBy>N/A</urgencyAssignedBy>
               <relatedObjects>
                  <null/>
               </relatedObjects>
               <affectingObjects>
                  <null/>
               </affectingObjects>
               <subscriberId>0</subscriberId>
               <nodeId>198.51.100.47</nodeId>
               <nodeName>sim20_47</nodeName>
               <affectedObjectDisplayedName>Port 1/1/32</affectedObjectDisplayedName>

               <applicationDomain>Physical Equipment</applicationDomain>
               <displayedClass>PhysicalPort</displayedClass>
               <alarmClassTag>equipment.EquipmentDown</alarmClassTag>
               <affectedObjectClassIndex>87</affectedObjectClassIndex>
               <affectedObjectInstanceIndex>2018</affectedObjectInstanceIndex>
               <deployerName/>
               <deployerId>0</deployerId>
               <requestId>N/A</requestId>
               <requestUser>N/A</requestUser>
               <correlatingAlarm>N/A</correlatingAlarm>
               <isImplicitlyCleared>true</isImplicitlyCleared>
               <numberOfCorrelatedAlarms>0</numberOfCorrelatedAlarms>
               <olcState>inService</olcState>
               <objectFullName>faultManager:network@198.51.100.47@shelf-
1@cardSlot-1@card@daughterCardSlot-1@daughterCard@port-32|alarm-10-3-8</
objectFullName>
               <objectClassName>fm.AlarmObject</objectClassName>
               <allomorphicClassName>fm.AlarmObject</allomorphicClassName>
               <objectId>606777272</objectId>
               <displayedName>Port 1/1/32 - network@198.51.100.47@shelf-
1@cardSlot-1@card@daughterCardSlot-1@daughterCard@port-32|alarm-10-3-8</
displayedName>
               <lifeCycleState>1</lifeCycleState>
               <deploymentState>0</deploymentState>
               <neId>198.51.100.47</neId>
               <spanObjectPointer>node-198.51.100.47</spanObjectPointer>
               <parentClassName>fm.FaultManager</parentClassName>
               <unsetProperties/>
            </fm.AlarmInfo>
         </objectCreationEvent>
      </jms>
```

```
   </SOAP:Body>
</SOAP:Envelope>
```

The following figure shows an attribute value change for a recurring alarm. Each time that the alarm recurs, the number of occurrences (both absolute and since the last clear) increment and the time last detected is updated. The following figure shows the old and new values for each of the fields. See also “AttributeValueChangeEvent” (p. 339) in Appendix B, “JMS events”.

Figure 11-2 Recurring alarm message example, attribute value change

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>AttributeValueChange</eventName>
         <ALA_category>FAULT</ALA_category>
         <ALA_OLC>2</ALA_OLC>
         <ALA_isVessel>false</ALA_isVessel>
         <ALA_allomorphic>fm.AlarmObject</ALA_allomorphic>
         <MTOSI_osTime>1316464116127</MTOSI_osTime>
         <MTOSI_NTType>NT_ALARM</MTOSI_NTType>
         <MTOSI_objectName>faultManager:network@3.3.3.3@router-1@ospf-v2@areaSite-
0.0.0.0@interface-18|alarm-40-11-43-packetSourceIpAddress=10.1.3.6</MTOSI_
objectName>
         <ALA_span>:2:</ALA_span>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <MTOSI_objectType>fm.AlarmObject</MTOSI_objectType>
         <ALA_eventName>AttributeValueChange</ALA_eventName>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <attributeValueChangeEvent>
            <objectFullName>faultManager:network@3.3.3.3@router-1@ospf-v2@areaSite-
0.0.0.0@interface-18|alarm-40-11-43-packetSourceIpAddress=10.1.3.6</
objectFullName>
            <attribute>
               <attributeName>numberOfOccurencesSinceClear</attributeName>
               <newValue>
                  <long>3490</long>
               </newValue>
               <oldValue>
                  <long>3489</long>
               </oldValue>
            </attribute>
            <attribute>
               <attributeName>lastTimeDetected</attributeName>
               <newValue>
                  <long>1316464116127</long>
               </newValue>
```

```
            <oldValue>
               <long>1316463991997</long>
            </oldValue>
         </attribute>
         <attribute>
            <attributeName>numberOfOccurences</attributeName>
            <newValue>
               <long>3490</long>
            </newValue>
            <oldValue>
               <long>3489</long>
            </oldValue>
         </attribute>
      </attributeValueChangeEvent>
   </jms>
</SOAP:Body>
</SOAP:Envelope>
```

11.6 Retrieving alarms

11.6.1 Overview

While ongoing monitoring of alarms is done using JMS, an OSS may need to retrieve alarms at system startup or after losing synchronization with the NFM-P. See 4.8 “Connection monitoring and error recovery” (p. 57) for information about situations in which an OSS may lose synchronization with the NFM-P.

The fm.FaultManager provides several convenience methods for retrieving faults. It is recommended the methods be used instead of find or findToFile for fm.AlarmObjects. See the fm.FaultManager in the XML API Reference for all of the methods available. The most common method for retrieving alarm information is fm.FaultManager.findFaults.

See the XML API SDK Sample Code Navigator Fault directory for example alarm retrieval methods.

11.6.2 Alarm filtering

The fm.FaultManager.findFaults method provides options to filter alarms and display a list of alarms. The faultFilter tag is an input parameter of the findFaults method for the alarm filtering. In the following figure, the fault filter specifies that alarms returned meet the condition where the alarms have a node Id of 198.51.100.71 and the last time the alarms were detected was after May 19th 2011, 12:03:12 GMT.

Figure 11-3 findFaults method example using faultFilter element

```
<faultFilter>
   <and class="fm.AlarmInfo">;
      <equal name="nodeId" value="198.51.100.71"/>
      <greater name="lastTimeDetected" value="1305806592556"/>
   </and>
</faultFilter>
```

faultFilter method usage requires knowledge of the fm.AlarmInfo object attributes. See fm.FaultManager in the XML API Reference for information.

11.6.3 Alarm result filtering

The resultFilter type applies to alarms that the findFault method returns. See Chapter 6, “XML API filtering” for more information about the resultFilter type.

The following figure shows a findFault method filter that returns only the alarm attributes listed in the resultFilter element.

Figure 11-4 findFaults method example using resultFilter element

```
<resultFilter>
   <attribute>severity</attribute>
   <attribute>highestSeverity</attribute>
   <attribute>probableCause</attribute>
   <attribute>alarmName</attribute>
   <attribute>affectedObjectFullName</attribute>
   <attribute>affectedObjectClassName</attribute>
   <attribute>isAcknowledged</attribute>
   <attribute>firstTimeDetected</attribute>
   <attribute>lastTimeDetected</attribute>
   <attribute>numberOfOccurences</attribute>
   <attribute>isServiceAffecting</attribute>
   <attribute>additionalText</attribute>
   <attribute>nodeId</attribute>
   <attribute>nodeName</attribute>
   <attribute>displayedName</attribute>
</resultFilter>
```

11.7 Alarm management

11.7.1 Overview

Most common operations on alarms, including promotion/demotion, acknowledgment, and clearing, can be performed using convenience methods that are provided by fm.FaultManager. These methods take either a single alarm or set of alarms as arguments. See the XML API Reference for more information. For code samples, see the XML API SDK Sample Code Navigator.

11.8 Alarm policies

11.8.1 Overview

Alarm policies are used to control how the NFM-P handles incoming alarms. For example, an OSS can manipulate the global policy (fm.GlobalPolicy), which applies to all of the alarms, or specific policies that apply to each type of alarm (fm.SpecificPolicy). There is one global policy for the entire system, and one specific policy for each alarm based on the alarmClassTag. Both types of policies are configured using generic.GenericObject.configureInstanceWithResult. See Figure 9-6, “XML

request example" (p. 37) and the XML API Reference for more information about the methods. See the XML API SDK Sample Code Navigator for a sample request to configure a specific alarm policy.

11.9 Alarm correlation

11.9.1 Overview

A correlated alarm indicates that a fault on an object is caused by a fault on another object. For example, if a port goes down, an alarm is generated for the port, and correlated alarms are generated for services that rely on the port. To ignore correlated alarms in the JMS stream, filter out alarms where ALA_isCorr is true.

Note: The ALA_isCorr property is not included in all events. See "Alarm-specific header properties" (p. 39) in 4.6 "JMS message filtering" (p. 37) for information.

See "Correlated alarms" in the *NSP NFM-P Classic Management User Guide* for more information about correlated alarms.

11.10 OSS client alarm testing

11.10.1 Overview

The following functions can be used for testing fault management OSS applications: test alarms, and the JMS record and playback tool.

11.10.2 Test alarms

You can use the fm.FaultManager.testAlarm method to initiate an alarm that is generated by the NFM-P and sent to the OSS client using JMS. The fm.FaultManager.testAlarm method allows you to generate alarms using the NFM-P and facilitates alarm testing using an OSS client. The fm.FaultManager.testAlarm method allows you to manually generate alarms instead of using equipment traps to initiate the alarm condition.

Note: It is recommended the fm.FaultManager.testAlarm method not be used in the production environment. The alarms created using this method are not self-clearing, so must be manually cleared.

The fm.FaultManager.testAlarm method does not perform a validation to ensure that you are generating an alarm with the correct parameters for the correct object class. The generated alarm does not have any correlation, aggregation, or association relationships with other objects or alarms.

To create an alarm that simulates a real alarm, you must include the following alarm attributes when you use the fm.FaultManager.testAlarm method:

- objectInstanceName
- alarmNameId
- alarmTypeId
- probableCauseId
- severity

- alarmClassTag

The fm.FaultManager.testAlarm method can be used in a development environment to test specific types of alarms that require real NEs, where real NEs are not accessible. It is an effective means to generate alarms and verify the alarm response output that is received by an OSS client. The following figure shows an example of how to generate a service site down alarm.

Figure 11-5 Alarm generation request example

```
<SOAP:Body>
   <fm.FaultManager.testAlarm xmlns="xmlapi_1.0">
      <deployer>immediate</deployer>
      <objectInstanceName>svc-mgr:service-3:10.1.1.90</objectInstanceName>
      <alarmNameId>97</alarmNameId>
      <alarmTypeId>17</alarmTypeId>
      <probableCauseId>84</probableCauseId>
      <severity>critical</severity>
      <alarmClassTag>svc.ServiceSiteDown</alarmClassTag>
      <namingComponent/>
      <additionalText>my additional text</additionalText>
   </fm.FaultManager.testAlarm>
</SOAP:Body>
```

The following figure shows an example of the response to the request to generate a service site down alarm.

Figure 11-6 Alarm generation response example

```
<SOAP:Body>
   <fm.FaultManager.testAlarmResponse xmlns="xmlapi_1.0"/>
</SOAP:Body>
```

11.10.3 JMS event recording and replay

You can use an NFM-P utility to record JMS event messages and replay the messages in a test environment. See 4.10 “NFM-P JMS client configuration and testing” (p. 61) for more information.

11.11 Workflow to set up alarm management

11.11.1 Stages

The following workflow lists the high-level steps required to set up alarm management.

1

Establish a JMS subscription. See 4.10 “NFM-P JMS client configuration and testing” (p. 61) for more information about creating a JMS consumer.

2

Retrieve inventory information for the required elements in the network. See A.2 “Recommended durable JMS client operation” (p. 327) for how to perform a retrieval without losing events during a retrieval of inventory.

3

If required, retrieve objects that affect or are related to the required network objects using fm.FaultManager.findObjectsAffectionOfn and fm.FaultManager.findObjectsRelatedToOfn methods. See the XML API Reference for more information about the methods.

4

Retrieve alarms for the network elements of interest using fm.FaultManager.findAlarmsForOfn. See the XML API Reference for more information.

5

As required, configure alarm policies. See 11.8 “Alarm policies” (p. 147) for more information.

6

As required, acknowledge, clear, or promote/demote the alarms. See 11.7 “Alarm management” (p. 147) for more information.

12 OAM

12.1 OAM

12.1.1 Overview

The NFM-P allows you to create, configure, execute, schedule, and collect results for OAM tests, for network troubleshooting, and to verify compliance with SLAs. This chapter covers topics of interest to an OAM OSS developer.

12.2 References

12.2.1 Overview

The following references are of interest to an OSS developer working in the OAM domain:

- *NSP NFM-P Classic Management User Guide* - See “Service Test Manager”, “OAM diagnostic tests” , and “Ethernet CFM” chapters, the CFM 802.lag Policies section in the Services chapter, and the Ethernet Connectivity Fault Management section in the Troubleshooting chapter.
- ITU-T Recommendation Y.1731
- IEEE 802.lag Standard
- XML API SDK Sample Code Navigator - documents samples of OSS objects

12.3 Key packages and classes

12.3.1 Overview

The sas package defines common interfaces and operations for service assurance, OAM tests, and test policies and suites. The package also defines abstract test classes such as sas.Ping and sas.Trace, which inherit from sas.Test. Specific tests inherit from the abstract classes, and are defined in the package of the objects that they test. For example, LSP Ping and LSP Trace are subclasses of sas.Ping and sas.Trace, and are defined in the mpls package. See the main page of the XML API Reference for a list of package descriptions.

12.4 Tests

12.4.1 Overview

The categories of service assurance tests are:

- ping tests (subclasses of sas.Ping)
- trace tests (subclasses of sas.Trace)
- actions (subclasses of sas.Action)

The types of each test are:

- test (subclass of sas.Test)

- deployed test (subclass of sas.DeployedTest)
- test definition (subclass of sas.TestDefinition)

Generally, deployed tests are transient objects that are created when the sas.Test is run. The deployed test is removed after the test is completed. The results are retrieved from the NE and associated with the sas.Test.

Note: The NFM-P supports OmniSwitch ping and traceroute tests via user-defined CLI scripts instead of sas package objects. For sample scripts, see the sample OmniSwitch ping and traceroute CLI scripts in the *NSP NFM-P Classic Management User Guide*. For information about deploying CLI scripts, see 9.2 “CLI command methods” (p. 122) .

To create a test using the OSS interface, use generic.GenericObject.configureChildInstance. The CreateServiceTunnelPing.xml file in the code samples demonstrates the creation of a svt.TunnelPing test with default values. See the XML API Reference for the test class and additional test properties. See the XML API SDK Sample Code Navigator for sample XML code.

To run an existing test, use the executeAndWait method of sas.Test. See the XML API Reference for method parameters. Several sample usages are available in the XML API code samples; for example, LspPing-UserExisting.xml. See the XML API SDK Sample Code Navigator for sample XML code.

The sas.Test object provides a convenience method to create and run a test using an XML request. See sas.Test.adhocExecuteAndWait in the XML API Reference for more information. Several sample usages are also in the XML code samples; for example, LspPing-CreateAndUse.xml. See the XML API SDK Sample Code Navigator for sample XML code.

See “OAM diagnostic tests” in the *NSP NFM-P Classic Management User Guide* for a list of the OAM diagnostic tests that are supported by the NFM-P. Classes for most tests are in the packages for the tested elements; for example, ATM ping is defined in the atm package. The only exception to this is Ethernet CFM tests, which can be found in the ethernetoam package. See 7.2 “Packages” (p. 86) for information about mapping entity types to packages.

12.5 Test suites

12.5.1 Overview

Test suites are used to group OAM tests. After the suites are created, they can be scheduled to provide periodic results or run on demand to investigate service issues. Test in a suite are organized in the following groups:

- first-run tests
- generated tests
- last-run tests

First-run tests typically include high-level diagnostics, such as service-side or VPRN ping. The tests can be run sequentially or in parallel. Generated tests are run after the first-run tests. See 12.6 “Generated tests” (p. 153) for more information about generated tests. Last-run tests typically include transport-layer diagnostics, such as an LSP trace or a tunnel ping, and can also be run sequentially or in parallel.

Test suites are usually associated with a specific type of tested entity, such as VLL service or VPLS. You can also create a test suite with a mix of tested entities by setting testedEntityType to none (1).

To create a test suite, use the generic.GenericObject.configureChildInstanceWithResult method to create a child of the sas object. For a simple example, see CreateSimpleTestSuite.xml. See the XML API SDK Sample Code Navigator for sample XML code.

To add tests to a test suite, use sas.TestSuite.addTest or addTests. To run a test suite, use the sas.AbstractTest.execute method. For an example, see ExecuteTestsFromTestSuite.xml. See the XML API SDK Sample Code Navigator for sample XML code.

See “Service Test Manager” in the *NSP NFM-P Classic Management User Guide* for more information about test suites. See the XML API Reference for more information about the properties and methods of sas.TestSuite.

12.6 Generated tests

12.6.1 Overview

The NFM-P can be used to generate tests using test definitions, policies, and suites. A test definition (subclass of sas.TestDefinition), describes the parameters of a test without actually attaching the test to a testing entity. A test policy defines a group of test definitions that apply to a specific type of tested entity; for example, LSPs. By adding a test policy and a list of tested entities to a test suite, tests can be generated for each entity, based on the definitions in the test policy.

For example, by creating a test suite with a group of LSPs and a test policy with an LSP ping test definition, a ping test can be generated for each of the LSPs in the suite.

12.7 Workflow to generate tests

12.7.1 Stages

The following workflow lists the high-level steps required to generate tests. See “Service Test Manager” in the *NSP NFM-P Classic Management User Guide* for more information about test suites, test policies, and test generation. See the XML API SDK Sample Code Navigator for sample XML code.

1

Create a test policy with test definitions. The test policy needs to specify the type of target object to be tested, and the test definitions need to be valid for the object. See the CreateLSPTestPolicy.xml file in the XML code samples.

2

Create a test suite, specifying the test policy. See the CreateSimpleTestSuite.xml file in the code samples. Alternatively, you can add your test policy to a test suite. See the XML API SDK Sample Code Navigator for sample XML code.

3

Add target objects to the test suite. See the AddTestedEntityToTestSuite.xml file in the code samples.

4

As required, combine Stage 2 and Stage 3 to create a test suite and add tested entities to the suite in a single xml request. See the CreateTestSuite.xml file in the code samples.

5

Generate the tests:

a. Use sas.TestSuite.generateTests. See the GenerateTestsFromTestSuite.xml file in the code samples.

b. Use the generateTests flag in sas.TestSuiteAddTestedEntities. See the XML API Reference for more information and for other methods on sas.TestSuite.

12.8 Scheduling tests

12.8.1 Overview

Tests can be scheduled in the NFM-P or on the NE.

12.8.2 NFM-P scheduled tests

Tests are scheduled in the NFM-P by associating a test suite with a schedule object.

12.9 To schedule a test suite using the NFM-P

12.9.1 Steps

1

Create a test suite. See 12.5 “Test suites” (p. 152) for information. The test suite can contain manually created tests, generated tests, or both.

2

Create a schedule. See Misc/createSchedule_hourlyOngoing.xml in the code samples. See the XML API SDK Sample Code Navigator for sample XML code.

3

Create a scheduled task that associates the test suite with the schedule. See ScheduleTestSuite.xml in the code samples.

End of steps

12.9.2 Reference

See “NFM-P-based schedules” in the *NSP NFM-P Classic Management User Guide* for information about scheduling. See schedule.Sam.Schedule in the XML API Reference for scheduling options.

12.10 NE schedule tests

12.10.1 Overview

The sas.NeSchedulableTest class represents a schedulable test on the NE. An NE schedulable test that is bound to a sas.Test results in the test being immediately installed onto the NE by the creation of a sas.DepolyedTest. Such tests may then be scheduled using the NE cron function.

12.11 Retrieving results

12.11.1 Overview

Test result objects are subclasses of sas.TestResult (abstract). The concrete classes are defined in the package for the tested object. For example, the results of a VCCV ping are a svt.VccvPingResult.

The following describes the ways for an OSS to access test results:

- Run a test and get the rest result as a response.
- Retrieve results of past tests (for example, for a particular time period).
- Register to get notification of scheduled test results as they become available. This is the preferred method. See 12.15 “Performance and scalability” (p. 160) for more information.

12.11.2 Run a test and get a response

An OSS can run an existing test (using sas.Test.executeAndWait) or create and run a test (using sas.Test.adhocExecuteAndWait). Both of these methods return the test result in their response. See the XML API Reference for detailed descriptions of these methods. See CreateAndUse.xml in the code samples. See the XML API SDK Sample Code Navigator for sample XML code.

12.11.3 Retrieve results of past tests

An OSS can use findToFile to retrieve results of tests that have already run. A common implementation would be to retrieve tests of a particular type for a specific time interval. See 13.3.3 “findToFile method” (p. 164) in Chapter 13, “Inventory management” for more information. An OSS which employs this option must keep in mind the lifetime of test result storage in the database. See 12.15 “Performance and scalability” (p. 160) for more information.

12.11.4 Register for notification of test results

XML API clients can use the registerSasLogToFile method to retrieve OAM test result export files for specific test class results. A LogFileAvailableEvent is sent each time a file is available for retrieval; a client must subscribe to JMS to receive the events. By default, if the client is not subscribed, no result files are created, and the NFM-P deregisters the client after a period of client inactivity. A JMS subscription for registerSasLogToFile is optional, however; you can disable the

JMS client inactivity check to ensure that no deregistration occurs. See A.6 “To configure the registerLogToFile or registerSasLogToFile client inactivity check” (p. 334) for information.

OAM data is exported to a result file after the data is read from the NE. The file is saved to a specified directory on the NFM-P server.

Note: Not all OAM tests provide OAM data when the registerSasLogToFile method is used. The following conditions are required for OAM data to be generated for registerSasLogToFile:

- OAM test sas.Test.neSchedulable attribute is true
- OAM test sas.Test.accountingPolicyObjectPointer attribute is not empty
- OAM test sas.Test.accountingFiles attribute is true
- OAM test sas.Test.testResultStorage attribute value is logToFile or logToDBAndFile

See “OAM diagnostic tests” in the *NSP NFM-P Classic Management User Guide* for configuration information.

The NFM-P server verifies the presence of the JMS client subscription that is specified in the registerSasLogtoFile method using the jmsClientId input parameter. If the client is not subscribed, the server deregisters the registerSasLogToFile request. This means that OAM result export files are not generated after the Log Retention Time expires.

It is recommended that you run the registerSasLogToFile method every time the client establishes a JMS session.

The following table describes the input parameters of the registerSasLogToFile method.

Table 12-1 registerSasLogToFile method input parameters

Input parameter	Description	Values
fullClassName	A comma-separated list of package qualified class names to register. Superclasses may be specified. If a class in the list is not an instance of sas.TestResult or sas.TraceHop, an exception occurs, none of the classes are registered, and request processing terminates unless continueOnFailure is true. (String)	Comma-separated list of OAM results
dirName	The relative path of the directory in which to save the files; the path is relative to the oam directory below the OSS XML export directory. The parameter is mandatory. A best practice is to use a separate directory for each application that exports different test results or uses different filters.	*String*
compress	Specifies whether export files are to be compressed when they are created. A value of True saves files with gzip compression and a filename extension of gz. The default is False. (Optional) Specifying a compression option increases the CPU processing load.	False (default) True
jmsClientId	The JMS client ID or Kafka Subscription ID that is notified when a new export file is created based on the specified parameters; a notification is sent to each client that is registered for OAM logs when an export file is created. If the JMS client inactivity check is disabled, an empty string must be specified.	See Chapter 4, “Event monitoring using JMS” for information about the JMS client ID format.

Table 12-1 registerSasLogToFile method input parameters (continued)

Input parameter	Description	Values
filter	Filter on the text attribute(s) of the log record that will be written to the stats file, or accountingPolicyId integer attribute (applies only to accounting statistics). (Optional) The following filter tags are supported: • and • equal • in • not • notEqual • or • wildcard	<filter>*element*</filter>
resultFilter	The attribute set to include in the OAM result export file, where the attributes are part of the specific test result object; you can also use the parameter to reduce the size of the result file by reducing the number of attributes included per test result object. (Optional)	<attribute>*attribute_name*</attribute>
continueOnFailure	Optional parameter that specifies whether to continue processing requests in the stream if an exception occurs, except in the case of a request parsing error; the default is false. (Boolean) (Optional)	False (default) True

The OAM result files are created using unique names in a directory relative to /opt/nsp/nfmp/server/xml_output/oam.

OAM result file retrieval using the registerSasLogToFile method requires a RHEL user account on each main server. Such a user account requires FTP or SFTP access, depending on whether TLS is enabled on the XML API. See 14.6.2 “Configuring FTP and SFTP access for OSS clients” (p. 181) for the specific requirements and best practices for creating OSS client user accounts.

See the XML API SDK Sample Code Navigator, OAM directory, for registerSasLogToFile request, jms logFileAvailableEvent, and OAM result export file samples.

The following figure shows an OAM log file request using the registerSasLogToFile method.

Figure 12-1 OAM log file request example

```
<registerSasLogToFile xmlns="xmlapi_1.0">
   <fullClassName>sas.TestResult, sas.TraceHop</fullClassName>
   <dirName>jmsclientdir</dirName>
   <jmsClientId>JMS_client@n</jmsClientId>
   <resultFilter>
      <attribute>testSuiteId</attribute>
      <attribute>testId</attribute>
      <attribute>fromNodeId</attribute>
      <attribute>neTestRunIndex</attribute>
      <attribute>resultStatus</attribute>
   </resultFilter>
</registerSasLogToFile>
```

The configuration of testResultStorage is different for OAM PM Diagnostics test results. The OAM PM diagnostic tests results are modeled from sas.PmStats. You can retrieve the results of the following classes of OAM PM diagnostic test sessions from LogToDB, LogTofFile or both.

- ethernetoam.CfmDmmSession
- ethernetoam.CfmLmmSession
- ethernetoam.CfmSlmSession
- sas. MplsDmSession
- sas. TWLSession

To store the test results in the database, you can specify the writeAccountingResultsToDb attribute per session test or globally as System Preferences. See the Assurance section of the User Guide for more details.

registerSasLogToFile filtering

To reduce the volume of exported statistics data, a client can apply a filter during registerSasLogToFile registration. The filter criteria determine which records are returned to a client; for example, if multiple API clients are individually responsible for statistics collection in specific subnet; each client can apply a filter to return only the statistics associated with the NEs in specific subnets.

You can filter on any properties of a statistics log record that are string-based, such as monitoredObjectClass or monitoredObjectSiteId. See the LogRecord class definition in the XML API Reference for a list of the class properties.

For accounting statistics, you can configure an NFM-P system preference to include the accounting policy ID in each accounting statistics record. This enables an OSS client to use registerSasLogToFile and JMS filters based on the accounting policy ID.

See the system preferences configuration procedures in the *NSP System Administrator Guide* for more information.

See the registerSasLogToFile method description in the XML API Reference for a list of the supported filter elements and syntax.

See 4.6 “JMS message filtering” (p. 37) for information about creating JMS filters.

12.12 Ethernet OAM

12.12.1 Overview

Ethernet OAM tests are modeled in the ethernetoam package. See the XML API Reference for the classes used to represent OAM tests and results. See the following areas in the *NSP NFM-P Classic Management User Guide* for more information about the NFM-P support for Ethernet OAM:

- OAM chapter
- Ethernet CFM chapter
- CFM 802.lag Policies section in the Services chapter
- Ethernet Connectivity Fault Management section the Troubleshooting chapter

See the OAM directory in the code samples for Ethernet OAM configuration examples. See the XML API SDK Sample Code Navigator for sample XML code.

12.13 PM Session OAM

12.13.1 Overview

PM Session CFM testing is based on Metro Ethernet Forum Specification 35 - Service OAM Performance Monitoring Implementation Agreement. Testing is performed in Layer 2 networks. The reporting of results occurs at standardized measurement intervals (15 minutes, 1 hour, and 1 day), and comprises a statistical summary of the results of individual test frames.

The PM Session testing framework can also be utilized in the IP domain to perform TWAMP IP level monitoring, and handles both IPv4 and IPv6 addresses. The TWAMP Light test targets Layer 3 interfaces. It provides an option to monitor IP SLA performance as related to KPI.

Figure 12-2, “PM session statistics request example” (p. 158) shows a request example for PM Session statistics using the registerSasLogToFile method. The <fullClassName> entries are the XML classes for the OAM PM accounting statistics that are available. They include:

- DMM session accounting statistics
- DMM bin accounting statistics
- SLM session accounting statistics
- LMM session accounting statistics
- TWAMP Light session accounting statistics
- TWAMP Light bin accounting statistics

Figure 12-2 PM session statistics request example

```
<registerSasLogToFile>
   <fullClassName>saspm.CfmDmmSessionAccStats,
      saspm.CfmDmmBinAccStats,
      saspm.CfmSlmSessionAccStats,
      saspm.CfmLmmSessionAccStats,
      saspm.TWLSessionAccStats,
      saspm.TWLBinAccStats
   </fullClassName>
      <dirName>oampm</dirName>
      <jmsClientId>JMS_client@n</jmsClientId>
</registerSasLogToFile>
```

See the following in the *NSP NFM-P Classic Management User Guide* for more information about the supported PM Session tests:

- PM Session tests chapter
- Service Test Manager chapter

See the *NSP NFM-P Statistics Management Guide* for information about viewing statistics data and graphing statistics using the NFM-P Statistics Plotter. Both historical and real-time plots are supported for OAM PM.

12.14 OAM for OmniSwitch

12.14.1 Overview

OmniSwitch tests are modeled in the aossas package. See the XML API Reference for classes used to represent OmniSwitch tests and results. See “OAM diagnostic tests” in the *NSP NFM-P Classic Management User Guide* for information about OmniSwitch diagnostic tests.

12.15 Performance and scalability

12.15.1 Overview

Size constraint policies control the number of historical records retained by the database. These are configured by the network administrator. If an OSS is using find or findToFile to retrieve test results periodically from the database, the OSS developer needs to work with the network operator to ensure that results are kept for long enough to be retrieved. Also, to reduce the NFM-P processing load, and for faster retrieval of test results, it is recommended that an OSS register for notifications of scheduled test results using the registerSasLogToFile method.

13 Inventory management

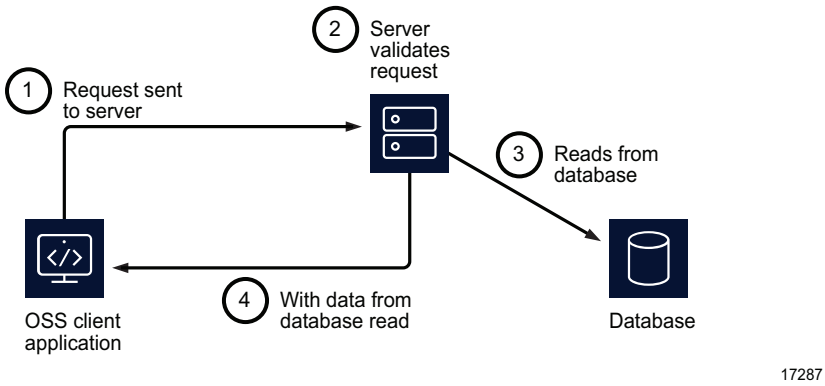
13.1 Inventory management

13.1.1 Overview

The XML API interface provides methods that allow OSS applications to request information about the managed network. The inventory information can be maintained by keeping it synchronized with the NFM-P in real time, or by periodically sending a request for updated information.

The following figure shows the message flow of an OSS request for information from the NFM-P database.

Figure 13-1 OSS request message flow



13.2 Network object model

13.2.1 Overview

For OSS inventory management applications, OSS developers need to identify the NEs and objects that are to be managed, and understand the structure of the inventory model and the relationships between contained objects. For example, the database of a typical inventory management system includes physical device elements such as cards, ports, and links, and the associations with services and service objects.

The NFM-P models the following object types in the network:

- physical elements, such as NEs, shelves, cards, ports, and physical links
- logical elements, such as protocols, policies, services, and OAM tests

13.2.2 NFM-P object hierarchy

The objectFullName element in an inventory request shows the hierarchical position of an object in the NFM-P network model. The following is an objectFullName example for a port:

```
network:<siteID>:shelf-<ID>:cardSlot-<ID>:card:daughterCardSlot-<ID>:daughterCard:port-<ID>
```

The objectFullName example shows the following object containment:

```
network
    →site
        →shelf
            →cardSlot
                →card
                    →daughterCardSlot
                        →daughterCard
                            →port
```

The XML API object model is organized in a hierarchy that has parent-child relationships for all management objects. For detailed information about the object model, see the XML API Reference, which is described in Chapter 8, “XML API Reference” , and the NFM-P schema files, which are described in Chapter 10, “Schema Reference” .

13.3 Inventory retrieval methods

13.3.1 Overview

The XML API provides the following methods to retrieve information:

- find—retrieves the set of objects that match the request criteria; the result is returned in XML format; see 13.3.2 “find method” (p. 164) in this section for more information
- findToFile—retrieves the set of objects that match the request criteria and stores the result in a file on a local or remote host; see 13.3.3 “findToFile method” (p. 164) in this section for more information
- findFaults—retrieves alarms that match the request criteria; see Chapter 11, “Fault management” for more information.

Note: Other methods in the equipment and network packages can be used to retrieve specific objects and classes. Nokia does not recommend the use of these methods because they are not optimized.

The following table lists and describes the XML methods and parameters that an OSS client can use to retrieve objects.

Table 13-1 XML object retrieval methods

Method and parameters	Parameter description
find	
fullClassName	The package-qualified class name in dot-separated format
filter (strongly recommended)	A filter that is based on the class properties
timeout (optional)	The time, in milliseconds, after which the NFM-P aborts the request and returns an exception
resultFilter (strongly recommended)	A filter that reduces the amount of object information that is retrieved
continueOnFailure (optional)	Whether to continue processing requests in the stream if an exception other than a request parsing error occurs
findToFile	
fullClassName here	The package-qualified class name in dot-separated format
fileName	The relative file in which to store the results. The format is: ftp://*user:password@host:port/directory/filename* or, if secure FTP is used: sftp://*user:password@host:port/directory/filename* The following rules apply: • When using a remote client, the directory must be relative to the default FTP directory on the remote client. • The characters '/' and ';' are reserved and must be encoded. For example, to specify a file in /tmp directory on the remote client, the path must be encoded as ftp://name@password/%2Ftmp/filename.xml. The path ftp://name@password/tmp/filename.xml specifies a file in a /tmp directory under the default FTP directory of the user. See *RFC 1738 Uniform Resource Locators (URL), Section 3.2.2* for more information about the FTP URL requirements.
timeStamp (optional)	Whether to include the timestamp in the name of the result file
compress (optional)	Whether to compress the result file using gzip compression; the default is False
timeout (optional)	The time, in milliseconds, after which the NFM-P aborts the request and returns an exception
filter (strongly recommended)	A filter that is based on the class properties
resultFilter (strongly recommended)	A filter that reduces the amount of object information that is retrieved
synchronous (optional)	Whether the method runs synchronously, in which case the result is returned only when the operation is complete; the default is true
continueOnFailure (optional)	Whether to continue processing requests in the stream if an exception other than a request parsing error occurs
findFaults	

Table 13-1 XML object retrieval methods (continued)

Method and parameters	Parameter description
faultFilter	A filter that is based on the alarm properties
resultFilter (optional)	A filter that reduces the amount of object information that is retrieved
continueOnFailure (optional)	Whether to continue processing requests in the stream if an exception other than a request parsing error occurs

13.3.2 find method

The find method returns a set of objects of the type specified in the <fullClassName> element that match the filter criteria. The method response is a streamed XML result.

The following figure shows a request to find an ACL IP filter.

Figure 13-2 find request example

```
<SOAP:Body>
   <find xmlns="xmlapi_1.0">
      <fullClassName>aclfilter.IpFilter</fullClassName>
      <filter>
         <equal name="objectFullName" value="IP Filter:1000"/>
      </filter>
   </find>
</SOAP:Body>
```

13.3.3 findToFile method

The findToFile method returns a set of objects of the type specified in the fullClassName element that match the filter criteria. You can specify the following result file storage options:

- save the file locally on the NFM-P server
- use FTP to transfer the results to a remote host

By default, the findToFile method runs synchronously, which means that one request has to complete and return before the processing of the next request begins.

The NFM-P returns the following when a findToFile request cannot be processed:

- asynchronous request:
  - HTTP error message, if a request timeout occurs
  - JMS notification, for an error during the query execution or file creation
- synchronous request:
  - HTTP error message, if the number of requests exceeds the maximum allowed or a request timeout occurs

The following figure shows a request to retrieve historical statistics from an NE.

Figure 13-3 findToFile request example

```
<SOAP:Body>
   <findToFile xmlns="xmlapi_1.0">
      <fullClassName>equipment.InterfaceAdditionalStatsLogRecord</fullClassName>
      <filter>
         <and>
            <equal name="monitoredObjectPointer" value="network:10.1.202.93:shelf-
1:cardSlot-1:card:daughterCardSlot-1:daughterCard:port-3"/>
            <between name="timeCaptured" first="1127142900000" second="1127143800000"/>

         </and>
      </filter>
      <fileName>Equipment.InterfaceAdditionalStatsLogRecord.xml</fileName>
   </findToFile>
</SOAP:Body>
```

For synchronous findToFile requests, the NFM-P sends the following findToFile response shown in the following figure if the request contains no errors.

Figure 13-4 findToFileResponse message example

```
<findToFileResponse xmlns="xmlapi_1.0">
```

If a findToFile request contains an error, the response message contains an exception that indicates the type of error. See Chapter 9, “XML message structure” for information about the XML message structure and exception messages.

When all request processing is complete and the results are stored in a file, the NFM-P sends a fileAvailableEvent JMS notification. The notification includes the result and the request ID, as shown in the following figure.

Figure 13-5 fileAvailableEvent JMS notification example

```
<SOAP:Body>
   <jms xmlns="xmlapi_1.0">
      <fileAvailableEvent>
         <requestID>JMS_client@n</requestID>
         <fileName>asynchronous_request.xml</fileName>
      </fileAvailableEvent>
   </jms>
</SOAP:Body>
```

If the request does not complete successfully, the notification contains an exception. See “FileAvailableEvent example” (p. 345) in Appendix B, “JMS events” for more information.

Scheduled findToFile export

The findToFile method supports the export of data using the NFM-P scheduler framework. The following table describes how to schedule a findToFile data export.

Table 13-2 Scheduled findToFile export

Step	Request example
1. Create a schedule, or choose an existing schedule.	<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0"> <deployer>immediate</deployer> <distinguishedName>scheduleManager</distinguishedName> <childConfigInfo> <schedule.SamSchedule> <actionMask> <bit>create</bit> </actionMask> <id>10</id> <displayedName>samSchedule-10</displayedName> <frequency>minute</frequency> <onGoing>true</onGoing> <runEveryMinute>5</runEveryMinute> </schedule.SamSchedule> </childConfigInfo> </generic.GenericObject.configureChildInstance>
2. Create a findToFileTask, or refer to an existing task.	<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0"> <deployer>immediate</deployer> <distinguishedName>scheduleManager</distinguishedName> <childConfigInfo> <schedule.OssTask> <actionMask> <bit>create</bit> </actionMask> <id>10</id> <displayedName>FindToFile</displayedName> <commandName>FindToFile</commandName> <command> <findToFile xmlns="xmlapi_1.0"> <fullClassName>netw.NetworkElement</fullClassName> <fileName>netw.xml</fileName> </findToFile> </command> </schedule.OssTask> </childConfigInfo> </generic.GenericObject.configureChildInstance>

Table 13-2 Scheduled findToFile export (continued)

Step	Request example
3. Create a samScheduledTask that points to a schedule, and a findToFile task.	<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0"> <deployer>immediate</deployer> <distinguishedName>scheduleManager</distinguishedName> <childConfigInfo> <schedule.SamScheduledTask> <actionMask> <bit>create</bit> </actionMask> <id>15</id> <administrativeState>2</administrativeState> <displayedName>FindToFileTask-15</displayedName> <description>FindToFileTask15</description> <scheduledObjectPointer>scheduleManager:OssTask-111</scheduled ObjectPointer> <schedulePointer>scheduleManager:samSchedule-10</ schedulePointer> </schedule.SamScheduledTask> </childConfigInfo> </generic.GenericObject.configureChildInstance>
4. Turn up, shut down, start, or stop the samScheduledTask.	—

Result file format

A findToFile result file has the same XML format as the streamed result of a find request. The result filters specified in a request determine the specific objects that a request returns. See 13.6 “Request filters” (p. 170) for information about result filters.

Local result file storage

The findToFile output location is an NFM-P main server directory below /opt/nsp/nfmp/server/xml_output. By default, a main server keeps findToFile output files for a retention period of 15 minutes. Every five minutes, the main server deletes each output file that is older than the retention period. Contact technical support if you require different findToFile retention settings.

The following figure shows an example of local findToFile output storage; the output file name is specified between the fileName tags.

Figure 13-6 findToFile example, local storage

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope
  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <security>
            <user>username</user>
            <password>password</password>
         </security>
```

```
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <findToFile xmlns="xmlapi_1.0">
         <fullClassName>equipment.PhysicalPort</fullClassName>
         <filter>
            <and>
               <equal name="siteId" value="10.1.202.93"/>
               <equal name="shelfId" value="1"/>
               <equal name="cardSlotId" value="1"/>
               <equal name="daughterCardSlotId" value="1"/>
               <equal name="mode" value="access"/>
            </and>
         </filter>
         <fileName>equipmentPhysicalPort.xml</fileName>
      </findToFile>
   </SOAP:Body>
</SOAP:Envelope>
```

Remote result file storage

You can configure the findToFile method to record the query results to a file on a remote station using FTP. The fileName parameter of the findToFile method allows you to configure the optional FTP credentials, host identifier, and file path. If the FTP credentials are not specified, the XML API uses the following:

- username—anonymous
- password—an e-mail address

The following figure is a remote storage configuration example for the findToFile method.

Figure 13-7 findToFile example, remote storage

```
<findToFile xmlns="xmlapi_1.0">
   <synchronous>false</synchronous>
   <fullClassName>class_name</fullClassName>
   <fileName>
    [s]ftp://user:password@host:port/directory/filename
   </fileName>
   <resultFilter>
.
.
.
   </resultFilter>
</findToFile>
```

13.4 “To configure remote findToFile result storage” (p. 169) describes how to configure the permissions for the findToFile FTP directory and FTP account on a remote server.

13.4 To configure remote findToFile result storage

13.4.1 Steps

1

For each XML API client application, create a directory on the remote station below the directory associated with the findToFile method. In the following example, the findToFile method uses a directory called XML API, and there are three client applications:

```
.../XMLAPI/clientapp1
.../XMLAPI/clientapp2
.../XMLAPI/clientapp3
```

2

Enable read and write permissions for each directory created in Step 1.

3

Create an FTP account for use by the XML API clients. The FTP account home directory must be the directory associated with the findToFile method, for example, XMLAPI.

4

Specify the appropriate client directory in the <fileName> section of the request for each client, as shown in the following example, which uses a directory named in Step 1:

```
<fileName>
   [s]ftp://user:password@host:port/clientapp1/output_file
</fileName>
```

END OF STEPS

13.5 Inventory request processing

13.5.1 Overview

The NFM-P processes find requests synchronously. A findToFile request runs synchronously by default, but you can use the <synchronous> Boolean element to specify whether a findToFile request runs synchronously or asynchronously.

13.5.2 Synchronous requests

When the NFM-P receives multiple synchronous find or findToFile requests, up to five are processed concurrently. Subsequent synchronous requests are rejected with the following exception, which is sent in the HTTP response:

```
javax.xml.stream.XMLStreamException: The Inventory connections reached its maximum please retry later
```

13.5.3 Asynchronous findToFile requests

When the NFM-P receives multiple asynchronous findToFile requests, up to five are processed concurrently, and up to an additional ten requests are queued to be processed. If the NFM-P is currently servicing 15 asynchronous requests, further incoming requests are immediately rejected with the following exception:

Figure 13-8 Too many requests exception

```
<findToFileException xmlns="xmlapi_1.0">
   <description>Cannot perform findToFile command, too many queued requests</
description>
</findToFileException>
```

You can configure a timeout value for the OSS requests associated with an NFM-P user or user group. When a request times out, the NFM-P sends an exception in a fileAvailableEvent JMS notification. The notification includes a status element that contains FAILURE, as shown in the following figure:

Figure 13-9 OSS request timeout notification

```
<SOAP:Body>
   <jms xmlns="xmlapi_1.0">
      <fileAvailableEvent>
         <requestID>XML_API_client@n</requestID>
         <fileName>asynchronous_request.xml</fileName>
         <status>FAILURE</status>
         <errorMessage>javax.xml.stream.XMLStreamException: The request timed out
for waiting for a worker. XML API request id [:0:1] User Name [ossuser_10] File
Name [asynchronous_request.xml]. Please retry later.</errorMessage>
      </fileAvailableEvent>
   </jms>
</SOAP:Body>
```

13.5.4 Mix of find and findToFile requests

When the NFM-P receives a combination of synchronous find, and synchronous or asynchronous findToFile requests, up to five requests are processed concurrently; subsequent synchronous or asynchronous requests are rejected with the following exception, which is sent in the HTTP response:

```
javax.xml.stream.XMLStreamException: The Inventory connections reached its maximum please retry later
```

13.6 Request filters

13.6.1 Overview

You can define a filter for any object property to limit or refine the following:

- information returned to the OSS application after a request is made to the server

- configuration requests sent to the server

See 6.1 “XML API filter types” (p. 75) for more information about request filtering.

13.6.2 Result filters

A result filter applies only to the returned objects. See 6.2 “Result filtering” (p. 77) for more information about result filters.

14 Accounting, performance, and flow monitoring

14.1 Accounting, performance, and flow monitoring

14.1.1 Overview

The NFM-P provides a scalable platform for reliably collecting statistics from the managed NEs and the NFM-P system. The statistics are typically used for monitoring and troubleshooting an NFM-P network, and for SLA and billing functions.

Statistics collection must be enabled on the NFM-P before an OSS can retrieve them. Statistics collection can be performed on-demand or scheduled. See “Statistics collection in the NFM-P” in the *NSP NFM-P Statistics Management Guide* for more information.

The NFM-P can collect the following statistics:

- performance statistics—collected by polling NE MIBs, transferred to the NFM-P using SNMP, and stored in the NFM-P database
- accounting statistics—collected in files on NEs, transferred to the NFM-P using FTP or SCP, and stored in the NFM-P database

  Note:

  The NFM-P does not store AA accounting statistics in the NFM-P database. Instead, the NFM-P saves the statistics data in files for reporting in NSP Analytics reports, or for IPDR record processing by an OSS application. See “AA accounting statistics collection” in the *NSP NFM-P Classic Management User Guide* for more information.
- server performance statistics—collected from NFM-P system functions and processes, stored in the NFM-P database
- AA Cflowd statistics—collected from NEs by NSP Flow Collectors, can be stored in Auxiliary database or exported in IPDR format
- Cflowd statistics—collected from NEs by external collector; the NFM-P does not collect cflowd statistics or store in the NFM-P database

To collect statistics, the NFM-P requires policies that specify the following:

- the network or service objects from which to collect statistics
- the statistics counters to collect
- the collection rate
- how long the NFM-P is to retain the collected statistics data

14.2 References

14.2.1 Documentation

The following guides include information that is relevant to an OSS developer who works in the statistics domain:

- *NSP NFM-P Statistics Management Guide*—contains information that is useful in the design phase, and implementation information such as NFM-P configuration for statistics collection
- *NSP NFM-P Classic Management User Guide*— documents AA and the collection of AA statistics

14.2.2 Developer tools

The following developer reference tools support NFM-P statistics collection. See the Network Developer Portal for more information about developer tools.

- IPDR Reference—lists and describes the fields in an IPDR-formatted flow record for each collection or aggregation domain based on user-specified criteria; also lists the IPDR schema changes between NFM-P releases
- NSP Flow Collector Fields Dependencies—describes the relationships and dependencies between ISA-AA Cflowd IPFIX fields and the fields in the associated NSP Flow Collector aggregation records
- Statistics Search Tool—provides detailed, device-specific information about the accounting and performance statistics counters that the NFM-P can collect. The Statistics Search Tool is downloadable from the Documentation Center.
- XML API Reference—documents the specific classes, attributes, and methods for configuring and collecting statistics using the XML API; see 14.3 “Key packages and classes” (p. 173) for the packages and classes that are of interest to an OSS developer. The XML API Reference is downloadable from the Documentation Center.
- XML API SDK Sample Code Navigator—provides code samples

14.3 Key packages and classes

14.3.1 Collection

accounting package

The accounting package contains methods that you can use to define accounting policies for the collection of accounting statistics. The package contains methods for the collection of customizable accounting records.

file package

The file package contains methods that you can use to define file policies. File policies define the storage location, retention period, and rollover period for accounting statistics files on managed devices.

snmp package

The snmp package contains methods that you can use to define statistics policies. Statistics policies define the MIBs to enable for collection, and the polling interval for performance statistics on managed devices. You can also define specific objects for which to collect statistics.

14.3.2 Storage

log package

The log package contains the CurrentData and LogRecord abstract classes, that encapsulate collected statistics. The concrete classes that inherit from these classes are in the same packages as the target objects for which statistics are collected. For example, access egress octets that are collected on a service are in a service.AccessEgressOctetsLogRecord, which inherits from log.LogRecord. The package includes the class LogToFileManager, that stores registerLogToFile related attributes.

statistics package

The statistics package contains methods that you can use to manage the NFM-P performance monitoring statistics. Performance data is collected for alarms, memory usage, SNMP traps, statistics, and NE polling.

14.3.3 Retrieval

root package

The root package contains the findToFile and registerLogToFile/deregisterLogToFile methods that are used to retrieve statistics data. See 14.8 “Statistics retrieval using registerLogToFile” (p. 183) and 14.9 “Statistics retrieval using findToFile” (p. 189) for more information. See general methods and types information in the XML API Reference for information about these methods.

14.4 Statistics objects

14.4.1 Overview

The XML API uses the following objects to track statistics:

- current data record—for the most recent performance statistics
- log records—for historical accounting and performance statistics

Statistics collection on an object results in creation of many log records but only one current data record per monitored object.

Note: The current data records do not apply to accounting statistics.

Log records are maintained for scheduled and non-scheduled (on-demand) statistics. The XML API organizes statistics objects by package. For example, the bgp package defines the BGP statistics. The BGP peer statistics are tracked by the PeerStats class, which contains the current data record, and the PeerStatsLogRecord class, which contains the log record in the bgp package.

You must use the corresponding log record object to retrieve statistics for an object. The log records contain the following information:

- time captured
- monitored object identifier
- relevant statistics

Scheduled and on-demand performance statistics use current data records. History is available for the scheduled polls using the LogRecord type. The data from the last scheduled poll is also stored using the LogRecord type. For example, if bgp.PeerStats are collected in a 5-minute polling interval, the scheduled polling creates a current data record and a log record. Historical statistics are not available for on-demand statistics.

The <objectFullName> of the scheduled current data starts with: logger:scheduled. The on-demand current data <objectFullName> starts with: logger:real-time. A historical log record is also created for the scheduled current data. See 10.5.4 “objectFullName element” (p. 135) in 10.1 “Schema Reference ” (p. 127) for more information about the unique identifier of the instance of the object.

A policy is associated with each type of log record. You can use the log policy to modify the retention period and alarm thresholds that are associated with a log record. For example, you can modify the PeerStatsLogPolicy class to customize the log records that are collected for the BGP Peer Stats. The default settings generally provide appropriate values for the retention period and thresholds. It also contains methods to remove specified log records from the NFM-P database. See the XML API Reference for information about this method.

14.5 Statistics collection methods: Scheduled and on-demand statistics

14.5.1 Overview

You can collect statistics data using the following methods:

- scheduled polls
- non-scheduled (on-demand) collection

To schedule collection of statistics, certain policies are required to be configured. You can configure the following policy types for statistics collection:

- accounting policy—specifies the accounting record type and collection interval
- file policy—specifies the storage criteria for accounting statistics files on NEs
- statistics policy—specifies the storage criteria for statistics in the NFM-P
- MIB statistics policy—specifies the collection of MIB-based statistics counters from managed NEs, and is one of the following:
  - NE MIB statistics policy—applies to all NE objects of the specified type
  - specific MIB statistics policy—applies to a specific NE object
- server performance statistics policy—specifies the collection criteria for statistics related to NFM-P server performance

The following table lists the policies required for each statistics type.

Table 14-1 Statistics policies

Statistics type	Policy				
	Accounting	File	Statistics	MIB statistics	Server performance statistics
Accounting • service • network • subscriber • AA	✓	✓	✓	—	—
Performance	—	—	✓	✓	—
Server performance	—	—	✓	—	✓

See the *NSP NFM-P Statistics Management Guide* for information about statistics collection.

See Chapter 15, “Configuration management overview” for information about configuration management. See Chapter 18, “Policy configuration management” for information about configuration of accounting and file policies.

14.5.2 Statistics log policy

The statistics log policy is used for specifying the retention period and alarm thresholds that are associated with a statistics log record.

To compose a modification command for the Statistics Policy, the key input parameters include:

- method—generic.GenericObject.configureInstanceWithResult
- class—package.ObjectStatsLogPolicy (for the example below, the package is bgp and the statistics log policy class is PeerStatsLogPolicy)
- <distinguishedName>—logger:package.ObjectStats

The following figure shows a sample request to modify the administrative state, retention time, and threshold reporting state of the BGP peer statistics log policy.

Figure 14-1 Statistics log policy modification request example

```
<generic.GenericObject.configureInstanceWithResult xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>logger:bgp.PeerStats</distinguishedName>
   <includeChildren>false</includeChildren>
   <configInfo>
      <bgp.PeerStatsLogPolicy>
         <actionMask>
            <bit>modify</bit>
         </actionMask>
         <administrativeState>down</administrativeState>
         <retentionTime>30</retentionTime>
         <thresholdReportingState>down</thresholdReportingState>
```

```
      </bgp.PeerStatsLogPolicy>
   </configInfo>
</generic.GenericObject.configureInstanceWithResult>
```

14.5.3 NE MIB statistics policy

The MIB statistics policy is used for specifying the administrative state, polling synchronization start time, and collection interval of performance statistics. A NE MIB entry policy applies to all objects on the NE that use the MIB entry.

To compose a modification command for the MIB statistics policy, the key input parameters include:

- method—generic.GenericObject.configureInstanceWithResult
- class—snmp.PollerPolicy
- <distinguishedName>—pollerManager:statspolicy-${*id}:product-${productId}:version-${versionId}:oid-${mibEntityOid}

The following figure shows a sample request to modify the administrative state and polling interval of the BGP peer statistics.

Figure 14-2 ME MIB statistics policy modification request example

```
<generic.GenericObject.configureInstanceWithResult xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>pollerManager:statspolicy-1:product-1:version-41:oid-.1.3.6.
1.2.1.15.3.1</distinguishedName>
   <includeChildren>false</includeChildren>
   <configInfo>
      <snmp.PollerPolicy>
         <actionMask>
            <bit>modify</bit>
         </actionMask>
         <administrativeState>up</administrativeState>
         <pollingInterval>30minutes</pollingInterval>
      </snmp.PollerPolicy>
   </configInfo>
</generic.GenericObject.configureInstanceWithResult>
```

14.5.4 Specific MIB statistics policy

The specific MIB statistics policy is a MIB entry policy that applies only to the specified objects on the NE.

To compose a creation command for the specific MIB statistics policy, the key input parameters include:

- method—generic.GenericObject.configureInstanceWithResult
- class—snmp.SpecificStatsPollerPolicy
- <distinguishedName>—pollerManager
- <id> (optional)—if not specified, then it is automatically assigned

- <monitoredClassName>—specify the class (in package.class format) for which to collect performance statistics
- <monitoredObjects>—specify the object for which to collect performance statistics

The following figure shows a sample request to create a specific MIB statistics policy for a specific port.

Figure 14-3 Specific MIB statistics policy request creation example

```
<generic.GenericObject.configureInstanceWithResult xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>pollerManager</distinguishedName>
   <includeChildren>false</includeChildren>
   <configInfo>
      <snmp.SpecificStatsPollerPolicy>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <id>115</id>
         <monitoredClassName>equipment.PhysicalPort</monitoredClassName>
         <monitoredObjects>
            <pointer>network:198.51.100.47:shelf-1:cardSlot-1:card:daughterCardSlot-
1:daughterCard:port-1</pointer>
         </monitoredObjects>
      </snmp.SpecificStatsPollerPolicy>
   </configInfo>
</generic.GenericObject.configureInstanceWithResult>
```

To compose a modification command for the specific MIB statistics policy to enable the collection of specific MIB, see 14.5.3 “NE MIB statistics policy” (p. 178) in this section. The following figure shows a distinguishedName example for modifying a MIB entry.

Figure 14-4 distinguishedName example, MIB entry modification request

```
<distinguishedName>pollerManager:statspolicy-115:product-1:version-41:oid-.1.3.6.
1.2.1.15.3.1</distinguishedName>
```

14.5.5 On-demand statistics collection

Statistics can be collected on-demand. The MIB associated with the selected statistics type is polled and a new real-time current data object is brought into the system. The record type is on-demand, which indicates that the object is not collected using a collection policy. To compose the method for triggering an on-demand statistics collection, the key input parameters include:

- method—generic.GenericObject.triggerCollect
- <instanceNames>—the object instances (FDNs) on which to collect statistics. The example below is collecting statistics for a port.
- <currentDataClasses>—the statistics current data classes for which to collect. The example below is collecting for the interface and interface additional statistics on a port.

You can list several instance names, but you must specify the current data class for each of the instances that contain the statistics to be collected. The following figure shows a sample on-demand statistics collection.

Figure 14-5 On-demand statistics collection request example

```
<generic.GenericObject.triggerCollect xmlns="xmlapi_1.0">
   <instanceNames>
      <string>network:IP_address:shelf-1:cardSlot-1:card:daughterCardSlot-1:
daughterCard:port-4</string>
   </instanceNames>
   <currentDataClasses>
      <string>equipment.InterfaceStats</string>
      <string>equipment.InterfaceAdditionalStats</string>
   </currentDataClasses>
</generic.GenericObject.triggerCollect>
```

For the equivalent behavior of the Collect button on the NFM-P GUI, you can specify one instance in the list with the corresponding data class in the second list. For the equivalent behavior of the Collect All button on the NFM-P GUI, you can specify all relevant instances in the list with all corresponding data classes in the second list. See “Workflow for viewing statistics” in the *NSP NFM-P Statistics Management Guide* for more information about how to use the Collect and Collect All buttons.

Note: If you initiate a non-scheduled collection using the generic.GenericObject.triggerCollect method, a new on-demand current data object might be created, or an on-demand current data object might be updated. When a statistic is first collected, there is an ObjectCreationEvent sent for the new current data record. When subsequent records are collected, the current data record is updated so an AttributeValueChangeEvent is sent. The XML API does not modify the scheduled current data object or create a new historical log record. On-demand statistics are propagated to the history as non-scheduled statistics and do not affect any scheduled statistics retrievals.

14.6 Statistics retrieval methods

14.6.1 Functional description

You can collect statistics data using the following methods:

- registerLogToFile—to continually collect accounting and performance statistics. Use this method for creating accounting and performance statistics export files for specific class types. The NFM-P exports the statistics data to a file after it retrieves the data from an NE, and saves the file in a specified directory on a main or auxiliary server. The NFM-P sends a LogFileAvailableEvent when a file is created using the registerLogToFile method, which is asynchronous. See 14.8 “Statistics retrieval using registerLogToFile” (p. 183) for more information about retrieving statistics using the registerLogToFile method.
- findToFile—to occasionally collect specific accounting or performance statistics. Use this method for infrequent, low-volume transfers of statistics data from the NFM-P database. The findToFile method can save statistics for multiple NEs on one file. The NFM-P sends a FileAvailableEvent

each time that a file is created using the findToFile method. See 14.9 “Statistics retrieval using findToFile” (p. 189) for more information about retrieving statistics using the findToFile method.

- using JMS to monitor statistics events and on-demand statistics collection—see 14.10 “Statistics monitoring using JMS” (p. 194) for more information about statistics monitoring using JMS.

Statistics retrieval using the registerLogToFile or findToFile method requires a RHEL user account on each main or auxiliary server from which statistics are to be collected. Such a user account requires FTP or SFTP access, depending on whether TLS is enabled on the XML API. See 14.6.2 “Configuring FTP and SFTP access for OSS clients” (p. 180) for the specific requirements and best practices for creating OSS client user accounts.

Note: It is recommended that you use the registerLogToFile file method to minimize collection latency and to reduce system load. The findToFile method can be used when less than 400 000 statistics records are retrieved in 15 minutes, and when greater collection latency is acceptable.

Note: AA Cflowd statistics are not available via the XML API, however an external target file server can obtain the statistics from the NSP Flow Collectors. See 14.12 “Collecting Application Assurance (AA) accounting statistics” (p. 197) for more information.

14.6.2 Configuring FTP and SFTP access for OSS clients

Each OSS client that retrieves data files from a main or auxiliary server using FTP or SFTP requires a RHEL user account on the server.

An OSS client user account on a main or auxiliary server has the following requirements.

- The user home directory must be the export directory of the statistics or OAM result files, and is one of the following:
  - on a main server:
    statistics—/opt/nsp/nfmp/server/xml_output
    OAM result files—/opt/nsp/nfmp/server/xml_output/oam
  - on an auxiliary server—/opt/nsp/nfmp/auxserver/xml_output
- The user requires read permissions on the export directory, so must belong to the RHEL nsp user group.

It is recommended that you observe the following best practices when creating an OSS client user account on a main or auxiliary server.

- Apply the same credentials to an OSS client user account that you create on multiple servers.
- For an SFTP user, restrict the RHEL system access to SFTP using the ForceCommand option in the RHEL system configuration for SSH. See the RHEL OS documentation for information.
- For statistics collection, assign a separate directory below *export_directory*/performanceStats and *export_directory*/accountingStats to each OSS client, where *export directory* is one of the following:
  - on a main server—/opt/nsp/nfmp/server/xml_output
  - on an auxiliary server—/opt/nsp/nfmp/auxserver/xml_output

    Note: An OSS client that retrieves both performance and accounting statistics has read access to the statistics files of other OSS clients.

- For OAM result file collection, assign a separate directory below the OAM result export directory to each OSS client.

14.7 Workflow to retrieve statistics data

14.7.1 Stages

Use the registerLogToFile method or the findToFile method to transfer statistics log records from the NFM-P database to an OSS client application.

1

As required, configure the system preferences associated with exported statistics files (applies to the registerLogToFile method only) on an NFM-P server such as the default log file retention time, log file rollover time, or the number of JMS client connection checks. See the system preferences configuration procedures in the *NSP System Administrator Guide* for more information.

2

Construct a SOAP request to perform one of the following:

a. Use the registerLogToFile method to create accounting or performance statistics export files.
   1. Specify the statistics class or classes.
   2. Specify a location for the exported data files.
   3. Specify whether to compress the files.
   4. Specify the attributes to include in the exported data records.
   5. If required, specify a filter to restrict the list of NEs for which data is exported.

b. Use the findToFile method to retrieve specific accounting or performance statistics.
   1. Specify the log record name.
   2. Specify an XML export format.
   3. Specify the name for the exported data file.
   4. Specify a filter to limit the amount of data stored in the file.
   5. Specify whether to compress the file.

3

Send the request.

4

Receive a response or exception to the request.

14.8 Statistics retrieval using registerLogToFile

14.8.1 Overview

You can use the registerLogToFile method to create accounting or performance statistics data files for specific classes. Accounting and performance statistics are collected and stored in the NFM-P database. When the registerLogToFile method is used, the specific statistics are stored in the NFM-P database, and files are exported to the NFM-P main or auxiliary server file system with the requested data for each registered XML API client. Because the two actions occur in parallel, the data is available in files before you can export the data from the database.

An OSS client that retrieves statistics files using the registerLogToFile method requires a RHEL user account on each main or auxiliary server from which statistics are to be collected. Such a user account requires FTP or SFTP access, depending on whether TLS is enabled on the XML API. See 14.6.2 “Configuring FTP and SFTP access for OSS clients” (p. 181) for the specific requirements and best practices for creating OSS client user accounts.

A LogFileAvailableEvent is sent each time a file is ready for retrieval by an OSS client, if the client subscribes to specific JMS topics. See Chapter 4, “Event monitoring using JMS” for information about JMS events.

By default, the NFM-P performs a regular JMS client inactivity check and deregisters any JMS client that does not subscribe to an event stream within a specified time. For registerLogToFile, a JMS subscription is optional; to avoid client deregistration after inactivity, you can disable the JMS client inactivity check. See A.5 “Statistics data retrieval with registerLogToFile and find/findToFile” (p. 333) for information about disabling the JMS client inactivity check.

The span of control associated with the JMS client is applied to each registerLogToFile request at the NE level. If an NE is not within the client span of control, the NFM-P excludes the requested statistics for the NE in the exported files. See the *NSP System Administrator Guide* for more information about span of control.

For scheduled accounting statistics collection, you can configure registerLogToFile system preferences that include the following:

- Log Retention Time (minutes)—the number of minutes to keep the statistics files on disk before purging the files
- JMS Client Registration Grace Intervals—the number of file-creation intervals to wait for a JMS subscriber to reconnect before cancelling the logToFile subscription. The parameter has no effect when the JMS client inactivity check is disabled. An interval is defined as 5 minutes.

For scheduled performance statistics collection, you can configure registerLogToFile system preferences that include the following:

- Log Retention Time (minutes)—the number of minutes to keep the statistics files on disk before purging the files
- Log Rollover Time (minutes)—the number of minutes during which statistics data is written to a file
- JMS Client Registration Grace Intervals—the number of file-creation intervals to wait for a JMS subscriber to reconnect before cancelling the logToFile subscription. The parameter has no effect when the JMS client inactivity check is disabled. An interval is defined as 5 minutes.

You can modify the parameters from the Statistics tab of the System Preferences form in the client GUI. Alternatively, an OSS can use the XML API to modify the retention, rollover, and jmsRetries properties in the log.LogToFileManager class.

See the system preferences configuration procedures in the *NSP System Administrator Guide* for more information.

registerLogToFile filtering

To reduce the volume of exported statistics data, a client can apply a filter during registerLogToFile registration. The filter criteria determine which records are returned to a client; for example, if multiple API clients are individually responsible for statistics collection in a specific subnet; each client can apply a filter to return only the statistics associated with the NEs in specific subnets.

You can filter on any properties of a statistics log record that are string-based, such as monitoredObjectClass or monitoredObjectSiteId. See the LogRecord class definition in the XML API Reference for a list of the class properties.

For accounting statistics, you can configure an NFM-P system preference to include the accounting policy ID in each accounting statistics record. This enables an OSS client to use registerLogToFile and JMS filters based on the accounting policy ID.

See the system preferences configuration procedures in the *NSP System Administrator Guide* for more information.

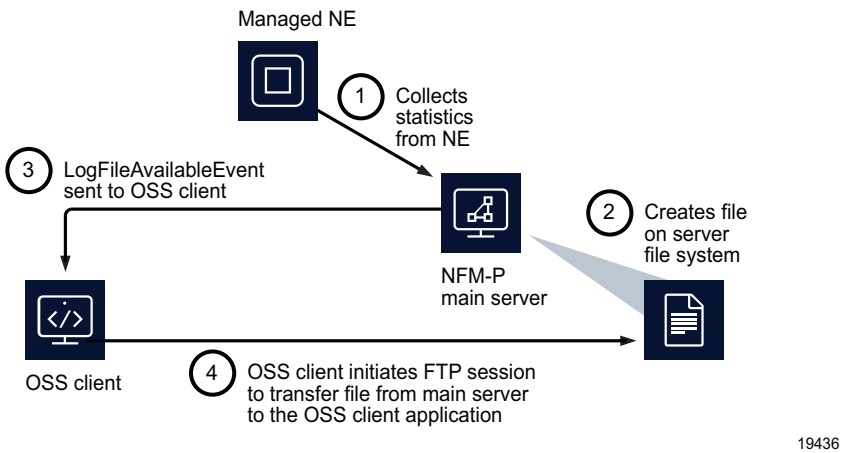
See the registerLogToFile method description in the XML API Reference for a list of the supported filter elements and syntax.

See 4.6 “JMS message filtering” (p. 37) for information about creating JMS filters.

registerLogToFile export from main server

The following figure shows the process associated with the event-driven statistics export from the NFM-P main server after an XML API client registers for notifications using the registerLogToFile method.

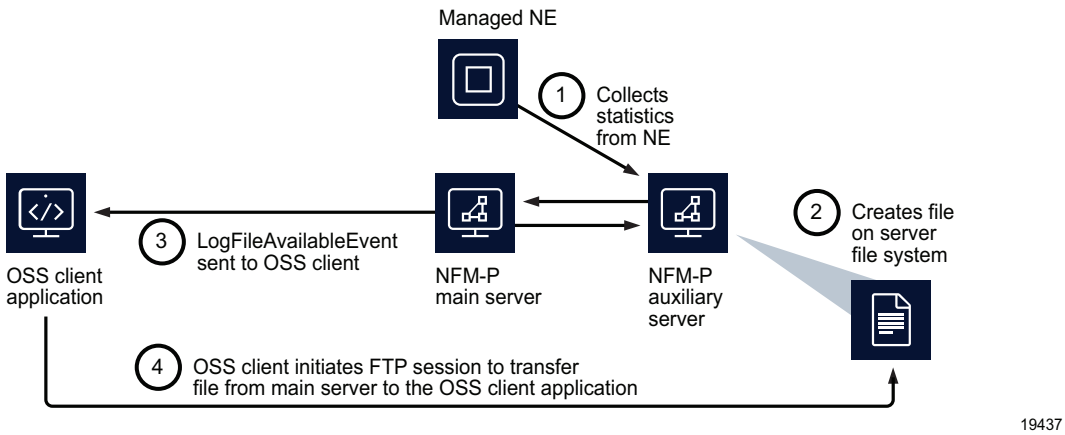
Figure 14-6 Event-driven registerLogToFile process using NFM-P main server



registerLogToFile export from auxiliary server

The following figure shows the process associated with the event-driven statistics export from an NFM-P auxiliary server after an XML API client registers for notifications using the registerLogToFile method.

Figure 14-7 Event-driven registerLogToFile process using NFM-P auxiliary server



14.8.2 Registration request

You can specify multiple accounting or performance statistics classes in a registration request. Table 14-2, “registerLogToFile method input parameters” (p. 185) describes the input parameters of the registerLogToFile method.

Note: The registerLogToFile file compression option requires additional NFM-P server resources. The additional load may prevent the NFM-P from collecting the maximum number of statistics records in the *NSP Planning Guide*.

Table 14-2 registerLogToFile method input parameters

Input parameter	Description	Values
fullClassName	Package qualified class name in dot-separated format	A comma-separated list of accounting statistics classes, performance statistics classes, or both

Table 14-2 registerLogToFile method input parameters (continued)

Input parameter	Description	Values
dirName	The relative path of the directory that is to contain the exported files. The path is relative to the accountingStats or performanceStats below the XML output directory that is specified during NFM-P server installation or upgrade. Nokia recommends that you use a separate directory for each application that requires exported statistics. Statistics classes for the same accounting file and jmsClientId cannot be sent to different directories. If statistics classes from the same file have different registrations to different directories, collection will not separate the classes to different directories. The statistics classes cannot be replicated in another registerLogToFile request to a different directory using the same jmsClientId. Each subsequent registerLogToFile request for the same statistics class and jmsClientId will overwrite the previous request.	*String*
compress	Optional parameter that specifies whether data files are compressed when they are exported If you set this parameter to true, the NFM-P statistics collection performance may be affected.	The options are: • true—compresses data files using gzip; each exported file has a gz extension • false (default)—does not compress exported files
jmsClientId	The JMS client ID or Kafka Subscription ID to be notified when a new export file is ready for retrieval	*String* See Chapter 4, “Event monitoring using JMS” for information about JMS client ID formats.
filter	Optional parameter that uses filter elements to specify the set of NEs for which data records are exported; you can use the parameter to reduce the volume of returned information	<filter>*element*</filter> See the registerLogToFile method description in the XML API Reference for a list of the supported filter elements and regular-expression syntax.
resultFilter	Optional parameter that specifies the attributes to include in the exported data records; you can use the parameter to reduce the size of the exported files	<attribute>*attribute_name*</attribute>
continuOnFailure	Optional parameter that specifies whether to continue processing requests in the stream after an exception occurs	The options are: • true—processing continues after an exception • false (default)—processing stops after an exception

The optional filter input parameter for registerLogToFile allows filtering on the monitoredObjectSiteName attribute of the log record that is written to the statistics file. The following filter tags are supported:

- and
- equal
- in
- not
- notEqual
- notSuperset
- or
- regex
- subset
- superset
- wildcard

The following figure shows a registerLogToFile filter that allows log records for objects whose siteName begins with toro, vanc, or mont, and is followed by one to eight non-digits, wan, and one or more digits.

Figure 14-8 regex filter example

```
<filter>
   <regex name="monitoredObjectSiteName" value="^(toro|vanc|mont)[^0-9]{1,8}wan[0-
9]+"/>
</filter>
```

The following figure uses an or tag to group three regular expressions that achieve the same results as in the previous figure. The regular expressions would each be evaluated against the same data set and the intermediate results would be merged together to produce the final result.

Figure 14-9 regex filter example using or expression

```
<filter>
   <or>
      <regex name="monitoredObjectSiteName" value="^toro[^0-9]{1,8}wan[0-9]+"/>
      <regex name="monitoredObjectSiteName" value="^vanc[^0-9]{1,8}wan[0-9]+"/>
      <regex name="monitoredObjectSiteName" value="^mont[^0-9]{1,8}wan[0-9]+"/>
   </or>
</filter>
```

Note: Regular expressions are case-sensitive. Case insensitivity flags may be introduced within the regular expression itself. For more information about Java regular expression syntax, refer to the Javadoc description (since Java 1.4) of class java.util.regex.Pattern.

The following figure shows a registerLogToFile request for performance statistics.

Figure 14-10 registerLogToFile request example, performance statistics

```
<registerLogToFile xmlns="xmlapi_1.0">
   <fullClassName>equipment.SystemCpuStats</fullClassName>
   <dirName>ossclient</dirName>
   <compress>false</compress>
   <jmsClientId>JMS_client@n</jmsClientId>
   <resultFilter>
      <attribute>systemCpuUsage</attribute>
      <attribute>monitoredObjectPointer</attribute>
      <attribute>monitoredObjectSiteId</attribute>
```

```
      <children/>
   </resultFilter>
</registerLogToFile>
```

One statistics file is created per statistics class for performance statistics and one statistics file is created per NE per record type, which translates to a statistics class, for accounting statistics. The statistics export files are created on the NFM-P main or auxiliary server that performs the statistics collection. The name of each exported file is unique because it is created using the requesting client ID and the current timestamp.

Statistics file creation frequency

The rollover period for accounting statistics files is equivalent to the NE file policy rollover time. See "To configure a file policy" in the *NSP NFM-P Statistics Management Guide* for more information about modifying a file policy.

You can use the NFM-P GUI to manage the rollover period for performance statistics files. See the system preferences configuration procedures in the *NSP System Administrator Guide* for more information.

Note: A statistics file export can generate a large number of files and use considerable disk space. The NFM-P periodically deletes aged files to control disk space usage. You can use the NFM-P GUI to manage the file retention time. See the procedure to configure the statistics data retention period for the main database in the *NSP System Administrator Guide* for more information.

14.8.3 XML statistics output file

Statistics retrieved from the registerLogToFile method in an XML format contain a set of elements and attributes that define the statistic class.

The following figure shows an extract from the XML output file for the statistic counter in the equipment.SystemCpuStats statistic object from the request in previous figure.

Figure 14-11 registerLogToFile statistics response example

```
<logToFileResponse xmlns="xmlapi_1.0">
   <equipment.SystemCpuStatsLogRecord>
      <systemCpuUsage>5</systemCpuUsage>
      <monitoredObjectPointer>network:198.51.100.60:shelf-1:systemStatsHolder</
monitoredObjectPointer>
      <monitoredObjectSiteId>198.51.100.60</monitoredObjectSiteId>
   </equipment.SystemCpuStatsLogRecord>
</logToFileResponse>
```

14.8.4 Stopping statistics file export from registerLogToFile

An XML API client can deregister from receiving notifications for accounting and performance statistics and stop the export file creation by sending a deregisterLogToFile request. A deregisterLogToFile request requires only the JMS client ID as a parameter.

The following figure shows a sample deregisterLogToFile request.

Figure 14-12 deregisterLogToFile request example

```
<deregisterLogToFile xmlns="xmlapi_1.0">
   <!-- fullClassName is optional -->
   <fullClassName> equipment.SystemCpuStats </fullClassName>
   <jmsClientId>JMS_client@n</jmsClientId>
</deregisterLogToFile>
```

An XML API client is also deregistered when the client JMS session closes. The NFM-P checks JMS client connectivity every 5 minutes; you can use a GUI client to configure the maximum number of intervals before a disconnected client is deregistered. See the system preferences configuration procedures in the *NSP System Administrator Guide*.

A deregistered JMS client must re-register to receive notifications.

14.8.5 Statistics recovery after OSS connection loss

If an OSS client loses connectivity to an NFM-P main or auxiliary server that is collecting statistics, the OSS needs to retrieve the registerLogToFile export files when it reconnects. The files are still available for retrieval if the log retention time does not elapse before the client reconnects. See the *NSP System Administrator Guide* for more information about configuring the log retention time.

14.8.6 Statistics recovery after JMS client unsubscribed

If an OSS client JMS subscription becomes unsubscribed, the NFM-P deregisters the registerLogToFile session and stops creating export files after a configured time interval. Contact Nokia technical support for information about configuring the time interval.

14.9 Statistics retrieval using findToFile

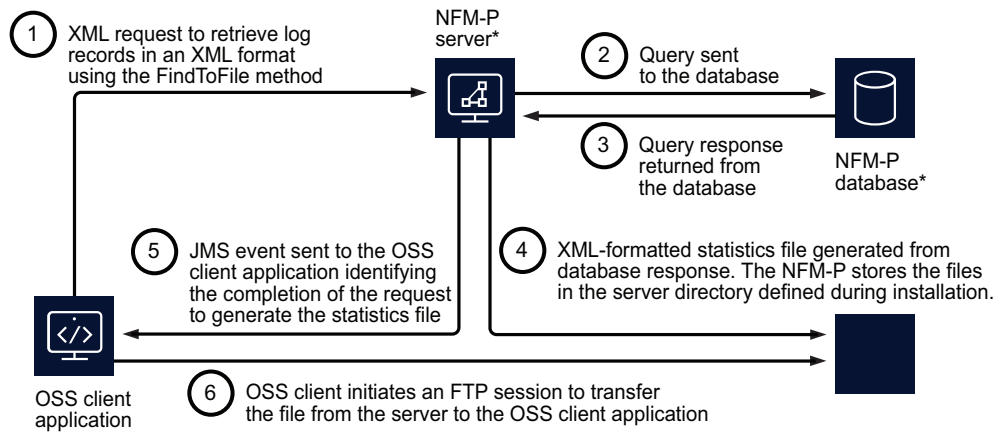
14.9.1 Overview

You can use the findToFile method to transfer accounting and performance statistics log records from the NFM-P to an OSS client. The method can save statistics from multiple NEs in one file. The NFM-P sends a FileAvailableEvent message each time a statistics file becomes available.

An OSS client that retrieves statistics files using the findToFile method requires a RHEL user account on each main server. Such a user account requires FTP or SFTP access, depending on whether TLS is enabled on the XML API. See 14.6.2 “Configuring FTP and SFTP access for OSS clients” (p. 181) for the specific requirements and best practices for creating OSS client user accounts.

The following figure shows the process associated with the statistics retrieval using the findToFile method.

Figure 14-13 findToFile statistics retrieval process



Note: In addition to the illustrated scenario, step 4 of Figure 14-13, “findToFile statistics retrieval process” (p. 190) can result in the following.

- If a file name is specified, a write file is created.
- If an FTP address is specified, the NFM-P server sends the file to a URL without creating an intermediate file.
- If the file location is on a remote disk that is read directly by the client application, the application may read or copy the file.

Note: The OSS application is responsible for managing files and disk space associated with the requests.

Note: The NFM-P maps the object FDNs to a corresponding database index. It is recommended to use FDN properties in queries to maximize the query processing speed. The XML API queries should also use concrete classes, rather than an abstract class, to optimize performance.

14.9.2 Statistics retrieval examples using the findToFile method

Figure 14-14, “findToFile request example, performance statistics” (p. 191) shows a sample request to perform the following tasks using the findToFile method:

- interface statistics collection
- storage of the collected statistics in XML format

Figure 14-14 findToFile request example, performance statistics

```
<SOAP:Body>
   <findToFile xmlns="xmlapi_1.0">
      <fullClassName>equipment.InterfaceAdditionalStats</fullClassName>
      <filter>
         <or>
            <equal name="monitoredObjectSiteId" value="10.1.202.95"/>
            <equal name="monitoredObjectSiteId" value="10.1.202.94"/>
         </or>
      </filter>
      <resultFilter>
         <attribute>receivedTotalOctetsPeriodic</attribute>
         <attribute>transmittedTotalOctetsPeriodic</attribute>
      </resultFilter>
      <fileName>Equipment.InterfaceAdditionalStatsLogRecord.xml</fileName>
   </findToFile>
</SOAP:Body>
```

The following table describes the input parameters of the findToFile method.

Table 14-3 findToFile method input parameters

Input parameter	Description	Values
fullClassName	Package qualified class name in dot-separated format	A comma-separated list of accounting statistics classes, performance statistics classes, or both
fileName	The relative file in which to store the results. The format is: [s]ftp://[*user:password*@]host[*:port*]/directory/file	The following format rules apply. • For a remote client, the directory is relative to the default client FTP directory. • The / and ; characters are reserved, so must be encoded. For example, to specify a file in the /tmp directory on the client, the path is ftp://name@password/%2Ftmp/filename.xml. The path ftp://name@password/tmp/filename.xml specifies a file in a /tmp directory under the client FTP directory. See *RFC 1738 Uniform Resource Locators (URL), Section 3.2.2* for more information about the FTP URL path requirements.
timeStamp	Include the timestamp in the name of the result file.	The options are: • true • false (default)
compress	Optional parameter that specifies whether data files are compressed when they are exported. If you set this parameter to true, the NFM-P statistics collection performance may be affected.	The options are: • true—compresses data files using gzip; each exported file has a gz extension • false (default)—does not compress exported files

Table 14-3 findToFile method input parameters (continued)

Input parameter	Description	Values
timeout	The time, in ms, after which the operation times out. The request is aborted and an exception is returned in the FileAvailableEvent notification. The result file may be incomplete.	The default is 0, which means no timeout.
filter	Optional but strongly recommended parameter that uses filter elements based on properties of the class specified by the fullClassName parameter	<filter>*element*</filter> See the findToFile method description in the XML API Reference for a list of the supported filter elements and regular-expression syntax. It is recommended to filter only the relevant attributes and to use the timeCaptured attribute to limit the scale of logRecords for efficiency.
resultFilter	Optional but strongly recommended parameter that filters to restrict the information returned per object	<attribute>*attribute_name*</attribute>
continuOnFailure	Optional parameter that specifies whether to continue processing requests in the stream after an exception occurs.	The options are: • true—processing continues after an exception • false (default)—processing stops after an exception
synchronous	Optional parameter that specifies that the method runs synchronously; the result is returned only after the method completes.	The options are: • true—processing continues after an exception • false (default)—processing stops after an exception

To return the data in a streamed XML result, you can use the <find> method with a similar set of filters. This method of statistics retrieval is recommended for testing, not for periodic retrieval. See Chapter 13, “Inventory management” for more information.

When you create export files using the findToFile method:

- the export files are placed in a directory that is specified during the NFM-P server installation
- an FTP server is not installed during the NFM-P database installation. To export files from the specified directory, an FTP server can be installed on the same machine as the database, if required. If the files are on a remote disk, an FTP server may not be required to access files.

An exception-free response indicates that the request was received and validated.

14.9.3 Sample request to retrieve historical statistics from an NE

The following figure shows a sample request to retrieve historical statistics from an NE using the findToFile method. The sample uses the timeCaptured attribute to limit the number of logRecords to be retrieved on a specific port only.

Figure 14-15 findToFile request example, historical NE statistics

```
<SOAP:Body>
   <findToFile xmlns="xmlapi_1.0">
      <fullClassName>equipment.InterfaceAdditionalStatsLogRecord</fullClassName>
      <filter>
         <and>
            <equal name="monitoredObjectPointer" value="network:10.1.202.93:shelf-
1:cardSlot-1:card:daughterCardSlot-1:daughterCard:port-3"/>
            <between name="timeCaptured" first="1127142900000" second="1127143800000"/>

         </and>
      </filter>
      <fileName>Equipment.InterfaceAdditionalStatsLogRecord.xml</fileName>
   </findToFile>
</SOAP:Body>
```

14.9.4 XML statistics output file

Statistics retrieved from the NFM-P database in an XML format contain a set of elements and attributes that define the statistic class.

The following figure shows an extract from the XML output file for the statistic counter in the equipment.InterfaceAdditionalStatsLogRecord statistic object from the request in the previous figure.

Figure 14-16 Statistics XML output file example

```
<findToFileResponse xmlns="xmlapi_1.0">
   <equipment.InterfaceAdditionalStatsLogRecord>
      <monitoredObjectClass>equipment.PhysicalPort</monitoredObjectClass>
      <monitoredObjectPointer>network:10.1.202.93:shelf-1:cardSlot-1:card:
daughterCardSlot-1:daughterCard:port-3</monitoredObjectPointer>
      <displayedName>Port 1/1/3</displayedName>
      <monitoredObjectSiteId>10.1.202.93</monitoredObjectSiteId>
      <monitoredObjectSiteName>sim202_93</monitoredObjectSiteName>
      <timeCaptured>1127878285113</timeCaptured>
      <periodicTime>938610</periodicTime>
      <suspect>false</suspect>
      <objectFullName>equipment.InterfaceAdditionalStatsLogRecord.equipment.
InterfaceAdditionalStats30-346</objectFullName>
      <name>equipment.InterfaceAdditionalStats30-346</name>
      <createdOnPollType>ScheduledFullNodeResync</createdOnPollType>
      <updatedOnPollType>ScheduledFullNodeResync</updatedOnPollType>
      <recordId>346</recordId>
      <bucketId>30</bucketId>
      <deploymentState>0</deploymentState>
      <receivedTotalOctets>0</receivedTotalOctets>
      <receivedTotalOctetsPeriodic>0</receivedTotalOctetsPeriodic>
      <receivedUnicastPackets>0</receivedUnicastPackets>
```

```
      <receivedUnicastPacketsPeriodic>0</receivedUnicastPacketsPeriodic>
      <receivedMulticastPackets>0</receivedMulticastPackets>
      <receivedMulticastPacketsPeriodic>0</receivedMulticastPacketsPeriodic>
      <receivedBroadcastPackets>0</receivedBroadcastPackets>
      <receivedBroadcastPacketsPeriodic>0</receivedBroadcastPacketsPeriodic>
      <transmittedTotalOctets>0</transmittedTotalOctets>
      <transmittedTotalOctetsPeriodic>0</transmittedTotalOctetsPeriodic>
      <transmittedUnicastPackets>0</transmittedUnicastPackets>
      <transmittedUnicastPacketsPeriodic>0</transmittedUnicastPacketsPeriodic>
      <transmittedMulticastPackets>0</transmittedMulticastPackets>
      <transmittedMulticastPacketsPeriodic>0</transmittedMulticastPacketsPeriodic>
      <transmittedBroadcastPackets>0</transmittedBroadcastPackets>
      <transmittedBroadcastPacketsPeriodic>0</transmittedBroadcastPacketsPeriodic>
      <children-Set/>
   </equipment.InterfaceAdditionalStatsLogRecord>
</findToFileResponse>
```

14.9.5 Statistics recovery after OSS connection loss

If an OSS loses connectivity to the NFM-P server and statistics records are not being retrieved, the OSS client needs to record the time of the most recent statistics retrieval. When the connectivity is restored, the OSS client must send specific findToFile requests for the missed statistics data. The OSS must be aware of the different Statistics Policy retention times so that it does not request older log records that have been deleted from the NFM-P database.

14.9.6 Missing performance statistics

It is possible that in between NFM-P performance statistics collection intervals, there are no statistics detected by the OSS. This means that performance statistics log records are missing from the NFM-P database. If these statistics records are missing, the NFM-P server does not attempt to retry collection from the NEs. Therefore, an OSS client does not need to attempt to retrieve the missing statistics log records for the collection interval, and can continue to process statistics for the next collection interval.

14.10 Statistics monitoring using JMS

14.10.1 Overview

An OSS performance management application can use a JMS interface to communicate with the NFM-P server for monitoring statistics events. The OSS needs to subscribe to the NFM-P-topic-xml for all JMS topics including statistics events or to the NFM-P-topic-xml-stats JMS topic for events related to statistics collection. See Chapter 4, “Event monitoring using JMS” for more information about connecting to the JMS event stream and subscribing to JMS topics.

For scheduled accounting or performance statistics collection, JMS StatsEvent notifications are generated to display the beginning and end of polling of an NE for accounting or performance statistics. The JMS StatsEvent indicates statistics collection activity. The following table describes the parameters in the scheduled performance statistics collection StatsEvent output.

Table 14-4 JMS StatsEvent parameters

Parameter	Description	Values
<state>	Marks the start or end of polling for an NE	• begin • end
<statsType>	Statistics type	• polling • accounting
<networkElement>	NE identifier	Site ID of the NE
<time>	Collection time	Unix Epoch time, in milliseconds

Figure 14-17 StatsEvent output example, scheduled performance statistics collection

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>StatsEvent</eventName>
         <ALA_category>STATISTICS</ALA_category>
         <ALA_OLC>0</ALA_OLC>
         <ALA_isVessel>false</ALA_isVessel>
         <ALA_allomorphic/>
         <MTOSI_osTime>1333998300824</MTOSI_osTime>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <MTOSI_objectName/>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <MTOSI_objectType>StatsEvent</MTOSI_objectType>
         <ALA_eventName>StatsEvent</ALA_eventName>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <statsEvent>
            <state>begin</state>
            <networkElement>198.51.100.57</networkElement>
            <statsType>polling</statsType>
            <time>1333998300824</time>
         </statsEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

For on-demand statistics collection, an ObjectCreationEvent or AttributeValueChangeEvent notification is generated to display the statistics class that is polled and the attributes and values of the class. When a statistic is first collected, there is an ObjectCreationEvent sent for the new current data record. When subsequent records are collected, the current data record is updated so an AttributeValueChangeEvent is sent. The following figure shows sample on-demand statistics collection ObjectCreation output.

Figure 14-18 ObjectCreation output example, on-demand statistics collection

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>ObjectCreation</eventName>
         <ALA_category>STATISTICS</ALA_category>
         <ALA_OLC>0</ALA_OLC>
         <ALA_isVessel>false</ALA_isVessel>
         <ALA_allomorphic>equipment.InterfaceStats</ALA_allomorphic>
         <MTOSI_osTime>1334089752549</MTOSI_osTime>
         <MTOSI_NTType>NT_OBJECTCREATION</MTOSI_NTType>
         <MTOSI_objectName>logger:real-time|equipment.InterfaceStats|network_198.
51.100.57_shelf-1_cardSlot-1_card_daughterCardSlot-1_daughterCard_port-1</MTOSI_
objectName>
         <ALA_span>:0:</ALA_span>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <MTOSI_objectType>equipment.InterfaceStats</MTOSI_objectType>
         <ALA_eventName>ObjectCreation</ALA_eventName>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <objectCreationEvent>
            <equipment.InterfaceStats>
               <receivedOctets>144677782</receivedOctets>
               <receivedOctetsPeriodic>0</receivedOctetsPeriodic>
               <receivedUnicastPackets>637046</receivedUnicastPackets>
               <receivedUnicastPacketsPeriodic>0</receivedUnicastPacketsPeriodic>
               <receivedPacketsDiscarded>0</receivedPacketsDiscarded>
               <receivedPacketsDiscardedPeriodic>0</receivedPacketsDiscardedPeriodic>

               <receivedBadPackets>0</receivedBadPackets>
               <receivedBadPacketsPeriodic>0</receivedBadPacketsPeriodic>
               <receivedUnknownProtocolPackets>0</receivedUnknownProtocolPackets>
               <receivedUnknownProtocolPacketsPeriodic>0</receivedUnknownProtocol
PacketsPeriodic>
               <transmittedOctets>198584536</transmittedOctets>
               <transmittedOctetsPeriodic>0</transmittedOctetsPeriodic>
               <transmittedUnicastPackets>553327</transmittedUnicastPackets>
               <transmittedUnicastPacketsPeriodic>0</transmittedUnicastPacketsPeriodic>

               <outboundPacketsDiscarded>0</outboundPacketsDiscarded>
               <outboundPacketsDiscardedPeriodic>0</outboundPacketsDiscardedPeriodic>

               <outboundBadPackets>0</outboundBadPackets>
               <outboundBadPacketsPeriodic>0</outboundBadPacketsPeriodic>
               <timeCaptured>1334089752528</timeCaptured>
```

```
               <periodicTime>0</periodicTime>
               <monitoredObjectClass>equipment.PhysicalPort</monitoredObjectClass>
               <monitoredObjectPointer>network:198.51.100.57:shelf-1:cardSlot-1:card:
daughterCardSlot-1:daughterCard:port-1</monitoredObjectPointer>
               <alarmedObjectClass>equipment.PhysicalPort</alarmedObjectClass>
               <alarmedObjectPointer>network:198.51.100.57:shelf-1:cardSlot-1:
card:daughterCardSlot-1:daughterCard:port-1</alarmedObjectPointer>
               <logRecordClass>equipment.InterfaceStatsLogRecord</logRecordClass>
               <displayedName>Port 1/1/1</displayedName>
               <monitoredObjectSiteId>198.51.100.57</monitoredObjectSiteId>
               <monitoredObjectSiteName>sim20_57</monitoredObjectSiteName>
               <suspect>false</suspect>
               <createdOnPollType>SelectiveResync</createdOnPollType>
               <updatedOnPollType>SelectiveResync</updatedOnPollType>
               <historyCreated>false</historyCreated>
               <deploymentState>0</deploymentState>
               <objectFullName>logger:real-time|equipment.InterfaceStats|network_198.
51.100.57_shelf-1_cardSlot-1_card_daughterCardSlot-1_daughterCard_port-1</
objectFullName>
               <name>Port 1/1/1</name>
               <selfAlarmed>false</selfAlarmed>
               <children-Set/>
            </equipment.InterfaceStats>
         </objectCreationEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

14.11 Collecting NFM-P performance statistics

14.11.1 Overview

You can use the NFM-P to collect the following server performance statistics:

- XML API Find Requests
- XML API Find To File Requests
- XML API Requests

The statistics are under the statistics package. See the XML API Reference for information about specific classes. See “Server performance statistics collection” in the *NSP NFM-P Statistics Management Guide* for information about collecting server performance statistics, and for a description of each performance statistics counter.

14.12 Collecting Application Assurance (AA) accounting statistics

14.12.1 Overview

AA accounting statistics provide information about application use on a SAP or spoke SDP, or by a subscriber in a network.

As least one ISA-AA MDA is required to collect AA accounting statistics.

The NFM-P can collect the following AA statistics types:

- AA application
- AA application group
- AA protocol
- AA subscriber protocol (special-study)
- AA subscriber application (special-study)
- AA subscriber custom record

See “Statistics collection in the NFM-P” in the *NSP NFM-P Statistics Management Guide* for general information about configuring and collecting accounting statistics.

See “To configure an AA accounting policy” in the *NSP NFM-P Classic Management User Guide* for more information.

See “Workflow for accounting statistics collection” in the *NSP NFM-P Statistics Management Guide* for the procedure to collect AA statistics.

The statistics can be viewed in graphical or tabular form using the NFM-P GUI, forwarded to a third-party server for processing by an OSS or third-party application, and used for reporting in NSP Analytics reports.

The NFM-P does not store the AA accounting statistics data in its database. Instead, the NFM-P exports the data in IPDR format. The exported files are stored in the aaAccountingStats directory below the OSS XML output directory on a main or auxiliary server station.

See “To configure AA accounting file export” in the *NSP NFM-P Classic Management User Guide* for more information about configuring an AA accounting policy.

The NFM-P can also collect AA network performance statistics to monitor the AA Cflowd processing load on an ISA-AA module. The procedure to configure and collect AA network performance statistics is similar to AA accounting statistics and statistics data files are stored in aaAccountingStats directory below the OSS XML output directory on a main or auxiliary server station. Follow the procedure described in “To configure an AA accounting policy” in the *NSP NFM-P Classic Management User Guide* and select statistics type AA Performance.

See the IPDR Reference for comprehensive information about AA accounting and AA network performance statistics types, counters, and IPDR file names for each statistics types.

14.13 Collecting flow statistics

14.13.1 Overview

Cflowd statistics are based on sampled flows. A flow is a series of IP packets that share a common source, destination, and type of payload; for example, traffic that is specific to an application.

There are two types of Cflowd statistics:

- System Cflowd collects traffic on network interfaces, IES/VPRN L3 access interfaces, IES/VPRN tunnel interfaces and VPRN network interfaces.

- AA Cflowd collects traffic for AA applications and application groups associated with ISA-AA groups or partitions.

The NFM-P does not collect system Cflowd statistics, instead a third-party Cflowd collector is used. To collect System Cflowd statistics, you must enable Cflowd globally on an NE and configure Cflowd collectors on the NE. See “To enable and configure global Cflowd sampling on an NE” in the *NSP NFM-P Classic Management User Guide* for more information.

AA Cflowd statistics are collected from ISA-AA MDAs. You use the NFM-P to create AA Cflowd group policies to specify the AA Cflowd collection criteria. See “To configure an AA Cflowd group policy” in the *NSP NFM-P Classic Management User Guide* for information.

The NFM-P does not store AA accounting statistics in the NFM-P database. Instead, the NFM-P saves the statistics data in files for reporting in NSP Analytics reports, or for IPDR record processing by an OSS application. Forwarding to a target file server can be configured using the NSP Flow Collector web UI.

See “Workflow to configure flow statistics collection” in the *NSP NFM-P Statistics Management Guide* for more information.

See the IPDR Reference for comprehensive information about AA Cflowd statistics types, counters, and IPDR file names for each statistics types.

14.14 Collecting JMS performance statistics

14.14.1 Overview

You can use the NFM-P to collect the following JMS server performance statistics:

- JMS Subscriber Topic
- JMS Durable XML Subscriber Session
- Publisher Map Event
- Publisher Object Event
- Publisher Queue
- Publisher Realtime Event
- Publisher SAMC Event
- Publisher XML Event

The statistics are under the statistics package. See the XML API Reference for information about specific classes. See “Server performance statistics collection” in the *NSP NFM-P Statistics Management Guide* for information about collecting server performance statistics, and for a description of each JMS performance statistics counter.

Real-time collection of JMS performance statistics is not supported because of the potential performance effect on the NFM-P.

14.15 Third-party applications for processing statistics

14.15.1 Overview

You can use an in-house statistics-processing application or a third-party vendor application to manage the statistics information that the NFM-P collects.

You can obtain a list of Nokia-certified third-party OSS products from the Nokia NSP Connected Partner Program.

15 Configuration management overview

15.1 Configuration management

15.1.1 Overview

OSS applications can use the XML API to perform configuration operations on NFM-P-managed network objects and their associated child objects. This is achieved by sending XML requests that allow the OSS to create, delete, or modify existing objects. These requests can also extend to the object's children at the same time, if required.

Note: The configuration operations that are allowed on each object is defined in the XML schema. See the Schema Reference. See the XML API Reference for more information about the object hierarchies (relationship between parent and child objects).

The OSS user account privileges define the access and restrictions for configuration operations in NFM-P functional areas. See the chapter on NFM-P user security in the *NSP System Administrator Guide* for information about configuring user accounts and assigning privileges.

Most of the configuration operations that can be performed on the functional areas of the NFM-P using the GUI can also be performed using the XML API. The main areas for OSS configuration management include:

- device configuration—for example, routers, cards, ports, GNE, and channels. See Chapter 16, "Device configuration management" for more information.
- network configuration—for example, protocols, MPLS, service tunnels, and VRRP. See Chapter 17, "Network configuration management" for more information.
- policy configuration—for example, service policy, routing policy, and network policy. See Chapter 18, "Policy configuration management" for more information.
- service configuration—for example, VLL, VPLS, VPRN, VLAN, mirror composite, and customer/subscriber. See Chapter 19, "Service configuration management" for more information.
- scripts and template configuration—See Chapter 20, "Script and template configuration management" for more information.

15.2 Configuration methods

15.2.1 Overview

CAUTION

Service Disruption

Configuration and deletion methods may have unintended side effects.

For example, modifying some port configurations may result in services being deleted if they are no longer supported by the new configuration. It is recommended that testing be performed so that the OSS developer fully understands what side effects may happen and to ensure that side effects are handled appropriately.

CAUTION

Service Disruption

The messages in this section are examples of the SOAP XML request format.

Use the sample as a base to build your request. Ensure that you test your request before network deployment.

The XML API uses the generic package to define methods that are applicable to all objects. Methods for object configuration (create, modify, delete) are provided by the generic.GenericObject class. The most commonly used configuration methods from this class are described in the following table.

Table 15-1 Object configuration methods using the generic.GenericObject class

Method	Description
generic.GenericObject.configureInstance	Modifies an existing object and associated children. Not recommended for OSS clients as there are no return parameters.
generic.GenericObject.configureInstanceWithResult	Modifies an existing object and associated children. The method returns the newly configured parameters on the object and child, including parameters set due to associated conditions. See the XML API Reference for input parameters. See Figure 15-4, “Port configuration response example” (p. 206) for an example of the method.
generic.GenericObject.configureChildInstance	Creates a child of the object with the desired configuration values for the child or children objects, and optional grandchildren. See the XML API Reference for input parameters.
generic.GenericObject.configureChildInstanceWithResult	Creates a child of an existing object. The method returns the newly configured parameters on the child object and grandchildren, including parameters set due to associated conditions. See the XML API Reference for input parameters. See Figure 15-5, “Epipe creation request example” (p. 207) for an example of the method.

Table 15-1 Object configuration methods using the generic.GenericObject class (continued)

Method	Description
generic.GenericObject.deleteInstance	Delete an existing object. See Figure 15-7, “Epipe deletion request example” (p. 208) for an example of the method.

The configureInstanceWithResult and configureChildInstanceWithResult methods accept the resultFilter tag as an input parameter. See 6.2 “Result filtering” (p. 77) for more information about result filtering in the XML API.

The resultFilter tag can be used to return only a subset of attributes for the newly configured object, its children, and grandchildren. The resultFilter returns only four attributes of the created epipe.Epipe service object instead of every attribute.

Figure 15-1 Epipe service creation example using configureChildInstanceWithResult

```
<generic.GenericObject.configureChildInstanceWithResult xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>svc-mgr</distinguishedName>
   <childConfigInfo>
      <epipe.Epipe>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <id>70000</id>
         <displayedName>Epipe Example 1</displayedName>
         <description>Epipe Example 1</description>
         <subscriberPointer>subscriber:1</subscriberPointer>
      </epipe.Epipe>
   </childConfigInfo>
   <resultFilter>
      <attribute>objectFullName</attribute>
      <attribute>administrativeState</attribute>
      <attribute>displayedName</attribute>
      <attribute>olcState</attribute>
   </resultFilter>
</generic.GenericObject.configureChildInstanceWithResult>
<generic.GenericObject.configureChildInstanceWithResultResponse xmlns="xmlapi_1.
0">
   <childConfigInfo>
      <epipe.Epipe>
         <displayedName>Epipe Example 1</displayedName>
         <administrativeState>up</administrativeState>
         <olcState>maintenance</olcState>
         <objectFullName>svc-mgr:service-70000</objectFullName>
         <children-Set/>
      </epipe.Epipe>
   </childConfigInfo>
</generic.GenericObject.configureChildInstanceWithResultResponse>
```

The following figure shows the modification of a VPLS site name. The includeChildren tag equal true returns all children of the configured object. The nested resultFilter limits the number of returned attributes for the vpls.Site and nested child objects.

Figure 15-2 VPLS site name modification example using configureChildInstanceWithResult

```
<generic.GenericObject.configureInstanceWithResult xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>svc-mgr:service-25:198.51.100.71</distinguishedName>
   <includeChildren>true</includeChildren>
   <configInfo>
      <vpls.Site>
         <actionMask>
            <bit>modify</bit>
         </actionMask>
         <displayedName>Example Site A-9</displayedName>
      </vpls.Site>
   </configInfo>
   <resultFilter>
      <attribute>name</attribute>
      <attribute>objectFullName</attribute>
      <attribute>serviceName</attribute>
      <attribute>siteId</attribute>
      <children>
         <resultFilter class="vpls.SiteMldSnooping">
            <attribute>administrativeState</attribute>
            <attribute>reportSrcAddressType</attribute>
         </resultFilter>
         <resultFilter class="vpls.L2AccessInterface">
            <attribute>ingressPolicyName</attribute>
            <attribute>egressPolicyName</attribute>
            <attribute>name</attribute>
            <children>
               <resultFilter class="vpls.L2AccessInterfaceMldSnpgCfg">
                  <attribute>genQueryInterval</attribute>
                  <attribute>serviceId</attribute>
                  <attribute>lastMemberInterval</attribute>
               </resultFilter>
            </children>
         </resultFilter>
      </children>
   </resultFilter>
</generic.GenericObject.configureInstanceWithResult>
<generic.GenericObject.configureInstanceWithResultResponse xmlns="xmlapi_1.0">
   <configInfo>
      <vpls.Site>
         <serviceName>Service Name Change2</serviceName>
         <siteId>198.51.100.71</siteId>
         <objectFullName>svc-mgr:service-25:198.51.100.71</objectFullName>
```

```
         <name>Example Site A-9</name>
         <children-Set>
            <vpls.SiteMldSnooping>
               <administrativeState>disabled</administrativeState>
               <reportSrcAddressType>ipv6</reportSrcAddressType>
               <children-Set/>
            </vpls.SiteMldSnooping>
            <vpls.L2AccessInterface>
               <ingressPolicyName>Access Ingress-1</ingressPolicyName>
               <egressPolicyName>Access Egress-1</egressPolicyName>
               <name>Port 1/1/3:4.6</name>
               <children-Set>
                  <vpls.L2AccessInterfaceMldSnpgCfg>
                     <serviceId>10</serviceId>
                     <genQueryInterval>125</genQueryInterval>
                     <lastMemberInterval>10</lastMemberInterval>
                     <children-Set/>
                  </vpls.L2AccessInterfaceMldSnpgCfg>
               </children-Set>
            </vpls.L2AccessInterface>
         </children-Set>
      </vpls.Site>
   </configInfo>
</generic.GenericObject.configureInstanceWithResultResponse>
```

The following section provides XML sample requests to perform the following.

- Configure an existing port.
- Create an Epipe service.
- Delete an Epipe service.

See the XML API SDK Sample Code Navigator for sample XML scripts.

To compose a configuration command that modifies, creates, or deletes an existing object, you need the following information:

- the appropriate generic configuration method
- the class of the object you want to configure (not required if deleting)
- the full name of the object you want to configure
- the deployment type (typically immediate). See 15.3 “Deployments” (p. 209) for more information about deployments.

15.2.2 Modify example: To configure an existing port

The following example outlines the parameters required for an XML request to set the description on a port.

To compose a modification command, the key input parameters include:

- method—generic.GenericObject.configureInstanceWithResult
- class—equipment.PhysicalPort

- full name—network:198.51.100.40:shelf-1:cardSlot-1:card:daughterCardSlot-1:daughterCard:port-10

Figure 15-3 Port configuration request example

```
<SOAP:Body>
   <generic.GenericObject.configureInstanceWithResult xmlns="xmlapi_1.0">
      <deployer>immediate</deployer>
      <distinguishedName>network:198.51.100.40:shelf-1:cardSlot-1:card:daughterCardSlot-
1:daughterCard:port-10</distinguishedName>
      <includeChildren>false</includeChildren>
      <configInfo>
         <equipment.PhysicalPort>
            <actionMask>
               <bit>modify</bit>
            </actionMask>
            <description>to Mumbai</description>
         </equipment.PhysicalPort>
      </configInfo>
   </generic.GenericObject.configureInstanceWithResult>
</SOAP:Body>
```

For this sample request, other key input parameters include:

- includeChildren - set to false, means do not include child info in the results
- configInfo - desired values to which to configure object, and optionally children
  - actionMask - specifies the modify operation
  - description - specifies the attribute and value of the equipment.PhysicalPort class that is being modified

The results of the command are shown in the following figure. Results include all of the port attributes, but no children (as per includeChildren) in the configInfo parameter.

Figure 15-4 Port configuration response example

```
<generic.GenericObject.configureInstanceWithResultResponse xmlns="xmlapi_1.0">
   <configInfo>
      <equipment.PhysicalPort>
         <usedAsSyncReference>none</usedAsSyncReference>
         <cardSlotId>1</cardSlotId>
         <daughterCardSlotId>1</daughterCardSlotId>
         <specificCardType>
            <bit>mda_m60_faste_tx</bit>
         </specificCardType>
         <description>to Mumbai</description>
         .
         .
         .
         <selfAlarmed>true</selfAlarmed>
      </equipment.PhysicalPort>
```

```
      <children-Set>
   </configInfo>
</generic.GenericObject.configureInstanceWithResultResponse>
```

15.2.3 Create example: To create an Epipe object

The following example outlines what is required for an XML request to create an Epipe object. The example only shows object creation, and does not include all of the child objects required to create a working service.

To compose a creation command, the key input parameters include:

- method - generic.GenericObject.configureChildInstanceWithResult
- class - epipe.Epipe
- parent - svc-mgr. The fully distinguished name of the parent can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference.

Figure 15-5 Epipe creation request example

```
<SOAP:Body>
   <generic.GenericObject.configureChildInstanceWithResult xmlns="xmlapi_1.0">
      <deployer>immediate</deployer>
      <synchronousDeploy>true</synchronousDeploy>
      <distinguishedName>svc-mgr</distinguishedName>
      <childConfigInfo>
         <epipe.Epipe>
            <actionMask>
               <bit>create</bit>
            </actionMask>
            <displayedName>Epipe Example 1</displayedName>
            <description>Epipe Example 1</description>
            <subscriberPointer>subscriber:1</subscriberPointer>
         </epipe.Epipe>
      </childConfigInfo>
   </generic.GenericObject.configureChildInstanceWithResult>
</SOAP:Body>
```

For this request, other key input parameters include:

- distinguishedName - set to svc-mgr, this is the pointer to the parent object
- childConfigInfo - desired values to configure the children information, in this case the child object is epipe.Epipe
  - actionMask - set to create, specifies the creation operation
  - displayedName, description, and subscriberPointer - these are the attributes and associated values that are set on the epipe.Epipe object

The results of the command are shown in the following figure. Results include all of the default Epipe attributes including the displayedName, description, and subscriberPointer that were set in the request.

Figure 15-6 Epipe creation response example

```
<generic.GenericObject.configureChildInstanceWithResultResponse xmlns="xmlapi_1.
0">
   <childConfigInfo>
      <epipe.Epipe>
         <lastCacTime>0</lastCacTime>
         <cacStatus>notApplicab.e</cacStatus>
         <cacProbableCause>notApplicable</cacProbableCause>
         <defaultVcId>49</defaultVcId>
         <topologyAutoCompletion>false</topologyAutoCompletion>
         <transportPreference>any</transportPreference>
         <useBwReservedPath>noPreference</useBwReservedPath>
         .
         .
         .
         <objectFullName>svc-mgr:service-10907</objectFullName>
         <displayedName>Epipe Example 1</displayedName>
         <description>Epipe Example 1</description>
         <subscriberPointer>subscriber:1</subscriberPointer>
         <selfAlarmed>false</selfAlarmed>
         <children-Set/>
      </epipe.Epipe>
   </childConfigInfo>
</generic.GenericObject.configureChildInstanceWithResultResponse>
```

15.2.4 Delete example: To delete an Epipe object

CAUTION

Service Disruption

Deleting an object deletes all of its child objects, so deleting the wrong object could delete an entire tree of objects.

The following example outlines what is required for an XML request to delete an Epipe service.

To compose a deletion command, the key input parameters include:

- method - generic.GenericObject.deleteInstance
- full name - svc-mgr:service-10907, from the creation example in Figure 15-5, “Epipe creation request example” (p. 207)

Figure 15-7 Epipe deletion request example

```
<SOAP:Body>
   <generic.GenericObject.deleteInstance xmlns="xmlapi_1.0">
      <deployer>immediate</deployer>
      <synchronousDeploy>true</synchronousDeploy>
      <distinguishedName>svc-mgr:service-10907</distinguishedName>
```

```
   </generic.GenericObject.deleteInstance>
</SOAP:Body>
```

The results of the command is in the following figure. An exception-free response indicates that the request was received and validated.

Figure 15-8 Epipe deletion response example

```
<generic.GenericObject.deleteInstanceResponse xmlns="xmlapi_1.0"/>
```

15.3 Deployments

15.3.1 Overview

Deployments model the communication requests made from the management domain via the NFM-P GUI or OSS to the network. Deployments are created by the NFM-P at system startup and are present in a finite deployment pool to be used to:

- perform actions from the NFM-P to the managed network
- queue configuration requests from GUI and OSS clients

Deployment objects have data about the current state of deployment and can be used to track changed objects and parameters (elements) that need to be sent to the network. When the deployment performs the network change, all data is cleared from the deployment. If the deployment fails, it attempts to redeploy the request based on the deployment policy configuration.

OSS applications that send configuration requests to the network must monitor deployments. When changes are sent to the network, deployments are created, queued, and dispatched to the managed routers. When the deployment is successful, the NFM-P notifies the OSS that the deployment is completed for the request that triggered the deployment. If the deployment fails, an alarm creation or change event is generated and sent using JMS. It is important that OSS applications clear deployments after they are used.

See the following sections for more information:

- 15.3.4 “Deployment failures” (p. 211) in this section for more information about managing deployment failures
- 15.3.5 “Deployment failure recovery procedures” (p. 213) in this section for more information about how to recover from deployment failure, depending on the type of configuration request
- 15.3.6 “Deployment alarms and events” (p. 213) in this section for more information about the deployment alarm format and DeployerEvent
- Chapter 4, “Event monitoring using JMS” for more information about JMS events

15.3.2 Deployment types

Deployment related methods are found in the generic package and all configuration methods require a specified deployment type, as specified in the <deployer> tag. The deployment type is specified in the xmlApiTypes.xsd file. Deployments specify how and when deployment occurs. The valid values (with numeric equivalents) are:

- immediate (-1) to deploy to the network immediately
- followTransaction (-2) reserved for future use

- postponed (-3) reserved for future use
- ignored (-4) to not deploy to the network, so changes are only added to the database

15.3.3 Synchronous and asynchronous requests

You can configure the following deployment types:

- asynchronous (default)
- synchronous

Synchronous requests

Users can specify synchronous network requests using the <synchronousDeploy> and other tags. These requests are sent by the OSS to the NFM-P server where the request is validated and then saved to the NFM-P database. Subsequently, a deployment is created and queued on the NFM-P server for dispatching to the network. When the deployment has successfully performed the deployment across the network, or when all of the retries specified in the request are completed, the synchronous response is returned and the deployment put back in the pool of available deployments. There may be cases where a deployment causes some objects to be resynchronized from the NE. In these cases, the response is deferred until after the resynchronization has completed.

If there is a failure of all retries, an alarm is raised that contains details of the failed deployment and a failure XML response is sent back containing the failed deployment Id. See Figure 9-2, “Database interaction with synchronous network deployment” (p. 107) for a synchronous network deployment.

The following figure shows the setting of previously discussed deployment parameters when deleting an object. The parameters <deployRetries> and <deployRetryInterval> may be omitted to use default values.

Figure 15-9 Synchronous deployment request example

```
<SOAP:Body>
   <generic.GenericObject.deleteInstance xmlns="xmlapi_1.0">
      <deployer>immediate</deployer>
      <synchronousDeploy>true</synchronousDeploy>
      <clearOnDeployFailure>true</clearOnDeployFailure>
      <deployRetries>0</deployRetries>
      <deployRetryInterval>0</deployRetryInterval>
      <distinguishedName>network:10.1.202.93</distinguishedName>
   </generic.GenericObject.deleteInstance>
</SOAP:Body>
```

Asynchronous requests

Asynchronous requests are sent by the OSS to the NFM-P server where the request is validated and then saved to the NFM-P database. Subsequently, a response from the server is sent back to the OSS. A deployment is created and queued on the NFM-P server for dispatching to the network. When the deployment has successfully performed the deployment across the network, or when all of the retries specified in the request are completed, the deployment is put back in the pool of available deployments.

If there is a failure of all retries, an alarm is raised that contains details of the deployment. See 15.3.6 “Deployment alarms and events” (p. 213) in this section for more information. The default setting is that requests are made asynchronously across the network.

Note: If the OSS is not listening for alarms, it will not know if the deployments was successful, and may have to retrieve deployment information to verify. See 15.3.4 “Deployment failures” (p. 210) in this section for more information.

See Figure 9-3, “Database interaction with asynchronous network deployment” (p. 108) for an asynchronous network deployment.

The sample synchronous request shown in Figure 15-9, “Synchronous deployment request example” (p. 210) can be changed to an asynchronous request by changing <synchronousDeploy> to false:

```
<synchronousDeploy>false</synchronousDeploy>
```

15.3.4 Deployment failures

In the case of synchronous request failures, the XML API notifies the OSS of the deployment failure in the returned XML response. For asynchronous request failures, the OSS will not receive an XML response about the deployment failure. For both cases, the OSS must subsequently perform procedures for error-recovery.

If the OSS connection is lost or times out before the XML API returns a result to the OSS application, the OSS may not be aware of any subsequent success or failure notifications of the request. Steps must be taken at a later time, either manually or through the OSS, to verify whether the request was successful. If the request was unsuccessful, for example, because of a failed deployment, the OSS must perform procedures for error-recovery.

Error recovery methods

If there is a deployment failure, the failed deployment has more information on the failed state of the deployment for the OSS to retrieve and recover. The following table describes the methods from the generic.GenericObject class that are available to help with recovering from failed deployments.

Table 15-2 Generic methods for error recovery

Generic method	Description
generic.GenericObject.getDeployers	To find information about deployments that matches a specified filter with the children hierarchy. For example, you can get the deployment IDs for deployments with the failed deployment states set in the generic.GenericObject.deploymentState parameter. See $core.DeploymentState in the XML API Reference to see all the valid failure values.
generic.GenericObject.getDeployer	To find all parameter values for a specific deployment ID
generic.GenericObject.clearDeployer	To clear the deployment with the specified deployment ID
generic.GenericObject.triggerResync	To perform a non-scheduled resynchronization of an object
generic.GenericObject.getDeployersShallow	To find matched deployments similar to getDeployers without children hierarchy

Filtering can be applied when retrieving information about deployments using the following methods:

- generic.GenericObject.getDeployers
- generic.GenericObject.getDeployer
- generic.GenericObject.getDeployersShallow

The following figure shows an example of filtering on the generic.DeployerInfo class. The filter returns the name of each deployment that fails with an error code of 16, which represents an internal error.

Figure 15-10 Example of filtering based on generic.DeployerInfo

```
<filter>
   <and class="generic.DeployerInfo">
      <equal name="state" value="16" />
      <wildcard value="Default.DeployerBank:depl-%" name="objectFullName" />
   </and>
</filter>
```

The following figure shows a failed response exception message for a deployment request. When all retries fail, the failure response indicates the failed deployment ID. An alarm is not raised until all retries are attempted. See the XML API SDK Sample Code Navigator for a sample XML request to retrieve deployment information.

Figure 15-11 Failed deployment exception example

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <requestID>XML_API_client@n</requestID>
         <requestTime>11-Dec-2006 10:53:48 AM</requestTime>
         <responseTime>11-Dec-2006 10:53:49 AM</responseTime>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <generic.GenericObject.configureChildInstanceException xmlns="xmlapi_1.0">
         <description> [ app: generic ] [ class: generic.GenericObject ] [ in-
stance: N/A ] [ descr: Operation Failed During Deployment ]
         </description>
         <deployers>
            <deployer>Default.DeployerBank:depl-482</deployer>
            <deployer>Default.DeployerBank:depl-480</deployer>
         </deployers>
      </generic.GenericObject.configureChildInstanceException>
   </SOAP:Body>
</SOAP:Envelope>
```

Note: The <responseTime> tag in the header is the time at which the response stream is opened.

15.3.5 Deployment failure recovery procedures

The deployment failure recovery procedures to be carried out varies with the type of action (creation, modification, or deletion):

- If deployment fails during object creation, you must clear the deployments to remove the previously created object from the database. To remove the possibility of partially-created objects on an NE, resynchronize the object (this can be done using the generic.GenericObject.triggerResync method) and remove all partially-created objects using the XML API or the NFM-P.
- If deployment fails during object modification, clearing a deployment does not reset the modified object in the database. To reset to the previous object setting, resynchronize the object. In there are partially-modified objects, set the objects back to their original values using the XML API or the NFM-P.
- If deployment fails during object deletion, clearing a deployment completes the object deletion in the NFM-P database. To reset to the previous object setting, resynchronize the object.

See 15.4 “Workflow to handle deployment failures” (p. 216) in this chapter for more information about handling a deployment error that has caused the NFM-P database to be out of synchronization with the managed network.

15.3.6 Deployment alarms and events

OSS applications that rely on alarms and events to notify them of deployment failures must subscribe to the JMS interface. See Chapter 4, “Event monitoring using JMS” for information.

The DeployerEvent indicates the success or failure of an asynchronous deployment request and will return a successList and failedList of deployerIds. See Figure B-7, “DeployerEvent message example” (p. 343) for a DeployerEvent message example. An OSS can then take the failed deployerId and retrieve more information on the reason for the failure and then perform the required recovery procedure.

Deployment alarms are also raised and OSS applications can use the requestId and the requestUser to match failures with specific requests. After the deployerId is extracted from the alarm, you can use the deployerId to query a specific deployment. See the XML API SDK Sample Code Navigator for a sample request to retrieve deployment information.

Figure 15-12, “Deployment alarm example” (p. 214) shows a deployment alarm example. The following information is contained in the deployment alarm:

- the alarmClass attribute identifies the alarm class as a deployment failure
- the deployment ID is last component of the objectName attribute, in this case deployerId=771
- the additionalText attribute also contains the deployerId, the requestId, and the client user (requestUser) that made the request in the format:

  deployerId=771;requestId=AreqGenericObjectConfigureInstance-CLIENT-admin-NFM-P@198.51.100.164-15;requestUser=*user name*;deploymentType=3

Figure 15-12 Deployment alarm example

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <fm.FaultManager.findFaultsResponse xmlns="xmlapi_1.0">
         <result>
            <fm.AlarmInfo>
               <severity>minor</severity>
               <previousSeverity>indeterminate</previousSeverity>
               <originalSeverity>minor</originalSeverity>
               <highestSeverity>minor</highestSeverity>
               <probableCause>11</probableCause>
               <alarmName>13</alarmName>
               <type>5</type>
               <affectedObjectFullName>network:10.1.202.93:shelf-1:cardSlot-1:
card:daughterCardSlot-1:daughterCard:port-5</affectedObjectFullName>
               <affectedObjectClassName>equipment.PhysicalPort</affectedObjectClassName>

               <isAcknowledged>false</isAcknowledged>
               <wasAcknowledged>false</wasAcknowledged>
               <acknowldegedBy>N/A</acknowldegedBy>
               <firstTimeDetected>1128369209032</firstTimeDetected>
               <lastTimeDetected>1128369209032</lastTimeDetected>
               <lastTimeSeverityChanged>0</lastTimeSeverityChanged>
               <lastTimeCleared>0</lastTimeCleared>
               <lastTimePromoted>0</lastTimePromoted>
               <lastTimeDemoted>0</lastTimeDemoted>
               <lastTimeEscalated>0</lastTimeEscalated>
               <lastTimeDeEscalated>0</lastTimeDeEscalated>
               <lastTimeAcknowledged>0</lastTimeAcknowledged>
               <frequency>0</frequency>
               <occurences>N/A</occurences>
               <numberOfOccurences>2</numberOfOccurences>
               <numberOfOccurencesSinceClear>2</numberOfOccurencesSinceClear>
               <numberOfOccurencesSinceAck>0</numberOfOccurencesSinceAck>
               <isServiceAffecting>false</isServiceAffecting>
               <additionalText>deployerId=771;requestId=AreqGenericObjectConfigureInstance-
CLIENT-admin-NFM-P@13@198.51.100.164-15;requestUser=admin;deploymentType=3;</
additionalText>
               <operatorAssignedUrgency>indeterminate</operatorAssignedUrgency>
               <urgencyAssignedBy>1</urgencyAssignedBy>
               <relatedObjects>
                  <null/>
```

```
               </relatedObjects>
               <affectingObjects>
                  <relationshipTreeList>
                     <relationshipTree>
                        <objectFullName>Network Queue:default</objectFullName>
                        <subtree/>
                     </relationshipTree>
                  </relationshipTreeList>
               </affectingObjects>
               <nodeId>10.1.202.93</nodeId>
               <nodeName>sim202_93</nodeName>
               <affectedObjectDisplayedName>Port 1/1/5</
affectedObjectDisplayedName>
               <applicationDomain>equipment</applicationDomain>
               <displayedClass>PhysicalPort</displayedClass>
               <alarmClassTag>generic.DeploymentFailure</alarmClassTag>
               <affectedObjectClassIndex>87</affectedObjectClassIndex>
               <affectedObjectInstanceIndex>5</affectedObjectInstanceIndex>
               <deployerName>N/A</deployerName>
               <deployerId>0</deployerId>
               <requestId>N/A</requestId>
               <requestUser>N/A</requestUser>
               <objectFullName>faultManager:network@10.1.202.93@shelf-1@cardSlot-
1@card@daughterCardSlot-1@daughterCard@port-5|alarm-13-5-11-deployerId=771</
objectFullName>
               <objectClassName>fm.AlarmObject</objectClassName>
               <allomorphicClassName>fm.AlarmObject</allomorphicClassName>
               <objectId>-578593836</objectId>
               <displayedName>Port 1/1/5 - network@10.1.202.93@shelf-1@cardSlot-
1@card@daughterCardSlot-1@daughterCard@port-5|alarm-13-5-11-deployerId=771</
displayedName>
               <lifeCycleState>1</lifeCycleState>
               <deploymentState>0</deploymentState>
               <neId>10.1.202.93</neId>
            </fm.AlarmInfo>
         </result>
      </fm.FaultManager.findFaultsResponse>
   </SOAP:Body>
</SOAP:Envelope>
```

15.3.7 Deployment simulation

CAUTION

Service Disruption

It is recommended that deployment never be disabled in a production network.

Since configuration errors may not always be caught by the NFM-P server before configuration changes are committed to the database, you must disable deployment simulation and test with live network equipment to validate changes prior to OSS deployment in the production network.

By default, the NFM-P server deploys configuration changes to network elements through SNMP. You can configure the XML API to save configuration changes in the NFM-P database without deploying the changes, which facilitates OSS development in the absence of real equipment or simulators.

You can use the XML API to configure a mediation.DeploymentPolicy with a mediation.DeploymentMode property set to 1, to disable deployment. The default setting is 2, SNMP.

See Chapter 18, “Policy configuration management” for more information about how to configure policies.

15.4 Workflow to handle deployment failures

15.4.1 Stages

The following workflow outlines the high-level steps necessary to handle a deployment error that may cause the NFM-P database to be out of synchronization. See the XML API SDK Sample Code Navigator for sample XML scripts.

1

Clear the deployment. The XML configuration request uses the GenericObject method clearDeployer on the specified deployerName which identifies the deployment to be cleared.

2

Resynchronize the object that caused the deployment error. The XML configuration request to resynchronize the object using the generic.GenericObject.triggerResync method.

Note: When you use the generic.GenericObject.triggerResync method for a specific <instanceName>, you can specify whether a resynchronization on the object is performed by setting the <resyncSelf> flag to true, or whether the resync is performed on its children by setting the <resyncChildren> flag to true.

The NFM-P does not notify XML API clients when objects are resynchronized.

3

Perform one of the following steps, depending on your XML API client implementation.

a. Set the object back to the previous setting. To reset the previous object setting,

resynchronize the object by using the generic.GenericObject.triggerResync method described in Stage 2 .
See 15.3.5 “Deployment failure recovery procedures” (p. 213) in 15.3 “Deployments” (p. 209) for more information about the different request types that you must consider when you set an object back to the previous setting.
For example, if a request to create an object with children objects successfully creates the parent object, but creates only some of the children objects, you would create a request to delete the partially created objects.
For partial failures on a request to modify a port mode, the OSS client should create a request to change the port mode to return it to the original setting.
For failed deletion requests, performing Stage 2 resynchronizes the NFM-P with the network. Depending on the state the deleted object or objects in the NFM-P, the XML API client should create a request to re-create the deleted objects to restore the NFM-P and the network back to their original state before the failed deployment.
Subsequent requests to set objects back to their previous settings may fail. After an unsuccessful attempt to set objects back to their previous setting, Nokia recommends that the OSS client not retry.
The NFM-P continues to try and deploy the change, even after the failure, according to the deployment policies configured for the router. If an automatic recovery mechanism is implemented by the OSS application, the OSS application should use a recovery mechanism that is consistent with the configured deployment policies. For example, if the policy defines that deployments should redeploy after a failure, the OSS application should use a timer mechanism to wait until the deployment is retried before attempting another recovery method

b. Raise an alarm for the administrator to investigate the deployment failure.
In many cases, such as a loss of connectivity, deployments that fail will continue to fail if retried. If an OSS client continues to retry deployments after a failure, NFM-P resources will be consumed that will be ineffective, ultimately until all deployment resources are consumed. Nokia recommends that you do not continue to retry failed deployment requests. Instead, the OSS client can raise an alarm to alert the administrator that there is a deployment problem.

Note: Contact your Nokia OIPS technical support representative for information about how to manage deployment failures.

16 Device configuration management

16.1 Device configuration

16.1.1 Overview

You can use the XML API to create, configure, and manage the devices and children objects that are part of the NFM-P network. Equipment, such as routers, are at the top of the hierarchy and their properties and attributes are defined in the XML schema.

The XML API interface uses the following packages to manage equipment:

- equipment for physical equipment
- fr for frame relay equipment
- ethernetequipment for Ethernet equipment
- sonetequipment for SONET and SDH equipment
- sonettiming for SONET and SDH timing
- tdmequipment for TDM equipment
- genericne for GNE equipment

Most of the configuration actions that can be performed using the NFM-P GUI can also be performed using the XML API. Typical configuration actions include configuring cards, daughter cards, ports, channels, and the associated properties.

16.2 Workflow to configure equipment

16.2.1 Stages

The following workflow outlines the high-level steps necessary to configure equipment. Each of the steps provides guidance on the type of XML request required to perform a particular configuration action. See the XML API SDK Sample Code Navigator for sample XML scripts.

1

Configure each card slot by sending XML create requests to set the card type on the card on the router.

2

Configure each daughter card slot by sending XML create requests to set the daughter card type on the daughter card slot on the router.

3

Configure all ports by sending XML modify requests to turn up the port, with options to set encap type and mode.

4

As required, configure the channels. Prior to configuring SONET channels, card and ports have to exist. Send XML create and modify requests to create channel and set the parameters on the channels.

5

As required, configure LAG and MC-LAG by sending XML create and modify requests to create LAG interfaces and modify the parameters.

Note: Nokia recommends using the generic package for configuring all objects in the above mentioned packages. See 15.2 “Configuration methods” (p. 202) for more information about generic methods. See 13.2 “Network object model” (p. 161) for more information about the NFM-P equipment object hierarchies.

16.3 GNE profiles

16.3.1 Overview

The NFM-P provides limited management support of GNEs, which are typically devices from other vendors. These devices can be discovered and managed by the NFM-P, however configuration of the physical components of the NE is not supported. See “Generic NEs” in the *NSP NFM-P Classic Management User Guide* for more information about GNE support.

Before a GNE can be discovered by the NFM-P, a GNE profile has to exist. An OSS can be used to create these profiles. The genericne package is used for the management of GNE profiles and interfaces. The profiles define parameters used to manage non-native network elements.

A GNE profile includes the following elements:

- the device MIB system object ID
- regular-expression strings that specify the format of prompts and commands
- the SNMP trap management configuration
- interfaces that can be specified as the endpoints of NFM-P physical links

Note: To manage GNE profiles, the OSS user requires the genericne scope of command role.

16.4 GNE profile parameter configuration

16.4.1 Overview

The following configuration parameters outline some of the parameters from the genericne.GenericNEProfile class required to develop an SML request that can be used to create a GNE profile. It does not include the parameters for configuring trap management and interface types.

- product name - used to specify the NE Type (mandatory on create)
- description (optional)
- system object ID (mandatory on create). The system object ID can be one of the following types:

  - system ID of the network element (specifies the SNMP OID for the specific product and is unique)
  - partial system ID (to facilitate the discovery of different network element types for a family of products, for example, Cisco product the general system object ID 1.3.6.1.4.1.9.1.*)
- If the devices discovered by the profile are using the NFM-P script management feature, you can configure the following parameters for the CLI profile that are used for read and write CLI access:
  - commandPrompt
  - DisablePagingCommand
  - errorIndicator
  - resetCommand
  - twoStepsLogin
- the following CLI parameters can be configured as valid CLI commands for read and write access privileges:
  - readLoginPrompt
  - readPasswordPrompt
  - writeAccessLoginCommand
  - enablingWriteAccessLoginPrompt
  - writeLoginPrompt
  - writePasswordPrompt

See the XML API SDK Sample Code Navigator for sample XML scripts.

Note: You cannot modify the GNE type after you create the GNE profile. You must delete the GNE profile and create a new profile with the GNE type. You cannot delete the GNE profile if any NEs have been discovered using the profile.

16.5 Device software upgrades

16.5.1 Overview

The NFM-P supports software upgrades of managed devices via the OSS interface. Software images must be extracted relative to the *installation_directory*/server/nms/nodeSoftware directory on the NFM-P server. Files in this directory are automatically deleted by the NFM-P after 1440 m, or one day.

Contact your Nokia OIPS technical-support representative for information about configuring this time period, if required.

17 Network configuration management

17.1 Network configuration management

17.1.1 Overview

The XML API can be used to configure the routing and forwarding parameters on NEs. It uses the netw package to configure the contained network elements managed by the network manager.

The following network object configurations are required to establish the routing infrastructure and connectivity before services can be provisioned on the network:

- network interface
- static route
- routing protocol
- MPLS, LSP and service tunnel

17.2 Network interface configuration

17.2.1 Overview

The network interface represents a virtual router interface in the system. This virtual router interface is defined on a network port. While the physical connection of one device to another is through a port or channel linked by physical wire, it is the network interface that determines the IP connectivity.

The XML API uses the rtr package to configure router details. The rtr.NetworkInterface class is created and configured under the parent where the FDN is network:${systemAddress}:router-${*routingInstanceId}:ip-interface-${*id}. It can also include the following parameters:

- an IP address and subnet mask associated to a physical port or channel
- associated L3 interface to a physical port or channel cabled to another device
- configured QoS policy. See Chapter 18, “Policy configuration management” for more information.
- configured routing protocols. See 17.5 “Routing protocol configuration” (p. 224) for more information.

See the XML API Reference for more information about all of the parameters that can be configured on the classes from rtr package.

17.3 Workflow to configure network interfaces

17.3.1 Overview

The following workflow lists the high-level steps required to configure network interfaces to perform routing and forwarding. Each of the steps provide guidance on the type of XML request required to perform a particular configuration action.

See the XML API SDK Sample Code Navigator for sample XML scripts.

17.3.2 Stages

1

Create and configure the network interfaces that are associated with network ports. The XML configuration request creates and configures the interfaces as follows:

a. Assign a name to the interface.

b. Associate IP address, a network port, and QoS policy settings to the interface.

2

As required, create static routes. See 17.4 “Static route configuration” (p. 223) for more information.

3

As required, configure a routing policy. See Chapter 18, “Policy configuration management” for more information.

4

Configure and enable the associated routing protocol. See 17.5 “Routing protocol configuration” (p. 224) for more information.

17.4 Static route configuration

17.4.1 Overview

A static route is a fixed route to a fixed destination through specific hops on NEs.

As OSS application can use the XML API interface to create and configure static routes using the rtr.StaticRoute class. See the XML API SDK Sample Code Navigator for sample XML scripts.

17.5 Routing protocol configuration

17.5.1 Overview

A routing protocol specifies how routers communicate with each other, dynamically disseminating information that enables them to select routes between NEs, the choice of the route being done by routing algorithms. The NFM-P supports a variety of standard routing protocols.

You can use the XML API to enable and configure network protocol communications and connections on NEs.

Supported routing protocols and their packages include:

- BGP (bgp)
- RIP (rip)
- OSPF (ospf)

- LDP (ldp)
- IS-IS (isis)
- L2TP (l2tp)
- RSVP (rsvp)

The XML API also supports the following multicast protocols:

- PIM (pim)
- IGMP (igmp)
- MSDP (msdp)
- MLD (mld)

17.5.2 BGP

BGP is an inter-AS routing protocol. An AS is a network or a group of devices logically organized and controlled by a common network administration. BGP enables devices to exchange network reachability information. AS paths are the routes to each destination. There are two types of BGP: IBGP and EBGP.

- IBGP is used to communicate with peer devices in the same AS.
- EBGP is used to communicate with peers in different ASs.

17.5.3 RIP

RIP is an IGP that uses a distance-vector algorithm to determine the best route to a destination, using hop count as the deciding factor. In order for the protocol to provide complete information about routing, every device in the domain must participate in the protocol. RIP, a UDP-based protocol, updates its neighbors, and the neighbors update their neighbors. RIP directly advertises reachability information to its neighbors by sending prefix, mask, and either hop count or cost metric data.

17.5.4 OSPF

OSPF is a hierarchical link state protocol. OSPF is an IGP used within large ASs. OSPF routers exchange the state, cost, and other relevant interface information with neighbors after the neighbors are discovered. The information exchange enables all participating routers to establish a network topology map. Each router applies the Dijkstra algorithm to calculate the shortest path to each destination in the network. The resulting OSPF forwarding table is submitted to the routing table manager to calculate the routing table.

17.5.5 LDP

LDP is used to distribute labels in non-traffic-engineered MPLS applications. Routers can establish LSPs across a network by mapping network-layer routing information directly to the data link layer switched paths. After LDP distributes the labels to the LSR, the LSR assigns the label to a FEC, and then informs all other LSRs in the path about the label and how the label will switch data accordingly.

17.5.6 IS-IS

IS-IS is a link-state IGP that uses the shortest path first algorithm to determine a route. Routing decisions are made using the link-state information. IS-IS entities include:

- networks, which are autonomous system routing domains
- intermediate systems (supported routers)
- end systems, which are network devices that send and receive PDUs

End systems and intermediate system protocols allow NEs to identify each other. The IS-IS protocol sends link-state updates periodically through the network, so each NE can maintain current network topology information.

17.5.7 L2TP

L2PT is a session-layer protocol that extends the PPP model by allowing L2 and PPP endpoints to reside on different devices that are interconnected by a PSN. L2TP extends the PPP sessions between the CPE and PPP/L2TP termination point (LNS), via an intermediate LAC.

17.5.8 RSVP

RSVP is a network-control protocol in the IP suite that is used for communicating application QoS requirements to intermediate transit NEs in a network. RSVP uses a soft-state mechanism to maintain path and reservation states on each NE in the reservation path.

17.5.9 PIM

PIM is a multicast routing architecture that allows the addition of IP multicast routing on existing IP networks. PIM is unicast routing protocol independent and can be operated in the following ways:

- sparse mode
- dense mode

17.5.10 IGMP

The XML API uses the igmp package to configure the multicast IGMP routing type on IPv4 hosts and routers. IPv4 hosts and routers use IGMP to report their group memberships to neighboring multicast routers.

17.5.11 MSDP

MSDP is a protocol that enables multiple PIM-SM domains to communicate with each other using their own RPs. MSDP also enables multiple RPs in a single PIM-SM domain to establish MSDP mesh-groups and to synchronize information between anycast RPs about the active sources being served by each anycast RP peer.

17.5.12 MLD

MLD is an asymmetric protocol used by IPv6 routers to discover the presence of NEs that wish to receive multicast packets on their directly-attached links, and to discover which multicast addresses are of interest to those neighboring NEs.

Note: See “NE routing and forwarding” in the *NSP NFM-P Classic Management User Guide* for more information about routing protocols.

17.6 Workflow to configure a routing protocol

17.6.1 Overview

The following workflow lists the high-level steps required to configure a BGP, RIP, OSPF, LDP, or ISIS routing instance, or a PIM or IGMP multicast instance. Each routing protocol has different configuration steps that need to be followed to get routing operational in the network.

Each step in this workflow represents an XML configuration request. See the XML API SDK Sample Code Navigator for sample XML scripts.

17.6.2 Stages

1

For a BGP routing instance:

1. Configure an AS number.
2. Configure a number for the confederation-autonomous system if your configuration requires confederations.
3. Enable BGP on a router.
4. Configure global-level BGP elements.
5. Configure a BGP confederation.
6. Configure a peer group-level BGP.
7. Configure a peer-level BGP.

2

For a RIP routing instance:

1. Enable RIP on a router.
2. Configure a global-level RIP.
3. Configure a group-level RIP.

3

For an OSPF routing instance:

1. Enable OSPF on a router.
2. Configure OSPF elements.
3. Configure an OSPF area.
4. Add an L3 interface to the OSPF area.

4

For an LDP routing instance:

1. Enable LDP on a router.
2. Configure global-level LDP elements.
3. Configure LDP interfaces.
4. Configure LDP targeted peers.

5

For an ISIS routing instance:

1. Enable ISIS on a router.
2. Configure router-wide ISIS elements.
3. Configure ISIS NET addresses.
4. Configure ISIS interfaces.

6

For PIM routing instance:

1. Enable PIM.
2. Configure global-level PIM.
3. Configure a PIM source-specific multicast group.
4. Configure a PIM static rendezvous point.
5. Configure a PIM interface.
6. Configure the PIM candidate rendezvous point.
7. Configure the PIM anycast rendezvous point.

7

For IGMP routing instance:

1. Enable IGMP.
2. Configure IGMP.
3. Configure an IGMP interface.

17.7 VRRP virtual router configuration

17.7.1 Overview

VRRP allows the creation of a redundant routing system that takes over packet transmissions on a common LAN segment when a router fails. VRRP designates alternative routing paths in the form of a virtual router, without changing the IP address or MAC address of a protected router. The XML API supports the creation of virtual routers on IES services and core network LAN services.

With VRRP, a protected router owns the IP address of the virtual router. The owner router normally forwards packets to hosts on the default gateway. If the owner router fails, the virtual router, which has the same IP address as the owner, shifts packet-forwarding responsibilities to a designated backup router. The backup router then becomes the primary router and forwards packets on the virtual router IP address.

17.8 Workflow to configure a virtual router

17.8.1 Overview

The following workflow lists the high-level steps required to configure a virtual router on a core network LAN or an IES. Each step represents an XML configuration request. See the XML API SDK Sample Code Navigator for sample XML scripts.

This workflow assumes the following:

- an L3 interface primary IP address is available for the network or IES service.
- each IP interface must have a primary IP address.

17.8.2 Stages

1

Create a global VRRP priority-control policy for non-owner VRRP instances.

2

Distribute the global VRRP priority-control policy to a non-owner VRRP router.

3

Create a virtual router for a core network LAN, or an IES service.

4

As required, create an owner VRRP instance in the virtual router.

5

As required, create a non-owner VRRP instance in the virtual router.

17.9 MPLS, LSP, and service tunnel configuration

17.9.1 Overview

After your routing protocols have been configured, the next steps are to establish the network's data forwarding transport infrastructure that will eventually be used to carry services across the entire network. MPLS paths, LSPS, and service tunnels need to be created and configured to achieve this.

The XML API interface uses the mpls package for the configuration and provisioning of MPLS paths and LSPs on managed routers and it uses the svt package to configure service tunnels. See the XML API Reference for more information about these packages.

17.9.2 MPLS

MPLS is a data-carrying mechanism that uses one or more routing protocols to forward packets. MPLS can be used as the underlying transport mechanism for service tunnels. For MPLS to be used as such, an MPLS mesh and an LSP mesh must be created before the tunnel is created.

17.9.3 LSP

An LSP is a path through an MPLS network that is set up based on criteria in a forwarding equivalency class, or FEC. LSPs are unidirectional; they enable the label switching of a packet through an MPLS network from one endpoint to another. Bidirectional communication through an MPLS network requires the configuration of an LSP in the opposite direction.

17.9.4 Service tunnels

A service tunnel is an entity used to uni-directionally direct traffic from one device to another device. The service tunnel is provisioned to use a specific encapsulation method, such as GRE or MPLS, and the services are then mapped to the service tunnel. The most common type of tunnel used in the NFM-P is a service distribution point binding. Service tunnels originate on an SDP on a source NE and terminate at a destination NE.

Note: See “MPLS” and “Service tunnels” in the *NSP NFM-P Classic Management User Guide* for more information about MPLS, LSPs, and service tunnels.

17.10 Workflow to configure an MPLS path, LSP, and service tunnel

17.10.1 Overview

The following workflow lists the high-level steps required to configure an MPLS path, LSP, and service tunnel. Each step represents an XML configuration request. See the XML API SDK Sample Code Navigator for sample XML scripts.

This workflow assumes that at least one IGP is enabled and configured on all participating NEs and includes the system interface. See 17.5 “Routing protocol configuration” (p. 224) for more information.

17.10.2 Stages

1

Enable MPLS routing on all applicable routers and L3 interfaces.

2

Assign a network interface to an MPLS instance.

3

Create a full mesh of MPLS path which is the route to be used by an LSP, between the routers using the L3 interfaces.

4

Create a full mesh of LSPs and bind to the MPLS paths.

5

Create service tunnels between all routers that have access ports. The service tunnels direct traffic from one router to another. You must create service tunnels in both directions because service tunnels are unidirectional.

a. As required, create a GRE service tunnel.

b. As required, create an MPLS LSP service tunnel.

18 Policy configuration management

18.1 Policy configuration

18.1.1 Overview

The XML API supports the template-based creation of rules called policies. The supported policies include:

- service management—specify how service traffic is handled by network resources such as interfaces, ports, daughter cards, and circuits. For example, access ingress, and scheduler polices.
- routing management—specify routing configuration according to configured parameters. For example, routing and multicast polices.
- network management—specify how the NFM-P communicates with network resources, handles alarms, manages statistics, and stores information. For example, file, accounting policies.

Policies are global or local. A global policy is created using the NFM-P to set properties of the policy before being assigned to the NEs. After the global policy is ready to be distributed to the NEs, a local policy is created for each selected NE and the configuration is distributed to the NEs. Local policies are instances of global policies that are assigned to individual NEs. The global policy is an NFM-P object, and the local policy represents what is configured on the NE.

This chapter only describes the type of policies that are distributed to the NE. The type of policies that are for managing NFM-P only objects such as size constraint, format and range, alarm, or mediation policies are outside the scope of this chapter.

OSS applications can use the XML API to create, configure, and manage policies based on the XML schema. The user privileges associated with the OSS user account define the ability to configure NFM-P objects. See the chapter on NFM-P user security in the *NSP System Administrator Guide* for information about assigning permissions to NFM-P user accounts and user groups.

18.2 References

18.2.1 Overview

The following references are relevant to an OSS developer that is working in the policy configuration domain:

- *NSP NFM-P Classic Management User Guide*— documents the description, workflow, and configuration information about policy management
- XML API Reference—documents specific classes, attributes, and methods to configure policies using the XML API interface. See 18.3 “Key packages and classes” (p. 234) for the packages and classes that are relevant to a policy management OSS developer.
- Chapter 8, “XML API Reference” —documents information about how to navigate the XML API Reference

- Chapter 15, “Configuration management overview” —documents configuration methods, general creation, modification, and deletion of NFM-P-managed objects, and describes deployments
- XML API SDK Sample Code Navigator—documents samples of policy objects

18.3 Key packages and classes

18.3.1 Overview

The XML API supports the creation and modification of policies using the classes described in this section. The policy package contains classes to define and manage policies. Each policy is an instance of the policy.PolicyDefinition class. Policy items such as a queue of an access ingress policy, or an entry of an ACL filter are instances of the policy.PolicyItemDefinition. The specific policy is a child object of the policy.Manager. The types of policies in this section inherit from classes in this package.

The global policy represents a policy at the network level and the local policy represents a policy at the network element level. They are both children of the policy.Manager for the specific policy type.

Network level policy managers are named with the policyTypeManaged; for example, Access Egress. Network element level policy managers are named with network:siteId:policyTypeManaged; for example, network:10.1.1.88:Access Egress.

Note: policyTypeManaged is the displayed name of policy.PolicyType type.

Figure 18-1 policy.Manager class

The information in the policy.Manager class and policy.PolicyType provides an OSS developer with an introduction to classes, object hierarchy, and properties of policies to be configured. Other packages that may be relevant for a policy configuration OSS developer are the qos and acl packages. The packages define enumerations associated with properties in QoS-related policies such as access ingress, and access control lists such as ACL IP filters. The qos and acl packages do not have associated classes or methods.

The following tables list some of the policies and packages.

Table 18-1 Service management type policies

Policy	Package
QoS	
Access Egress	aengr
Access Ingress	aingr
Network	niegr
Network Queue	nqueue

Table 18-1 Service management type policies (continued)

Policy	Package
Slope	slope
HSMDA Slope	
Shared Queue	squeue
Scheduler	vs
Port Scheduler	portscheduler
HSMDA Port Scheduler	
Policer Control	policing
Queue Group	qgroup
QoS Profile	qosprofile
7705 Fabric Profile	fabricqos
OmniSwitch QoS	aosqos
7210 SAS QoS policies	sasqos
9500 QoS	mpr
Filter	
ACL IP Filer	aclfilter
ACL MAC Filter	
ACL IPv6 Filter	
DHCP Filter	

Table 18-2 Routing management type policies

Policy	Package
Multicast	
Egress Multicast Groups	multicast
Multicast Package	
Multicast CAC	
Ingress Multicast Bandwidth	
Ingress Multicast Info	
Ingress Multicast Reporting Destination	
7210 Multipoint Bandwidth Management	sasqos
VRRP	
VRRP	vrrp
Routing	

Table 18-2 Routing management type policies (continued)

Policy	Package
AS Path	rp
Community	
Damping	
Prefix List	
Statement	
MPLS	
MPLS Admin Group	mpls
Shared Risk Link Group	
LSP Template MVPN	
ISA	
IPSec Static Security Association	ipsec
IPSec Transform	
IPSec Tunnel Template	
IKE	
NAT	nat

Table 18-3 Network management type policies

Policy	Package
Statistics	
Accounting	multicast
File	file
RMON	
RMON	rmon
Time of Day	
Time Range	tod
Time of Day Suite	todsuite
Security	

Table 18-3 Network management type policies (continued)

Policy	Package
CPM Filter	sitesec
DoS Protection	
User Profile	
Password Policy	
RADIUS Authentication	
TACACS+ Authentication	

Note: The terms distinguishedName, instanceName, and objectFullName are used synonymously in the XML API for the unique identifier of the object instance. See 10.5.4 “objectFullName element” (p. 135) in 10.1 “Schema Reference ” (p. 127) for more information about the object full name.

To understand the notation used for creating the distinguishedName, or how to fill in a pointer property, see the Help link in the XML API Reference. For example, if you see the notation for a distinguishedName is Access Egress:${*id}:forwarding-class-${*forwardingClass}*, it means that to form the distinguishedName, the part in italics is the property value. For this distinguishedName, it could be Access:Egress:100:forwarding-class-be, where the 100 is the ${*id}* and the be is the ${*forwardingClass}*. The value for a pointer property is the object full name.

18.4 General policy configuration

18.4.1 Overview

CAUTION

Service Disruption

The code samples in this section show the structures for object creation or modification, and do not include all of the properties required to create a working policy.

See the XML API SDK Sample Code Navigator for working samples.

18.4.2 Policy configuration and examples

The XML API supports the creation and modification of policies. A unique numeric ID or unique name (a string) is used to uniquely identify a policy. Some policies have a default policy, which is automatically created by the NFM-P and in general, cannot be deleted or modified. There are exceptions to this. The ID is usually 1 or name of default.

The following example shows the minimum XML parameter tags required to develop an XML request to create the basic policy.

The genericObject.configureChildInstance method is used to create a policy. The request response uses the <objectFullName> element to identify the new policy.

Policy creation parameters

- method—generic.GenericObject.configureChildInstance
- distinguishedName—see Table 18-4, “Policy objects and associated distinguishedName” (p. 238) . The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference
- childConfigInfo—creates and configures the following policy parameters:
  - Policy object - see Table 18-4, “Policy objects and associated distinguishedName” (p. 238)
  - actionMask - specifies the create operation
  - one of the following:
    - <id> - assigns a unique numeric ID
    - <displayedName> - assigns a unique name

The following table lists the most commonly deployed policies. See policy.PolicyType in the XML API Reference for a complete list.

Table 18-4 Policy objects and associated distinguishedName

Policy	Policy object	distinguishedName
Access Egress	aengr.Policy	Access Egress
Access Ingress	aingr.Policy	Access Ingress
Network	niegr.Policy	Network
Network Queue	nqueue.Policy	Network Queue
Slope	slope.Policy	Slope
Shared Queue	squeue.Policy	Shared Queue
Scheduler	vs.Policy	Scheduler
ACL IP Filter	aclfilter.IpFilter	IP Filter
ACL MAC Filter	aclfilter.MacFilter	MAC Filter
ACL IPv6 Filter	acfilter.Ipv6Filter	IPv6 Filter
Routing: AS Path Community Damping Statement Prefix List	 rp.ASPath rp.Community rp.Damping rp.PolicyStatement rp.PrefixList	 rpASPath rpCommunity rpDamping rpStatement rpPrefixList
Accounting	accounting.Policy	Accounting
File	file.Policy	File

Access Egress

Access egress policies define egress service queues, map forwarding class flows to queues, are applied to access egress interfaces, and specify the QoS on egress.

Access egress policy creation parameters

- method—generic.GenericObject.configureChildInstance
- distinguishedName—Access Egress. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference.
- childConfigInfo—creates and configures the following policy parameters:
  - Policy object - aengr.Policy
  - actionMask - specifies the create operation
  - <id> - (optional) assigns a unique numeric ID. If the ID is not specified, the NFM-P automatically assigns the ID.
  - <children-Set>—tag to enclose children objects
    - Queue object - aengr.Queue
    - actionMask - specifies the create operation
    - <id> - (optional) assigns a unique queue ID. If the ID is not specified, the NFM-P automatically assigns the ID.
    - Forwarding class object - aengr.ForwardingClass
    - actionMask - specifies the create operation

Figure 18-2 Access egress policy creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>Access Egress</distinguishedName>
   <childConfigInfo>
      <aengr.Policy>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <id>99</id>
         <children-Set>
            <aengr.Queue>
               <actionMask>
                  <bit>create</bit>
               </actionMask>
               <id>2</id>
                ..
            </aengr.Queue>
            <aengr.ForwardingClass>
               <actionMask>
                  <bit>create</bit>
               </actionMask>
                .
            </aengr.ForwardingClass>
         </children-Set>
      </aengr.Policy>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

ACL IP Filter

Filter policies specify a forward, drop, or HTTP redirect action for packets based on information specified as the match criteria. ACL IP filter policies are applied to network and access IP interfaces and service tunnels.

ACL IP filter policy creation parameters

- method—generic.GenericObject.configureChildInstance
- distinguishedName—IP Filter. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference.
- childConfigInfo—creates and configures the following policy parameters:
  - Policy object - aclfilter.IpFilter
  - actionMask - specifies the create operation
  - <id> - (optional) assigns a unique numeric ID. If the ID is not specified, the NFM-P automatically assigns the ID.
  - <defaultAction> - selects a default action
  - <children-Set>—tag to enclose children objects
    - IP filter entry object - aclfilter.IpFilterEntry
    - actionMask - specifies the create operation
    - <id> - assigns a unique queue ID. If the ID is not specified, the NFM-P automatically assigns the ID.
    - <action> - select an action
    - set other parameters for IP filter entry

Figure 18-3 ACL IP filter policy creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>IP Filter</distinguishedName>
   <childConfigInfo>
      <aclfilter.IpFilter>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <id>1000</id>
         <defaultAction>forward</defaultAction>
         <children-Set>
            <aclfilter.IpFilterEntry>
               <actionMask>
                  <bit>create</bit>
               </actionMask>
               <id>20</id>
               <action>drop</action>
               <protocol>UDP</protocol>
               <destinationIpAddress>4.4.4.4</destinationIpAddress>
               <destinationIpAddressMask>32</destinationIpAddressMask>
               <destinationPort1>1</destinationPort1>
```

```
                <destinationPort2>6</destinationPort2>
                <destinationOperator>RANGE</destinationOperator>
             </aclfilter.IpFilterEntry>
          </children-Set>
       </aclfilter.IpFilter>
    </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

Routing policy statement

Routing policies, also known as route redistribution policies, control the size and content of the routing tables, the routes that are advertised, and the best route to take to reach a destination.

Routing policy statement creation parameters

- method—generic.GenericObject.configureChildInstance
- distinguishedName—rpStatement. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference.
- childConfigInfo—creates and configures the following policy parameters:
  - Policy object - rp.PolicyStatement
  - actionMask - specifies the create operation
  - <policyStatementName> - assigns a unique name
  - <children-Set>—tag to enclose children objects
    - Statement entry object - rp.PolicyStatementEntry
    - actionMask - specifies the create operation
    - <entryId> - assigns an entry ID.
    - <action> - select an action
    - <children-Set>—tag to enclose children objects
      - Statement action object - rp.Action
      - actionMask - specifies the create operation
      - <communityAction1> - select a community action
      - <communityName1> - assigns a community name
      - Statement from criteria object - rp.FromCriteria
      - actionMask - specifies the create operation
      - <communityName1> - assign community
      - <prefixList> - assigns a prefix list

Figure 18-4 Routing policy statement creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>rpStatement</distinguishedName>
   <childConfigInfo>
      <rp.PolicyStatement>
         <actionMask>
            <bit>create</bit>
         </actionMask>
```

```
         <policyStatementName>Test Sample</policyStatementName>
         <children-Set>
            <rp.PolicyStatementEntry>
               <actionMask>
                  <bit>create</bit>
               </actionMask>
               <entryId>5</entryId>
               <action>accept</action>
               <children-Set>
                  <rp.Action>
                     <actionMask>
                        <bit>create</bit>
                     </actionMask>
                     <communityAction1>replace</communityAction1>
                     <communityName1>c20</communityName1>
                  </rp.Action>
                  <rp.FromCriteria>
                     <actionMask>
                        <bit>create</bit>
                     </actionMask>
                     <communityName>c30</communityName>
                     <prefixList1>IP Address 32</prefixList1>
                  </rp.FromCriteria>
               </children-Set>
            </rp.PolicyStatementEntry>
         </children-Set>
      </rp.PolicyStatement>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

File and accounting policies

NEs collect accounting statistics using an accounting policy and as associated file policy that are assigned to a SAP, an SDP, a network port, or a subscriber profile.

A file policy specifies the relative size, storage location, and backup location of the files on the NE that contain accounting statistics data. An NE collects accounting statistics based on the collection interval specified for a SAP, network port, or subscriber in an accounting policy. The NE writes the statistics data in XML format to a file on the NE.

File policy creation parameters

- method—generic.GenericObject.configureChildInstance
- distinguishedName—File. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference.
- childConfigInfo—creates and configures the following policy parameters:
  - Policy object - file.Policy
  - actionMask - specifies the create operation
  - <id> - (optional) assigns a unique numeric ID. If the ID is not specified, the NFM-P automatically assigns the ID.

- <collectionInterval> - sets how long before the file rollover. After this time, the file closes and is compressed.
- <retentionInterval> - sets how long to retain the file
- <drive> - sets the location of the storage drive for the file

Figure 18-5 File policy creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>File</distinguishedName>
   <childConfigInfo>
      <file.Policy>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <id>88</id>
         <collectionInterval>600</collectionInterval>
         <retentionInterval>12</retentionInterval>
         <drive>cf1A</drive>
      </file.Policy>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

An accounting policy specifies an accounting statistics record type, a collection interval, an administrative state, and a file policy.

Accounting policy creation parameters

- method—generic.GenericObject.configureChildInstance
- distinguishedName—Accounting. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference.
- childConfigInfo—creates and configures the following policy parameters:
  - Policy object - accounting.Policy
  - actionMask - specifies the create operation
  - <id> - (optional) assigns a unique numeric ID. If the ID is not specified, the NFM-P automatically assigns the ID.
  - <fileObjectPointer> - assigns a file policy. See the Parent Hierarchy of the class file.Policy for the FDN.
  - <recordType> - selects a predefined type of accounting record to be written to the accounting file
  - <collectionInterval> - sets the statistics collection interval

Figure 18-6 Accounting policy creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>Accounting</distinguishedName>
   <childConfigInfo>
      <accounting.Policy>
```

```
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <id>89</id>
         <fileObjectPointer>File:88</fileObjectPointer>
         <recordType>svcEgressPkt</recordType>
         <collectionInterval>5</collectionInterval>
      </accounting.Policy>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

18.4.3 Policy distribution

After the policy is created in the NFM-P , the policy must be distributed before the configuration is received by the NEs. The following table lists the methods in the policy package to distribute policy configurations to the NEs.

Note: Distribution performed from the GUI or via the OSSI method policy.PolicyDefinition.distribute does not generate the PolicyDistributeEvent. Only distribution performed via the OSSI method policy.PolicyDefinition.distributeV2 will generate the PolicyDistributeEvent.

Table 18-5 Policy distribution methods

Method	Description
policy.PolicyDefinition.distribute	The global policy is distributed to the NEs specified in the siteIds. If the distribution fails the NFM-P validation for one NE, the global policy is not distributed to any of the NEs.
policy.PolicyDefinition.distributeV2	The global policy is distributed to the NEs specified in the siteIds. If the distribution fails the NFM-P validation for one NE, distribution continues to the other NEs. A JMS message is sent back to indicate the failed site. The JMS PolicyDistributeStatusEvent will indicate whether the distribution failed or succeeded for each NE. A failure is indicated if an NE is not reachable, an NE is not present, or an NE is in the Local Edit Only mode.

See the XML API Reference for information about policy distribution methods and the required input parameters. See “Policy distribution” in the *NSP NFM-P Classic Management User Guide* for information about policy distribution.

The following example shows the minimum XML parameter tags required to develop an XML request to distribute a policy.

`Policy distribution parameters`

- method—policy.PolicyDefinition.distribute
- instanceFullName—Access Egress: ${*id}. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference
- siteIds—the site ID list for distribution. An empty list means that the global policy is distributed to all applicable NEs.
  - <strings> - site ID—add one for each site

Figure 18-7 Policy distribution request example

```
<policy.PolicyDefinition.distribute xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <instanceFullName>Access Egress:99</instanceFullName>
   <siteIds>
      <string>198.51.100.55</string>
   </siteIds>
</policy.PolicyDefinition.distribute>
```

18.4.4 Policy configuration mode

The configuration mode parameter specifies the mode in which the global policy is configured, and determines whether the policy can be distributed to the NEs. The following table describes the policy configuration modes.

Table 18-6 Configuration mode

Option	Description
Draft	The policy is not reviewed or approved for distribution to NEs. The policy cannot be distributed (see information below for the exception to this).
Released	The policy is reviewed and approved for distribution to NEs. The policy can be distributed.

The isPolicyConfigurationModeUsed parameter in the server configuration determines the configuration mode in which a policy is created using the XML API. By default, the parameter is set to false, as shown in the following:

```
<policyOssiConfig isPolicyConfigurationModeUsed="false" />
```

When the parameter is set to false, the following occurs:

- A global policy that is created using the XML API is set to the Released configuration mode. To distribute the policy, the policy.PolicyDefinition.distribute method may be used. See 18.4.3 “Policy distribution” (p. 245) in this section for more information.
- Any modification to the global policy in Released configuration mode, the changes are automatically distributed to local policies belonging to the global policy according to the following:
  - If the local policy is in SyncWithGlobal distribution mode, the applicable configuration on the global policy is set on the local policy and distributed to the corresponding NE.
  - If the local policy is in LocalEditOnly distribution mode, changes to the global policy do not affect the local policy. Therefore, changes are not distributed to the corresponding NE.

When the parameter is set to false and you attempt to distribute a global policy that is in Draft configuration mode, the policy is distributed to local policies in SyncWithGlobal distribution mode and to the corresponding NEs.

When the parameter is set to true, the following occurs:

- A global policy that is created using the XML API is set to the Draft configuration mode. The policy must be set to the Released mode before the policy can be distributed. To distribute the policy, the policy.PolicyDefinition.distribute method may be used. See 18.4.3 “Policy distribution”

(p. 245) in this section for more information. The policy.PolicyDefinition.setConfigurationModeToReleased method may be used to release a draft policy.

- Any modification to the global policy in Released configuration mode, the XML API returns the policy to Draft mode. Before the policy is distributed, the policy must be set to the Released mode.
- The following rules apply:
  - If the local policy is in SyncWithGlobal distribution mode, the applicable configuration on the global policy is set on the local policy and distributed to the corresponding NE.
  - If the local policy is in LocalEditOnly distribution mode, changes to the global policy do not affect the local policy. Therefore, no changes are distributed to the corresponding NE.

To modify the isPolicyConfigurationModeUsed parameter, contact Nokia technical support.

18.4.5 Policy distribution mode

The distribution mode parameter specifies whether the local policy is synchronized with the associated global policy. Local policies can be different from each other as well as may differ from the global policy. The following table describes the policy distribution modes

Table 18-7 Distribution mode

Option	Option description
Sync With Global	The local policy is synchronized with the global policy. The local instance cannot be modified. [1] When you distribute a global policy, local policies in this distribution mode allow the NEs to receive the policy.
Local Edit Only	You can modify the local instance only, which affects the associated network element. Local policies in this distribution mode do not allow the NE to receive the distribution of a global policy. Changes to the global policy do not affect the local policy unless a synchronization operation is manually performed.

Notes:

1. WARNING - this is applicable if the operation is attempted using the NFM-P GUI. Using the XML API or CLI on the router, modifications on the local policy is possible and may cause the local and global policies to be out of synchronization even though the distribution mode on the local policy remains in SyncWithGlobal mode. Synchronization methods may be performed to synchronize the policies.

18.4.6 Policy modification workflows

The following workflow describes modification for global policy.

1. Make the proper changes.
2. Set the policy to Released mode if it is in Draft mode. See 18.4.4 “Policy configuration mode” (p. 246) in this section for information.
3. Distribute the policy. Local policies in SyncWithGlobal mode and the corresponding NEs receive the changes.

The following workflow describes modification for local policy.

1. If you want to set properties on a specific local policy that is to be different from the global policy, the policy must be in LocalEditOnly distribution mode.
2. Make the proper changes.

18.4.7 Policy modification examples

Global Policy

The following example shows the minimum XML parameter tags required to modify an access egress global policy. This method can be used for other types of policies. The find method can be used to retrieve the FDN of the object. See Chapter 13, “Inventory management” for information about how to use the find method.

`Global policy modification parameters`

- method—generic.GenericObject.configureChildInstance
- distinguishedName—Access Egress:${*id}. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference
- configInfo—creates and configures the following parameters:
  - Policy object - aengr.Policy
  - actionMask - specifies the modify operation
  - Properties that are modifiable
  - <children-Set> - tag to enclose children objects
    - Queue object - aengr.Queue
    - actionMask - specifies the modify operation
    - <objectFullName> - Access Egress:${*id}:queue-${*id}
    - Properties that are modifiable
    - Forwarding class object - aengr.ForwardingClass
    - <objectFullName> - Access Egress:${*id}:forwarding-class-${*forwardingClass}*
    - Properties that are modifiable

Figure 18-8 Global access egress policy modification request example

```
<generic.GenericObject.configureInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>Access Egress:99</distinguishedName>
   <configInfo>
      <aengr.Policy>
         <actionMask>
            <bit>modify</bit>
         </actionMask>
           Properties to modify
            ..
         <children-Set>
            <aengr.Queue>
               <actionMask>
                  <bit>modify</bit>
```

```
               </actionMask>
               <objectFullName>Access Egress:99:queue-2</objectFullName>
                 Properties to modify
                 ..
            </aengr.Queue>
            <aengr.ForwardingClass>
               <actionMask>
                  <bit>modify</bit>
               </actionMask>
               <objectFullName>Access Egress:99:forwarding-class-be</
objectFullName>
                 Properties to modify
                 ..
            </aengr.ForwardingClass>
         </children-Set>
      </aengr.Policy>
   </configInfo>
</generic.GenericObject.configureInstance>
```

Local Policy

It is recommended that the local policy be set to LocalEditOnly mode first before being modified.

The following example shows the minimum XML parameter tags required to set the distribution mode of a policy.

`Policy distribution mode parameters`

- method—policy.PolicyDefinition.setDistributionModeToLocalEditOnly
- instanceFullName—Access Egress: ${*id}. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference.
- siteIds—the site ID list for setting distribution mode
  - <strings> - site ID - add one for each site

Figure 18-9 Policy distribution mode request example

```
<policy.PolicyDefinition.setDistributionModeToLocalEditOnly xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <instanceFullName>Access Egress:99</instanceFullName>
   <siteIds>
      <string>198.51.100.55</string>
   </siteIds>
</policy.PolicyDefinition.setDistributionModeToLocalEditOnly>
```

The following example shows the minimum XML parameter tags required to modify an access egress local policy. This method can be used for other types of policies.

`Local policy modification parameters`

- method—generic.GenericObject.configureChildInstance

- distinguishedName—network:${*systemAddress}*:Access Egress:${*id}*. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference
- configInfo—creates and configures the following parameters:
  - Policy object - aengr.Policy
  - actionMask - specifies the modify operation
  - Properties that are modifiable
  - <children-Set> - tag to enclose children objects
    - Queue object - aengr.Queue
    - actionMask - specifies the modify operation
    - <objectFullName> - network:${*systemAddress}*:Access Egress:${*id}*:queue-${*id}*
    - Properties that are modifiable
    - Forwarding class object - aengr.ForwardingClass
    - <objectFullName> - network:${*systemAddress}*:Access Egress:${*id}*:forwarding-class-${*forwardingClass}*
    - Properties that are modifiable

Figure 18-10 Local access egress policy modification request example

```
<generic.GenericObject.configureInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>network:198.51.100.55:Access Egress:99</distinguishedName>
   <configInfo>
      <aengr.Policy>
         <actionMask>
            <bit>modify</bit>
         </actionMask>
           Properties to modify
            ..
         <children-Set>
            <aengr.Queue>
               <actionMask>
                  <bit>modify</bit>
               </actionMask>
               <objectFullName>network:198.51.100.55:Access Egress:99:queue-2</objectFullNam

                 Properties to modify
                 ..
            </aengr.Queue>
            <aengr.ForwardingClass>
               <actionMask>
                  <bit>modify</bit>
               </actionMask>
               <objectFullName>network:198.51.100.55:Access Egress:99:forwarding-class-
be</objectFullName>
                 Properties to modify
```

```
                ..
            </aengr.ForwardingClass>
         </children-Set>
      </aengr.Policy>
   </configInfo>
</generic.GenericObject.configureInstance>
```

18.5 Policy configuration workflow

18.5.1 Introduction

The following workflow describes the creation of a policy using the NFM-P. The workflow does not include the discovery on a NE or configuration using CLI. See “Policy management” in the *NSP NFM-P Classic Management User Guide* for more information about policy management.

18.5.2 Stages

1

Create the required policy using the appropriate policy class. Policies created using the XML API are global and are in Released configuration mode. See 18.4.4 “Policy configuration mode” (p. 246) in this section for information about configuration mode. When the global policy is in the Released configuration mode, a backup copy of the latest released version of the policy is saved. This can be used later to restore the released version of the global policy if the operator decides to revert back to the setting in the previous released version of the global policy.

2

Distribute the policy to selected NEs. After the policy is distributed successfully, a local policy is created for each NE. Local policies are instances of global policies that are assigned to individual NEs and contain properties applicable only to that NE. Local policies have a distribution mode, which is set to SyncWithGlobal when the policy is created using the workflow.

18.6 Policy methods

18.6.1 Overview

The XML API uses the policy package to define methods applicable for policies. Methods for policy distribution, setting modes, and synchronizing policies are provided by the policy.PolicyDefinition class. Methods for auditing policies are provided by the policy.Manager class. The most commonly used policy methods are described in the following table. See the XML API Reference for input parameters.

Table 18-8 Policy methods

Method	Description
policy.PolicyDefinition.setConfigurationModeToReleased	Set Configuration Mode Released and distribute the global policy to the local definitions network-wide. It is only applicable for the global policy. See Figure 18-11, “Configuration mode set request example” (p. 252) for an example of the method.
policy.PolicyDefinition.setConfigurationModeToDraft	Set Configuration Mode to Draft. It is only applicable for the global policy.
policy.PolicyDefinition.setDistributionModeToSyncWithGlobal	Set Distribution Mode to Sync with Global for local policies on the sites specified by siteIds. It automatically synchronizes local policies with the most recent released global policy. If it is called from the local policy, only that local policy is updated. See Figure 18-12, “Distribution mode set request example” (p. 253) for an example of the method.
policy.PolicyDefinition.setDistributionModeToLocalEditOnly	Set Distribution Mode to local edit only for local policies on the sites specified by siteIds. If it is called from the local policy, only that local policy is updated. See Figure 18-9, “Policy distribution mode request example” (p. 249) for an example of the method.
policy.PolicyDefinition.resetToReleasedPolicy	Reset the global policy to the most recent released policy. The local policies aren’t affected. See Figure 18-13, “Global policy reset request example” (p. 253) for an example of the method.
policy.PolicyDefinition.syncTo	If it is called from global policy, the selected local policy is compied into this global policy. If Configuration Mode isn’t used, the synchronized global policy is distributed to existing local definitions network-wide. If it is called from the local policy, the selected local policy is copied into this local policy, and applied to the network.

The following are the mandatory XML parameter tags to set the configuration mode of a global policy:

- method—policy.PolicyDefinition.setConfigurationModeToReleased
- instanceFullName—Access Egress: ${*id}*. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference.

The following figure shows a request to set the configuration mode of a global policy.

Figure 18-11 Configuration mode set request example

```
<policy.PolicyDefinition.setConfigurationModeToReleased xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <instanceFullName>Access Egress:99</instanceFullName>
</policy.PolicyDefinition.setConfigurationModeToReleased>
```

To set the mode to Draft, you must replace setConfigurationModeToReleased with setConfigurationModeToDraft.

The following are the mandatory XML parameter tags to set the distribution mode of a policy:

- method—policy.PolicyDefinition.setDistributionModeToSyncWithGlobal
- instanceFullName—Access Egress: ${*id}* for a global policy and

network:${*systemAddress*}:Access Egress:${*id*} for a local policy. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference.

- siteIds—site ID list for setting distribution mode; required only when setting the distribution mode from the global instance level
  - <strings> - site ID - add one for each site

The following figure shows a request to set the distribution mode of a policy.

Figure 18-12 Distribution mode set request example

```
<policy.PolicyDefinition.setDistributionModeToSyncWithGlobal xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <instanceFullName>Access Egress:99</instanceFullName>
   <siteIds>
      <string>198.51.100.55</string>
   </siteIds>
</policy.PolicyDefinition.setDistributionModeToSyncWithGlobal>
```

The following example shows the minimum XML parameter tags required to reset a global policy to the most recent released.

Policy reset to released parameters

- method—policy.PolicyDefinition.resetToReleasedPolicy
- instanceFullName—Access Egress: ${*id*}. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference.

The following figure shows a request to reset a global policy to the most recent released.

Figure 18-13 Global policy reset request example

```
<policy.PolicyDefinition.resetToReleasedPolicy xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <instanceFullName>Access Egress:99</instanceFullName>
</policy.PolicyDefinition.resetToReleasedPolicy>
```

19 Service configuration management

19.1 Service configuration

19.1.1 Overview

After the network devices, protocols, IP/MPLS components, and policies have been configured and established, network providers need to provision services to be used by end users. See Chapter 16, “Device configuration management” and Chapter 17, “Network configuration management” for more information about configuring devices and network objects.

A service is a means of transport for the application content required by end users. A service is owned by a service customer typically called the service provider. See 19.7 “Customer configuration” (p. 260) in 19.4 “General service configuration” (p. 258) for more information about how to use the XML API to create and configure the customers that require services for the end users and residential subscribers. See 19.25 “Residential subscriber configuration” (p. 288) for more information about residential subscriber configuration.

Services provide Internet or VPN connectivity. VPN services can provide Layer 2 bridged service, or Layer 3 IP routing connectivity between a SAP on one router and another SAP on the same router (for local service) or another router (for distributed service). Each service in the network is uniquely identified by a service ID.

A distributed service on a supported NE uses SDPs to direct service traffic between service sites. Therefore, a service must be bound to an SDP to send traffic between sites. See 17.9.4 “Service tunnels” (p. 230) in 17.9 “MPLS, LSP, and service tunnel configuration” (p. 229) for more information about configuring SDPs.

The NFM-P supports the following service types:

- VLL—a type of VPN where IP is transported point-to-point for the following:
  - Epipe—a Layer 2 point-to-point VLL service for Ethernet frames
  - Apipe—a point-to-point ATM service between users connected to routers on an IP/MPLS network
  - Fpipe—a point-to-point frame relay service between users connected to routers on an IP/MPLS network
  - Ipipe—provides IP connectivity between a host attached to a point-to-point access circuit with routed IPv4 encapsulation and a host attached to an Ethernet interface
  - Cpipe—used to connect the far-end circuit for circuits encapsulated in MPLS
  - Hpipe—a point-to-point HDLC service over an IP/MPLS network
- VPLS—a Layer 2 multipoint-to-multipoint VPN where a number of sites are connected in a bridged domain over an IP/MPLS network
- IES—a direct Internet access service where the customer is assigned an IP interface for Internet connectivity
- VPRN—a Layer 3 IP multipoint-to-multipoint VPN service as defined in RFC 2547 bis
- VLAN—a logical grouping of two or more nodes that are not necessarily on the same physical network segment, but share the same IP network address

- Mirror—a type of service that copies the packets from a customer service to a destination outside the service for troubleshooting or surveillance purposes
- Composite—a set of linked services
- Optical Transport—a wavelength that traverses the network between two endpoints

The OSS applications can use the XML API to create, configure, and manage services based on the XML schema. The privileges associated with an OSS user account define which objects the user can configure. See the chapter on NFM-P user security in the *NSP System Administrator Guide* for information about assigning permissions to NFM-P user accounts and groups.

19.2 References

19.2.1 Overview

The following references are relevant to OSS developers who work in the service configuration domain:

- *NSP NFM-P Classic Management User Guide*— documents the description, workflow, and configuration information about service management
- XML API Reference—documents classes, class hierarchy, attributes, and methods to configure services using the XML API interface. See 19.3 “Key packages and classes” (p. 256) for the packages and classes that are relevant to an service management OSS developer.
- Chapter 8, “XML API Reference” —documents information about how to navigate the XML API Reference
- Chapter 15, “Configuration management overview” —documents configuration methods, creation, modification and deletion of NFM-P-managed objects, and describes deployments
- XML API SDK Sample Code Navigator—documents samples of service objects

19.3 Key packages and classes

19.3.1 Overview

The XML API interface uses the service.serviceManager class, which is the parent class for most of the service types, to manage services. See Figure 19-1, “service.serviceManager (svc-mgr) class” (p. 257) . When you create, modify, or delete a service, the distinguishedName and objectFullName start with svc-mgr if applicable. The FDN of the service.servicManager class is svc-mgr, and there is only one instance of this class from which the service objects inherit.

Note: The terms distinguishedName, instanceName, and objectFullName are used synonymously in the XML API to refer to the unique identifier of the object instance. See 10.5.4 “objectFullName element” (p. 135) in 10.1 “Schema Reference ” (p. 127) for more information about the object full name.

Figure 19-1 service.serviceManager (svc-mgr) class

- **Parent Hierarchy**
  - service.**ServiceManager** [svc-mgr]
- **Children Hierarchy**
  - service.**ServiceManager**
    - apipe.Apipe
      - apipe.Site

The packages for the service components are:

- Customer—subscr
- Service—vll, apipe, epipe, fpipe, hpipe, ies, ipipe, vpls, mvpls, vprn, ies, vlan, mpr, optical
- SDP Binding—svt

The XML API interface uses the service.CompositeServiceManager class to manage composite services. When you create, modify, or delete objects in a composite service, the distinguishedName and objectFullName start with comp-svc-mgr, if applicable. The FDN of the servic.CompositeServiceManager is comp-svc-mgr, and there is only one instance of this class from which the composite service objects inherit.

Figure 19-2 service.CompositeServiceManager (comp-svc-mgr) class

- **Parent Hierarchy**
  - service.**CompositeServiceManager** [comp-svc-mgr]
- **Children Hierarchy**
  - service.**CompositeServiceManager**
    - service.CompositeService
    - service.CrossConnect
    - service.RoutedVplsConnector
    - service.ScpConnector
    - service.SpokeConnector

To understand the notation used for creating the distinguishedName, or how to fill in a pointer property, see the Help link in the XML API Reference. For example, when a distinguishedName is svc-mgr:service-${*id}:${siteId}, italicized items after the dollar sign and enclosed in parenthesis are the property value. The distinguishedName may be svc-mgr:service-10000:10.10.20.22, where the 1000 is the ${*id} and the 10.10.20.22 is the ${siteId}. The value for a pointer property is the object full name.

The XML API Reference specifies whether the properties need to be set during the creation of the object. The following figure shows that the property must be set when the object is created because access for the property siteId is read-create.

Figure 19-3 Sample siteID property settings

siteId	*type*=InetAddress *access*=read-create *default*=0.0.0.0 *Displayed(tab/group)*=System ID (/Network Element)

19.4 General service configuration

19.4.1 Overview

CAUTION

Service Disruption

The code samples in this section display the structures for object creation or modification.

However, the samples do not include all of the properties required to create a working service.

For working samples, see the XML API SDK Sample Code Navigator.

19.5 Service configuration workflow

19.5.1 Stages

The service configuration workflow describes how to create each of the service components.

1

Configure policies, as required. It is recommended that policies be created and configured before you create the service. Policies that are applied to be part of services include QoS policies such as Access Ingress, Access Egress, Scheduler, Filter, Accounting, and ANCP. See Chapter 18, “Policy configuration management” for more information.

2

Configure ports or channels as access ports with encapsulation types and an encapsulation identifier (if required, depending on encapsulation type). See Chapter 17, “Network configuration management” for more information.

3

Configure service tunnels, as required. See Chapter 17, “Network configuration management” for more information.

4

Create and configure the customer. See 19.7 “Customer configuration” (p. 260) in this section for more information.

5

Create a service and associate the customer with the service:

- Add site(s) to the service.
- Add access interface(s) to the site of the service.
- (Optional) Add SDP binding(s) if the service spans more than one site and the SDP binding is added to the site of the service.

19.6 CLI mapping

19.6.1 Overview

The following table contains an example of NFM-P service objects and CLI mapping to create a customer and an Epipe service.

Table 19-1 Sample service NFM-P object and CLI mapping

NFM-P Epipe model	CLI Epipe model (CLI commands)
Subscriber - subscr.Subscriber	customer
Epipe Service - epipe.Epipe	service
Epipe Site - epipe.Site	
Epipe Access Interface - vll.L2AccessInterface	sap
Spoke SDP Binding - svt.SpokeSdpBinding	spoke-sdp

`configure service customer <customer-id>`

- where <customer-id> in CLI is <subscriberId> under the subscr.Subscriber class in the XML API

`configure service <service-type> <service-id> customer <customer-id>`

- where <service-type> in CLI is the appropriate service class. For example, when the <service-type> is epipe, the class is epipe.Epipe to create the service.
- In the XML API, the service site must also be created. For example, when the <service-type> is epipe, the class for the service site is epipe.Site.

configure service <service-type> <service-id> customer <customer-id>

sap 1/1/4:23

- sap in CLI is the access interface class in the XML API. For example, when the access interface is for an epipe, the class is vll.L2AccessInterface to create the SAP.
- The 1/1/4:23 in CLI is configured with the following properties under the vll.L2AccessInterface class:
  - <portPointer>
  - <innerEncapValue>
  - <outerEncapValue>

configure service <service-type> <service-id> customer <customer-id>

spoke-sdp 165:5

- spoke-sdp in CLI is the svt.SpokeSdpBinding class in the XML API for creating the spoke SDP binding
- The 165:5 in CLI is configured with the following properties under the svt.SpokeSdpBinding class:
  - <pathPointer>
  - <vcId>

19.7 Customer configuration

19.7.1 Overview

Customers are the service providers that pay for services from a network provider. A customer can own more than one service, but a service is owned by only one customer. Two or more services can be joined to form a composite service. The services that comprise a composite service can be owned by different customers. Customers that own services in a composite service are associated with the composite service.

Each customer is associated with a customer ID, which is assigned when the customer account is created. The customer ID can be used to modify customer information. The NFM-P has a default customer already created with a customer ID value of 1. See the *NSP NFM-P Classic Management User Guide* for more information about customer management.

The XML API allows you to create and manage customers of services by using the subscr package.

Customer creation parameters

- method - generic.GenericObject.configureChildInstance
- distinguishedName - subscriber
- childConfigInfo - creates and configures the following customer parameters:
  - Customer object - subscr.Subscriber
  - actionMask - specifies the create operation
  - <subscriberId> - assigns a unique subscriber ID

Figure 19-4 Customer creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
```

```
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>subscriber</distinguishedName>
   <childConfigInfo>
      <subscr.Subscriber>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <subscriberId/>
          ..
      </subscr.Subscriber>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

The XML request can include other contact information such as subscriberName, address, and phone number. See the XML API Reference for more information about the subscr package.

The subscriber ID can be used to modify customer information. When you associate a customer with a service, enter subscriber:${*subscriberId}* to specify the subscriberPointer property of the service.

19.8 Service configuration

19.8.1 Overview

The NFM-P supports the following service types:

- VLL
- VPLS/HVPLS/MVPLS/PBB
- IES
- VPRN
- VLAN
- Mirror
- Optical Transport
- Composite

See “Service management” in the *NSP NFM-P Classic Management User Guide* for more information about service management.

The XML API allows you to create and manage services by using the classes specified in the following table.

Table 19-2 Service objects

Service type	Package and class
Apipe	apipe.Apipe, mpr.Apipe (for Wavence only)
Epipe	Epipe.Epipe, mpr.Epipe (for Wavence only)
Fpipe	fpipe.Fpipe
Ipipe	ipipe.Ipipe
Cpipe	cpipe.Cpipe, mpr.Cpipe (for Wavence only)
Hpipe	hpipe.Hpipe

Table 19-2 Service objects (continued)

Service type	Package and class
VPLS, MVPLS, PBB	vpls.Vpls, mvpls.Mvpls
IES	ies.Ies
VPRN	vprn.Vprn

The service.Service class is the parent class of the service objects in the previous table. The id and serviceId properties in the class are inherited by the subclasses that are used to identify the service. The following table describes the properties.

Table 19-3 Service class properties

Property	Description
id	SVC Mgr service ID—a unique value that identifies the service in the NFM-P database. The ID is also included as part of the FDN for the service objects. For example, 620 is the ID for svc-mgr:service-620.
serviceId	service ID—this ID is unique on the NE but may not be unique in the NFM-P database.

The following example shows the minimum XML parameter tags required to develop an XML request to create a basic service.

The genericObject.configureChildInstance method is used in the XML request to create a service. The XML response returns the <objectFullName> parameter that can be used to identify the new service.

Service creation parameters

- method - generic.GenericObject.configureChildInstance
- distinguishedName - svc-mgr
- childConfigInfo - creates and configures the following service parameters:
  - Service object - see Table 19-2, “Service objects” (p. 261)
  - actionMask - specifies the create operation
  - <subscriberPointer> - assigns a subscriber
  - <serviceId> - (optional) assigns a unique service ID. If an ID is not specified, the NFM-P automatically assigns the ID.

Figure 19-5 Service creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>svc-mgr</distinguishedName>
   <childConfigInfo>
      <epipe.Epipe>
         <actionMask>
            <bit>create</bit>
         </actionMask>
```

```
        <subscriberPointer>subscriber:1</subscriberPointer>
          ..
      </epipe.Epipe>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

See the XML API Reference and Schema Reference for more information about classes, properties, and methods to create the required services.

19.9 Site configuration

19.9.1 Overview

Service sites are the NEs that participate in a service.

The XML API allows you to create and manage service sites by using the classes specified in the following table.

Table 19-4 Service site objects

Service type	Package and class
Apipe	apipe.Site, mpr.Asite (for Wavence only)
Epipe	epipe.Site, mpr.Esite (for Wavence only)
Fpipe	fpipe.Site
Ipipe	ipipe.Site
Cpipe	cpipe.Site, mpr.Site (for Wavence only)
Hpipe	hpipe.Site
VPLS, MVPLS, PBB	vpls.Site, vpls.BSite, vpls.ISite, mvpls.Site, mvpls.BSite, mvpls.ISite
IES	ies.Site
VPRN	vprn.Site

The following example shows the minimum XML parameter tags required to develop an XML request to create a basic service site.

The genericObject.configureChildInstance method is used in the XML request to create a site. The XML response returns the <objectFullName> parameter that can be used to identify the new service site.

`Service site creation parameters`

- method - generic.GenericObject.configureChildInstance
- distinguishedName - svc-mgr:service-${*id}*
- childConfigInfo - creates and configures the following service site parameters:
  - Service object - see Table 19-4, “Service site objects” (p. 263)
  - actionMask - specifies the create operation
  - <siteId> - IP address of the NE

Figure 19-6 Service site creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>svc-mgr:service-620</distinguishedName>
   <childConfigInfo>
      <epipe.Site>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <siteId>198.51.100.198</siteId>
          ..
      </epipe.Site>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

See the XML API Reference and Schema Reference for more information about classes, properties, and methods to create the required service sites.

19.10 SAP configuration

19.10.1 Overview

Each customer service is configured with at least one SAP. The access interface identifies the point of customer interface for a service on the managed device.

The XML API allows you to create and manage service access interfaces by using the classes specified in the following table.

Table 19-5 SAP objects

Service type	Package and class
Apipe	vll.L2AccessInterface, mpr.AL2AccessInterface (for Wavence only)
Epipe	vll.L2AccessInterface, mpr.EL2AccessInterface (for Wavence only)
Fpipe	vll.L2AccessInterface
Ipipe	ipipe.L2AccessInterface
Cpipe	vll.L2AccessInterface, mpr.L2AccessInterface (for Wavence only)
Hpipe	vll.L2AccessInterface
VPLS, MVPLS, PBB	vpls.L2AccessInterface, vpls.BL2AccessInterface, vpls.IL2AccessInterface, mvpls.L2AccessInterface, mvpls.BL2AccessInterface, mvpls.IL2AccessInterface
IES	ies.L3AccessInterface
VPRN	vprn.L3AccessInterface

The following example shows the minimum XML parameter tags required to develop an XML request to create a basic service access interface.

The genericObject.configureChildInstance method is used in the XML request to create a SAP. The XML response returns the <objectFullName> parameter that can be used to identify the new SAP.

Service access interface creation parameters

- method - generic.GenericObject.configureChildInstance
- distinguishedName - svc-mgr:service-${*id}:${*siteId}*
- childConfigInfo - creates and configures the following SAP interface parameters:
  - Service object - see Table 19-5, “SAP objects” (p. 264)
  - actionMask - specifies the create operation
  - <portPointer> - assigns an access port
  - <innerEncapValue>/<outerEncapValue> - assigns encapsulation values
    - To configure an IP address for an IES or VPRN service, the rtr.VirtualRouterIpAddress class properties must be configured. This class is the child class of the ies.L3AccessInterface or vprn.L3AccessInterface and must be enclosed using the <children-Set> tags.
      - <ipAddress> - assigns an IP address

Figure 19-7 SAP creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>svc-mgr:service-620: 198.51.100.198</distinguishedName>
   <childConfigInfo>
      <vll.L2AccessInterface>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <portPointer>network:198.51.100.234:shelf-1:cardSlot-1:card:daughterCardSlot-
1:daughterCard:port-7</portPointer>
         <innerEncapValue>0</innerEncapValue>
         <outerEncapValue>3001</outerEncapValue>
          ..
      </vll.L2AccessInterface>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

See the XML API Reference and Schema Reference for more information about classes, properties, and methods to create the desired service access interfaces.

19.11 SDP binding configuration

19.11.1 Overview

SDP bindings are associations of SDPs to services. The SDPs are required only on distributed services. A distributed service spans more than one NE. SDPs are used to direct traffic from one NE to another through a service tunnel.

The XML API allows you to create and manage SDP bindings by using the classes specified in the following table.

Table 19-6 Service SDP binding objects

Service type	Package and class
Apipe	svt.SpokeSdpBinding
Epipe	
Fpipe	
Ipipe	
Cpipe	
Hpipe	
VPRN	
IES	
VPLS, MVPLS, PBB	svt.MeshSdpBinding, svt.SpokeSdpBinding

The following example shows the minimum XML parameter tags required to develop an XML request to create a basic service SDP bindings.

The genericObject.configureChildInstance method is used in the XML request to create a SDP binding. The XML response returns the <objectFullName> parameter to identify the new service SDP binding.

Service SDP binding creation parameters

- method - generic.GenericObject.configureChildInstance
- distinguishedName - svc-mgr:service-${*id}*:${*siteId}*
- childConfigInfo - creates and configures the following SDP binding parameters:
  - Service object - see Table 19-6, “Service SDP binding objects” (p. 266)
  - actionMask - specifies the create operation
  - <tunnelSelectionTerminationSiteId> - assigns the termination site ID of the SDP binding

Figure 19-8 SDP binding creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>svc-mgr:service-620: 198.51.100.198</distinguishedName>
   <childConfigInfo>
      <svt.SpokeSdpBinding>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <tunnelSelectionTerminationSiteId>198.51.100.82</tunnelSelectionTerminationSiteId>

          ..
```

```
      </svt.SpokeSdpBinding>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

See the XML API Reference and Schema Reference for more information about classes, properties, and methods to create the required SDP bindings.

19.12 Creating service with site, access interface, and SDP binding in one request

19.12.1 Overview

The following example shows an XML request to create a VPLS, site, access interface, and SDP binding in one request. This method can be used for other types of services.

Looking at the specific service object class in the XML API Reference, under the Children Hierarchy and the specific attribute, you can determine what child objects are configured for the service object. The following figure displays the children hierarchy of the vpls.Vpls object.

Figure 19-9 Children hierarchy of the vpls.Vpls object

- **Children Hierarchy**
  - vpls.Vpls
    - vpls.Site

and the properties is as follows:

vpls.Site-Set	*type*=Children-Set

The XML request to set the VPLS site inside the VPLS service must be enclosed in the <children-Set> tags. The XML request example does not include all of the properties required to create a working service.

See the XML API Reference for the service object class under the Children Hierarchy and the specific attribute, to determine the child objects that can be configured for this service object.

Service creation parameters

- method - generic.GenericObject.configureChildInstance
- distinguishedName - svc-mgr
- childConfigInfo - creates and configures the following service parameters:
  - Service object - vpls.Vpls
  - actionMask - specifies the create operation

- <subscriberPointer> - assigns a subscriber. See the Parent Hierarchy of the class subscr.Subscriber for the FDN.
- <serviceId> - assigns a unique service ID
- <children-Set> - tag to enclose children objects
  - Site object - vpls.Site
  - actionMask - specifies the create operation
  - <siteId> - IP address of the NE
  - <children-Set> - tag to enclose children objects
    - SDP binding object- svt.MeshSdpBinding
    - actionMask - specifies the create operation
    - <tunnelSelectionTerminationSiteId> - assigns site ID of tunnel termination for the mesh SDP binding
    - Access interface object- vpls.L2AccessInterface
    - actionMask - specifies the create operation
    - <portPointer>- assigns an access port
    - <innerEncapValue>/<outerEncapValue> - assigns encapsulation values

Figure 19-10 Service creation request example including site, SAP, and SDP binding

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>svc-mgr</distinguishedName>
   <childConfigInfo>
      <vpls.Vpls>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <serviceId/>
         <subscriberPointer/>
         <children-Set>
            <vpls.Site>
               <actionMask>
                  <bit>create</bit>
               </actionMask>
               <siteId/>
               <children-Set>
                  <svt.MeshSdpBinding>
                     <actionMask>
                        <bit>create</bit>
                     </actionMask>
                     <circuitType>mesh</circuitType>
                     <tunnelSelectionTerminationSiteId/>
                  </svt.MeshSdpBinding>
                  <vpls.L2AccessInterface>
                     <actionMask>
                        <bit>create</bit>
```

```
                    </actionMask>
                    <portPointer/>
                    <innerEncapValue/>
                    <outerEncapValue/>
                 </vpls.L2AccessInterface>
              </children-Set>
           </vpls.Site>
        </children-Set>
     </vpls.Vpls>
  </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

19.13 Modifying service configuration

19.13.1 Overview

The following are the requirements for an XML request to modify a VPLS, site, access interface, and SDP binding separately. This method can be used for other types of services. The find method can be used to retrieve the FDN of the object. See Chapter 13, “Inventory management” for more information about the find method.

Service modification parameters

- method - generic.GenericObject.configureInstance
- distinguishedName - determined by the Parent Hierarchy of the class to be created in the XML API Reference
  - for a service - svc-mgr:service-${*id}
  - for a site - svc-mgr:service-${*id}:${*siteId}
  - for an access interface - svc-mgr:service-${*id}:${*siteId}:interface-${*portPointer}-inner-tag-${*innerEncapValue}-outer-tag-${*outerEncapValue}
  - for an SDP binding- svc-mgr:service-${*id}:${*siteId}:circuit-${*pathId}-${*vcId}
- configInfo - creates and configures the following service parameters:
  - Service object:
    - For a service - vpls.Vpls
    - For a site - vpls.Site
    - For an access interface- vpls.L2AccessInterface
    - For an SDP binding- svt.MeshSdpBinding or svt.SpokeSdpBinding
  - actionMask - specifies the modify operation
  - properties that are modifiable

The following request modifies properties in a VPLS. For other service objects, replace the value in the <distinguishedName> and use the appropriate service object. For example, to modify a site, use the following:

```
<distinguishedName>svc-mgr:service-123:198.51.100.55</distinguishedName>
```

- where <vpls.Site> and </vpls.Site> are in the positions of the <vpls.Vpls> and </vpls.Vpls> respectively.

Figure 19-11 VPLS modification request example

```
<generic.GenericObject.configureInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>svc-mgr:service-123</distinguishedName>
   <configInfo>
      <vpls.Vpls>
         <actionMask>
            <bit>modify</bit>
         </actionMask>
          Properties to modify
          ..
      </vpls.Vpls>
   </configInfo>
</generic.GenericObject.configureInstance>
```

The following example outlines what is required for an XML request to modify a VPLS service, site, access interface, and SDP binding in one request. This method can be used for other types of services.

Service modification parameters

- method - generic.GenericObject.configureInstance
- distinguishedName - svc-mgr:service-${*id}. The FDN is determined by the Parent Hierarchy of the class to be created in the XML API Reference
- configInfo - creates and configures the following service parameters:
  - Service object - vpls.Vpls
  - actionMask - specifies the modify operation
  - Properties that are modifiable
  - <children-Set> - tag to enclose children objects
    - Site object - vpls.Site
    - actionMask - specifies the modify operation
    - <objectFullName>- svc-mgr:service-${*id}:${*siteId}
    - Properties that are modifiable
    - <children-Set> - tag to enclose children objects
      - SDP binding object- svt.MeshSdpBinding
      - <objectFullName>- svc-mgr:service-${*id}:${*siteId*}:circuit-${*pathId*}-${*vcId*}
      - Properties that are modifiable
      - Access interface - vpls.L2AccessInterface
      - <objectFullName>- svc-mgr:service-${*id}:${*siteId*}:interface-${*portPointer*}-inner-tag-${*innerEncapValue*}-outer-tag-${*outerEncapValue*}
        - Properties that are modifiable

Figure 19-12 Multiple service object modification request example

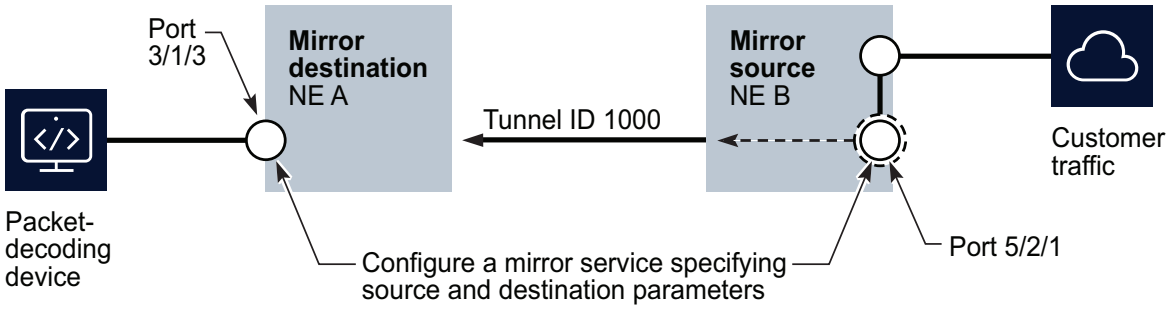
```
<generic.GenericObject.configureInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName> svc-mgr:service-620</distinguishedName>
   <configInfo>
      <vpls.Vpls>
         <actionMask>
            <bit>modify</bit>
         </actionMask>
          Properties to modify
          ..
         <children-Set>
            <vpls.Site>
               <actionMask>
                  <bit>modify</bit>
               </actionMask>
               <objectFullName>svc-mgr:service-620:198.51.100.198</object
FullName>
                Properties to modify
                ..
               <children-Set>
                  <svt.MeshSdpBinding>
                     <actionMask>
                        <bit>modify</bit>
                     </actionMask>
                     <objectFullName>svc-mgr:service-620:198.51.100.198:circuit-1-10</object

                    Properties to modify
                    ..
                  </svt.MeshSdpBinding>
                  <vpls.L2AccessInterface>
                     <actionMask>
                        <bit>modify</bit>
                     </actionMask>
                     <objectFullName>svc-mgr:service-620: 198.51.100.198:
interface-1/1/4-inner-tag-0-outer-tag-95</objectFullName>
                      Properties to modify
                      ..
                  </vpls.L2AccessInterface>
               </children-Set>
            </vpls.Site>
         </children-Set>
      </vpls.Vpls>
   </configInfo>
</generic.GenericObject.configureInstance>
```

19.14 Mirror service

19.14.1 Overview

The XML API implementation of service mirroring provides mirroring of service traffic packets from any service type. In a mirror service, packets from one or more sources are forwarded to their normal destinations. A copy of the entire packet or a specified portion of the packet, is sent to the mirror destination. The mirrored packet can be viewed using a packet-decoding device. The device is attached to the destination port. For a local mirror service, the source site and the destination site are on the same device. For a remote mirror service, the source site and the destination site are on different devices and the mirrored packets are transported unidirectionally through the core network using IP or MPLS tunneling.

Figure 19-13 Remote service mirroring



See the “Mirror services” chapter of the *NSP NFM-P Classic Management User Guide* for information about implementing a mirror service.

19.15 Mirror service package and classes

19.15.1 Overview

The XML API uses the mirror package to manage mirror services. Under this package, the following classes are used to configure properties for a mirror service:

- Mirror service- mirror.Mirror
- Mirror site- mirror.Site
- Mirror L2 access interface- mirror.L2AccessInterface
- To configure mirror source to match criteria such as IP addresses, MAC addresses and MPLS ingress labels to filter mirror traffic, use the following classes:
  - mirror.SourceIngressLabel
  - mirror.SourceInterface
  - mirror.SourceIpFilter
  - mirror.SourceMacFilter

- mirror.SourcePort
- mirror.SourceSubscriberHost

The mirror SDP binding is configured under the class svt.MirrorSdpBinding.

19.16 CLI mapping

19.16.1 Overview

The following table shows an example of NFM-P service objects and CLI mapping to create a mirror service.

Table 19-7 Mirror service NFM-P and CLI mapping

NFM-P Mirror model	CLI Mirror model (CLI commands)
Mirror Service- mirror.Mirror	configure mirror mirror-dest
Mirror Destination/Source Site - mirror.Site	
Mirror Destination L2 Access Interface - mirror.L2AccessInterface	configure context sap (on the mirror destination)
Mirror Destination Source Far End - mirror.RemoteSource	configure context remote-source (on the mirror destination)
Mirror Source SDP Binding - svt.MirrorSdpBinding	configure context spoke-sdp (on the mirror source)
Mirror Source filters: • mirror.SourceIngressLabel • mirror.SourceInterface • mirror.SourceIpFilter • mirror.SourceMacFilter • mirror.SourcePort • mirror.SourceSubscriberHost	debug mirror-source ingress-label sap ip-filter mac-filter port subscriber

19.17 Mirror service configuration

19.17.1 Workflow

See the “Mirror services” chapter of the *NSP NFM-P Classic Management User Guide* for information about the prerequisites and workflow to create a mirror service. The following describes the workflow to create a mirror service.

1

Create a mirror service.

2

Create a mirror service destination with a source far end and an access interface. This is the mirror destination on router A in Figure 19-13, “Remote service mirroring” (p. 272) .

3

Create a mirror service source site with a spoke SDP binding and a source port. This is the mirror source on router B in Figure 19-13, “Remote service mirroring” (p. 272) .

19.17.2 Examples

The following example shows XML requests to create a remote mirror service that contains a destination site with the far end address of the source and an access interface, and another request to create source site with a spoke SDP binding.

Mirror service creation parameters

- method - generic.GenericObject.configureChildInstance
- distinguishedName - svc-mgr
- childConfigInfo - creates and configures the following service parameters:
  - Service object - mirror.Mirror
  - actionMask - specifies the create operation
  - <subscriberPointer> - only subscriber 1 can be assigned
  - <serviceId> - (optional) assigns a unique service ID. If the service ID is not specified, the NFM-P automatically assigns the ID.

Figure 19-14 Simple mirror service creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>svc-mgr</distinguishedName>
   <childConfigInfo>
      <mirror.Mirror>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <subscriberPointer>subscriber:1</subscriberPointer>
          ..
      </mirror.Mirror>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

Mirror service destination site creation parameters

- method - generic.GenericObject.configureChildInstance
- distinguishedName - svc-mgr:service-${*id}*
- childConfigInfo - creates and configures the following service site parameters:
  - Service object - mirror.Site
  - actionMask - specifies the create operation

- <siteId> - IP address of the destination NE
- <mirrorSiteType> - for a destination site, select destination

  <children-Set> - tag to enclose child objects
  - Source far end object - mirror.RemoteSource
  - actionMask - specifies the create operation
  - <remoteSourceSiteId> - IP address of the source NE
  - Access interface object- mirror.L2AccessInterface
  - actionMask - specifies the create operation
  - <portPointer>- assigns a access port. See the Parent Hierarchy of the port or channel object such as equipment.PhysicalPort for the FDN.
  - <innerEncapValue>/<outerEncapValue> - assigns encapsulation values

Figure 19-15 Mirror service destination creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>svc-mgr:service-732</distinguishedName>
   <childConfigInfo>
      <mirror.Site>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <siteId/>
         <mirrorSiteType>destination</mirrorSiteType>
         <children-Set>
            <mirror.RemoteSource>
               <actionMask>
                  <bit>create</bit>
               </actionMask>
               <remoteSourceSiteId/>
            </mirror.RemoteSource>
            <mirror.L2AccessInterface>
               <actionMask>
                  <bit>create</bit>
               </actionMask>
               <portPointer/>
               <innerEncapValue/>
               <outerEncapValue/>
            </mirror.L2AccessInterface>
         </children-Set>
      </mirror.Site>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

Mirror service source site creation parameters

- method - generic.GenericObject.configureChildInstance

- distinguishedName - svc-mgr:service-$*{*id}*
- childConfigInfo - creates and configures the following service site parameters:
  - Service object - mirror.Site
  - actionMask - specifies the create operation
  - <siteId> - IP address of the source NE
  - <mirrorSiteType> - for a source site, select source
  - <children-Set> - tag to enclose children objects
    - SDP binding object - svt.MirrorSdpBinding
    - actionMask - specifies the create operation
    - <tunnelSelectionTerminationSiteId> - assigns the termination site ID of the SDP binding
    - Source port object - mirror.SourcePort
    - actionMask - specifies the create operation
    - <portName> - mirror source port name

Figure 19-16 Mirror service source creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>svc-mgr:service-732</distinguishedName>
   <childConfigInfo>
      <mirror.Site>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <siteId/>
         <mirrorSiteType>source</mirrorSiteType>
         <children-Set>
            <svt.MirrorSdpBinding>
               <actionMask>
                  <bit>create</bit>
               </actionMask>
               <tunnelSelectionTerminationSiteId/>
            </svt.MirrorSdpBinding>
            <mirror.SourcePort>
               <actionMask>
                  <bit>create</bit>
               </actionMask>
               <portName/>
            </mirror.SourcePort>
         </children-Set>
      </mirror.Site>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

The following example outlines what is required for an XML request to create a remote mirror service that contains a destination site with the far-end address of the source and an access interface, and a source site with a spoke SDP binding, in one request.

Mirror service creation parameters

- method - generic.GenericObject.configureChildInstance
- distinguishedName - svc-mgr. To determine the FDN, check the Parent Hierarchy of the class to be created in the XML API Reference
- childConfigInfo - creates and configures the following service parameters:
  - Service object - mirror.Mirror
  - actionMask - specifies the create operation
  - <subscriberPointer> - only subscriber 1 can be assigned. See the Parent Hierarchy of the class subscr.Subscriber for the FDN.
  - <serviceId> - (optional) assigns a unique service ID. If the service ID is not specified, the NFM-P automatically assigns the ID.
  - <children-Set> - tag to enclose children objects
    - Destination site object - mirror.Site
    - actionMask - specifies the create operation
    - <siteId> - IP address of the destination NE
    - <mirrorSiteType> - for a destination site, select destination
    - <children-Set> - tag to enclose children objects
      - Source far end object- mirror.RemoteSource
      - <remoteSourceSiteId> - IP address of the source NE
      - Access interface object - mirror.L2AccessInterface
      - <portPointer> - assigns an access port. See the Parent Hierarchy of the port or channel object such as equipment.PhysicalPort for the FDN.
      - <innerEncapValue>/<outerEncapValue> - assigns encapsulation values
    - Source site object - mirror.Site
    - actionMask - specifies the create operation
    - <siteId> - IP address of the source NE
    - <mirrorSiteType> - for a source site, select source
    - <children-Set> - tag to enclose children objects
      - SDP binding object - svt.MirrorSdpBinding
      - <tunnelSelectionTerminationSiteId> - assigns the termination site ID of the SDP binding
      - Source port object - mirror.SourcePort
      - <portName> - mirror source port name

The following figure shows an example of a request to create a mirror service, including the source, destination, and SDP binding

Figure 19-17 Complete mirror service creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>svc-mgr</distinguishedName>
   <childConfigInfo>
      <mirror.Mirror>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <subscriberPointer>subscriber:1</subscriberPointer>
         <children-Set>
            <mirror.Site>
               <actionMask>
                  <bit>create</bit>
               </actionMask>
               <siteId/>
               <mirrorSiteType>destination</mirrorSiteType>
               <children-Set>
                  <mirror.RemoteSource>
                     <actionMask>
                        <bit>create</bit>
                     </actionMask>
                     <remoteSourceSiteId/>
                  </mirror.RemoteSource>
                  <mirror.L2AccessInterface>
                     <actionMask>
                        <bit>create</bit>
                     </actionMask>
                     <portPointer/>
                     <innerEncapValue/>
                     <outerEncapValue/>
                  </mirror.L2AccessInterface>
               </children-Set>
            </mirror.Site>
            <mirror.Site>
               <actionMask>
                  <bit>create</bit>
               </actionMask>
               <siteId/>
               <mirrorSiteType>source</mirrorSiteType>
               <children-Set>
                  <svt.MirrorSdpBinding>
                     <actionMask>
                        <bit>create</bit>
                     </actionMask>
                     <tunnelSelectionTerminationSiteId/>
                  </svt.MirrorSdpBinding>
```

```
                  <mirror.SourcePort>
                     <actionMask>
                        <bit>create</bit>
                     </actionMask>
                     <portName/>
                  </mirror.SourcePort>
               </children-Set>
            </mirror.Site>
         </children-Set>
      </mirror.Mirror>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

19.18 VLAN service

19.18.1 Overview

VLAN services enhance the traditional segmentation services that are provided by routers in LAN configurations by addressing scalability, security, and network management issues.

See “Device VLAN support” in the *NSP NFM-P Classic Management User Guide* for more information about the types of VLAN services that are supported on the NFM-P by device type, and the prerequisites and workflow to create different types of VLAN services.

19.19 VLAN groups

19.19.1 Overview

VLAN groups are required to create a VLAN service. VLAN groups are used by OmniSwitch and Wavence devices to:

- logically group OmniSwitch and Wavence devices to represent a typical network topology.
- manage the VLAN IDs that are assigned to Wavence VLAN group members

VLAN group creation parameters

- method - generic.GenericObject.configureChildInstance
- distinguishedName - group:${*groupTypeManaged*}
- childConfigInfo - creates and configures the following parameters:
  - Group object - netw.VlanGroup
  - actionMask - specifies the create operation
  - <groupName> - assigns a unique name for the group

Figure 19-18 VLAN group creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>group:tree</distinguishedName>
   <childConfigInfo>
```

```
      <netw.VlanGroup>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <groupName/>
          ..
      </netw.VlanGroup>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

19.20 VLAN service configuration

19.20.1 Workflow

The following workflow describes how to create a VLAN service.

1

Create a VLAN service.

2

Choose VLAN type and VLAN group.

3

Create a VLAN site.

4

Add an access interface.

5

For an OmniSwitch, and depending on the type of VLAN, create Ethernet service, service access multipoint, SAP, and customer VLANs.

For separate requests, see 19.8 “Service configuration” (p. 261) in 19.4 “General service configuration” (p. 258) . Replace the service object with vlan.Vlan, replace the site object with vlan.Site, and replace the access interface object with vlan.L2AccessInterface. Use the proper distinguishedName for the object you are creating.

19.20.2 Examples

The following example outlines what is required for an XML request to create a VLAN service, site, and access interface in one request.

`VLAN service creation parameters`

- method - generic.GenericObject.configureChildInstance
- distinguishedName - svc-mgr. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference
- childConfigInfo - creates and configures the following service parameters:

- Service object - vlan.Vlan
- actionMask - specifies the create operation
- <subscriberPointer> - assigns a subscriber. See the Parent Hierarchy of the class subscr.Subscriber for the FDN.
- <displayedName> - assign a service name
- <serviceId> - (optional) assigns a unique service ID. If the service ID is not specified, the NFM-P automatically assigns the ID.
- <groupPointer> - assigns a VLAN group. See the Parent Hierarchy of the class netw.VlanGroup for the FDN.
- <vlanSubType> - choose a VLAN type
- <children-Set> - tag to enclose children objects
    - Site object - vlan.Site
    - actionMask- specifies the create operation
    - <siteId> - IP address of the NE
    - <children-Set> - tag to enclose children objects
        - Access interface object - vlan.L2AccessInterface
        - <portPointer> - assign a port. See the Parent Hierarchy of the port or channel object such as equipment.PhysicalPort for the FDN.
        - If applicable, depending on the VLAN type, configure an Ethernet service instead of an access interface: Ethernet service object - vlan.EthernetService.
        - actionMask - specifies the create operation
        - <ethernetServiceName> - assigns an Ethernet service name
        - <children-Set> - tag to enclose children objects
            - Service access multipoint object - vlan.ServiceAccessMultipoint
            - <children-Set> - tag to enclose children objects
                - Customer VLAN object - vlan.CustomerVlan
                - Service access point object - vlan.ServiceAccessPoint
                - <portPointer> - assigns a port. See the Parent Hierarchy of the port or channel object such as equipment.PhysicalPort for the FDN.

Figure 19-19 VLAN service creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>svc-mgr</distinguishedName>
   <childConfigInfo>
      <vlan.Vlan>
         <actionMask>
            <bit>create</bit>
         </actionMask>
```

```
         <vlanSubType>tlsVlan</vlanSubType>
         <serviceId/>
         <subscriberPointer/>
         <groupPointer>group:tree:name</groupPointer>
         <children-Set>
            <vlan.Site>
               <actionMask>
                  <bit>create</bit>
               </actionMask>
               <siteId/>
               <children-Set>
                  <vlan.EthernetService>
                     <actionMask>
                        <bit>create</bit>
                     </actionMask>
                     <ethernetServiceName/>
                     <children-Set>
                        <vlan.ServiceAccessMultiPoint>
                           <actionMask>
                              <bit>create</bit>
                           </actionMask>
                           <sapProfilePointer>Ethernet Service SAP Policy:default-sap-profil
ProfilePointer>
                           <children-Set>
                              <vlan.ServiceAccessPoint>
                                 <actionMask>
                                    <bit>create</bit>
                                 </actionMask>
                                 <portPointer/>
                              </vlan.ServiceAccessPoint>
                              <vlan.CustomerVlan>
                                 <actionMask>
                                    <bit>create</bit>
                                 </actionMask>
                              </vlan.CustomerVlan>
                           </children-Set>
                        </vlan.ServiceAccessMultiPoint>
                     </children-Set>
                  </vlan.EthernetService>
               </children-Set>
            </vlan.Site>
         </children-Set>
      </vlan.Vlan>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

19.21 Optical transport service

19.21.1 Overview

An optical transport service is a wavelength that traverses the network between two endpoints. The path the service takes through an NE is cross-connect (XC).

The NFM-P recognizes the adjoining XCs and managed link XCs as an optical transport service. The NFM-P supports optical transport services between two of the following NEs (7750 SR, 7705 SAR, 7450 ESS, 7210 SAS).

19.21.2 Workflow

The following describes the workflow to configure an optical transport service.

1. Configure cards and timeslots (if applicable on the card type).
2. Configure the rate on the optical NE.
3. Configure optical links to logically bind the ports; the links are used to create XCs.
4. Configure the services by selecting the endpoints on the sites of the service.

19.22 Optical transport service configuration

19.22.1 Workflow

The following workflow describes how to create an optical transport service.

1. Create an optical transport service.
2. Create the A end optical service site.
3. Create the termination point port.
4. Create the Z end optical service site.

5

Create the termination point port.

Up to two sites can be selected for the creation of an optical transport service. The NFM-P names the sites as follows: A end and a Z end site.

For separate requests, see 19.8 “Service configuration” (p. 261) in 19.4 “General service configuration” (p. 258) . Replace the service object with optical.TransportService, replace the site object with optical.ServiceSite, and replace the access interface object with optical.TerminationPoint. Use the proper distinguishedName for the object you are creating.

19.22.2 Examples

The following example outlines what is required for an XML request to create an optical transport service, site, and termination point port in one request.

`Optical transport service creation parameters`

- method - generic.GenericObject.configureChildInstance
- distinguishedName - svc-mgr. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference
- childConfigInfo - creates and configures the following service parameters:
  - Service object - optical.TransportService
  - actionMask - specifies the create operation
  - <subscriberPointer> - assigns a subscriber. See the Parent Hierarchy of the class subscr.Subscriber for the FDN.
  - <rate> - assigns a rate
  - <serviceId> - (optional) assigns a unique service ID. If the service ID is not specified, the NFM-P automatically assigns the ID.

    <children-Set> - tag to enclose children objects
    - Site object - optical.ServiceSite
    - actionMask - specifies the create operation
    - <siteId> - IP address of the NE
    - <sitePosition> - indicates whether site is at the A end (from) or Z end (to) of the service
    - <children-Set> - tag to enclose children objects
      - Termination point object - optical.TerminationPoint
      - actionMask - specifies the create operation
      - <portPointer> - assigns a port. See the Parent Hierarchy of the port or channel object such as equipment.PhysicalPort for the FDN.

Figure 19-20 Optical transport service creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>svc-mgr</distinguishedName>
   <childConfigInfo>
      <optical.TransportService>
```

```
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <serviceId/>
         <subscriberPointer/>
         <rate/>
         <children-Set>
            <optical.ServiceSite>
               <actionMask>
                  <bit>create</bit>
               </actionMask>
               <siteId/>
               <sitePosition/>
               <children-Set>
                  <optical.TerminationPoint>
                     <actionMask>
                        <bit>create</bit>
                     </actionMask>
                     <portPointer/>
                  </optical.TerminationPoint>
               </children-Set>
            </optical.ServiceSite>
         </children-Set>
      </optical.TransportService>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

19.23 Composite services

19.23.1 Overview

A composite service is a set of linked services. Composite service functions support complex applications that require a combination of services, such as VLAN connections to an HVPLS, an IES spoke into a VPLS, or a VPRN-to-VPLS interconnection.

Composite services exist only in the context of the NFM-P and are configured using the NFM-P GUI or an OSS application. The composite services are unknown to individual network devices.

See “Composite service management” in the *NSP NFM-P Classic Management User Guide* for more information about composite services, connector types, sample composite service configuration, and for the prerequisites and workflow to create a composite service.

19.24 Composite service configuration

19.24.1 Workflow

The XML API uses the service.CompositeServiceManager (comp-svc-mgr) class to manage composite services. The following workflow describes how to create a composite service.

1

Create a composite service.

2

Add service components (such as an Epipe and a VPLS) to the composite service.

3

Create connectors to link the service components.

19.24.2 Examples

The following example outlines what is required for an XML request to create a composite service. One type of composite service is a VLL such as an Epipe spoke into a VPLS service.

The generic.GenericObject.configureInstance method is used to create the composite service. The service.CompositeService class contains methods to add or create service components to the composite service, and methods to create connectors for the service components in the composite service. See the XML API Reference for information about these methods and the required input parameters.

Composite service creation parameters

- method - generic.GenericObject.configureInstance
- distinguishedName - comp-svc-mgr. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference
- configInfo - creates and configures the following service parameters:
  - Service object - service.CompositeService
  - actionMask - specifies the create operation
  - <compositeSvcId> - assigns a unique composite service ID.

Figure 19-21 Composite service creation request example

```
<generic.GenericObject.configureInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>comp-svc-mgr</distinguishedName>
   <configInfo>
      <service.CompositeService>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <compositeSvcId/>
      </service.CompositeService>
   </configInfo>
</generic.GenericObject.configureInstance>
```

Composite service add service parameters

- method - service.CompositeService.addServices
- instanceFullName - comp-svc-mgr:cmp-service-${*id}*. The FDN can be determined by looking at

the Parent Hierarchy of the class to be created in the XML API Reference

- alnFdns - pointers to the services to be added
  - <pointer> - svc-mgr:service-${*id}* - add one for each service

Figure 19-22 Composite service addition request example

```
<service.CompositeService.addServices xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <instanceFullName>comp-svc-mgr:cmp-service-1002</instanceFullName>
   <aInFdns>
      <pointer>svc-mgr:service-84</pointer>
      <pointer>svc-mgr:service-87</pointer>
   </aInFdns>
</service.CompositeService.addServices>
```

Composite service connector parameters

- method - see Table 19-8, “Composite service connector methods and objects” (p. 287)
- instanceFullName - comp-svc-mgr:cmp-service-${*id}*. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference
- Configuration input parameter - see Table 19-8, “Composite service connector methods and objects” (p. 287)
  - Connector object - see Table 19-8, “Composite service connector methods and objects” (p. 287)
  - Other properties to be configured depend on the connector type. The sample in Figure 19-23, “Composite service SCP connector creation request example” (p. 287) is for a SCP connector.

Table 19-8 Composite service connector methods and objects

Connector type	Method (service.CompositeService.*method)*	Configuration parameter	Connector object
SCP	createScpConnector	configInfoConnector	service.ScpConnector
Spoke	createSpokeConnector	cfgSpokeConnector	service.SpokeConnector
Cross-connect	createCrossConnect	cfgCrossConnect	service.CrossConnect
Routed VPLS	createRoutedVplsConnector	cfgRoutedVplsConnector	service.RoutedVplsConnector

Figure 19-23 Composite service SCP connector creation request example

```
<service.CompositeService.createScpConnector xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <instanceFullName>comp-svc-mgr:cmp-service-1002</instanceFullName>
   <configInfoConnector>
      <actionMask>
         <bit>create</bit>
      </actionMask>
```

```
      <displayedName>abcdef</displayedName>
      <firstSvcPointer>svc-mgr:service-serviceIdA</firstSvcPointer>
      <secondSvcPointer>svc-mgr:service-serviceIdB</secondSvcPointer>
      <firstSitePointer>svc-mgr:service-serviceIdA:siteIdA</firstSitePointer>
      <secondSitePointer>svc-mgr:service-serviceIdB:siteIdB</secondSitePointer>
      <firstScpPointer>svc-mgr:service-serviceIdA:siteIdA:interface-portAandEncapValues
</firstScpPointer>
      <secondScpPointer>svc-mgr:service-serviceIdB:siteIdB:interface-portBandEncapValues
</secondScpPointer>
   </configInfoConnector>
</service.CompositeService.createScpConnector>
```

19.25 Residential subscriber configuration

19.25.1 Overview

The XML API uses the ressubscr package to create and manage residential subscribers. A residential subscriber is an end recipient of multiple service offering, such as VoD, VoIP, or other triple play applications. A residential subscriber host, which is sometimes called a subscriber host or a host, is an end device, such as a computer or set-top box that connects to the provider network and receives the service traffic.

In the context of the XML API, a residential subscriber, which is sometimes called a subscriber, has a unique identifier that associates a group of end-user devices with policies. A subscriber can be associated with multiple SAPs on multiple NEs, and a customer can be associated with multiple subscribers.

To configure residential subscriber management on the NFM-P, create and configure the following components, depending on the service delivery model:

- subscriber identification policy
- subscriber profile
- SLA profile
- subscriber explicit map
- ANCP policy
- MSAP policy
- host tracking policy
- category map policy
- credit control policy
- IGMP policy
- BGP peering policy
- diameter policy
- subscriber multicast CAC policy

A host requires subscriber-profile and SLA-profile associations to access the network. Profiles define service attributes for hosts such as scheduling, accounting, security, and traffic prioritization by application type. A profile uses existing NFM-P policy definitions and allows the customization of policy parameters using override values.

See “Residential subscriber management” in the *NSP NFM-P Classic Management User Guide* for more information about residential subscriber management, including prerequisites and workflows.

19.25.2 Residential subscriber related packages

- Residential subscriber manage - ressubscr
- Subscriber profile - subscrprofile

- Subscriber identification - subscrident
- Subscriber authentication - subscrauth
- Subscriber explicit map - subscrexpmap
- SLA profile - slaprofile
- ANCP policy - ancp
- ARP host - arp
- Credit control policy - crdtctrl
- Diameter policy - diameter
- MSAP policy - msapolicy
- PPP management - ppp, ppoe
- MSAP, group interface, IGMP host tracking, subscriber interface - vprn, ies

19.25.3 Example

The following example outlines what is required for an XML request to create a subscriber identification policy for residential subscribers.

Subscriber identification policy creation parameters

- method - generic.GenericObject.configureChildInstance
- distinguishedName - Subscriber Identification Policy. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference
- childConfigInfo - creates and configures the following policy parameters:
  - Policy object- subscrident.Policy
  - <displayedName> - assigns a unique name
  - <children-Set>- tag to enclose children objects
    - Subscriber profile entry object- subscrident.SubscrProfileEntry
    - <displayedName> - assigns a name
    - <subscrProfilePointer>- pointer to the subscriber profile
    - <subscrIdentPolicyName> - name of the subscriber identification profile

Figure 19-24 Subscriber identification policy creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <synchronousDeploy>true</synchronousDeploy>
   <distinguishedName>Subscriber Identification Policy</distinguishedName>
   <childConfigInfo>
      <subscrident.Policy>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <displayedName/>
          ..
         <children-Set>
```

```
            <subscrident.SubscrProfileEntry>
               <actionMask>
                  <bit>create</bit>
               </actionMask>
               <displayedName>Subscr Profile 4</displayedName>
               <subscrProfilePointer>Subscriber Profile:Subscriber Profile Example
10</subscrProfilePointer>
               <subscrIdentPolicyName/>
            </subscrident.SubscrProfileEntry>
         </children-Set>
      </subscrident.Policy>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

20 Script and template configuration management

20.1 Script management

20.1.1 Overview

WARNING

Equipment Damage

CLI and XML API scripts that are not correctly applied or created can cause serious damage to the network.

Nokia recommends that system administrators clearly define user responsibilities for script usage, and ensure that scripts are verified and validated before they are executed on devices in a live network.

See the appropriate device hardware documentation for the applicable CLI commands, structure, and usage.

Script management is typically delivered by the NFM-P GUI which provides the framework to develop scripts to create, delete, configure, or view objects in the NFM-P network. An OSS client can use the NFM-P script manager for the following:

- invoking XML scripts outside of the NFM-P GUI framework
- XML script and OSS software troubleshooting

The XML API uses the script package to allow for script management. There are two types of scripts:

- CLI script—used to securely create and manage scripts executed against managed devices, including GNEs
- XmlAPI script—is the same type of script used for OSS but are typically managed within NFM-P, allowing them to be used from the NFM-P GUI, and perform complex actions on not just network elements, but NFM-P itself, if required.

Note: The generic methods are not used to configure scripts, instead use the various configuration methods that are included for the various classes in the script package. See the XML API Reference for more information.

20.2 Workflow to execute a script

20.2.1 Overview

The following workflow outlines the high-level steps necessary to create either CLI or XML API scripts. Each of the steps provides guidance on the type of XML requests required to perform a particular configuration action. See the XML API SDK Sample Code Navigator for sample XML scripts.

You can synchronously or asynchronously execute the scripts. For synchronous execution, a response attribute identifies the success or failure of script execution. For asynchronous execution, the OSS client can either wait for a JMS notification or poll for status on the execution of the script.

The script results are stored in a file for the OSS client to collect. When the OSS client retrieves the results using the SOAP interface, the result is stripped of non-printable characters embedded in the response.

20.2.2 Stages

1

Create script object using the script.Scriptmanager class.

a. Create CLI script object.

b. Create XML API script object.

2

Create script version.

a. For CLI script version, include the CLI commands for the target NEs.

b. For XML API script version, include velocity content and configuration details.

3

Create script target on instance

a. For CLI, create a script target that identifies the NE and specific NE parameter targets.

b. For XML API, create a script instance for an XML API script that identifies the targetObjectFullName of the object and parameters to be configured.

4

Preview scripts.

5

Start and stop the execution of scripts.

6

Retrieve script results of executions using FTP SFTP, or SOAP/XML.

20.3 XML API template configuration management

20.3.1 Overview

A template is an object managed by the NFM-P that is used as a starting point for configuring complex managed objects such as services and tunnels. The NFM-P supports the configuration of templates using XML API script-based templates for services, tunnels and some equipment. Templates also use the NFM-P GUI framework to create, modify, or delete template-able objects.

Templates use the NFM-P GUI framework for configuring services, tunnels and equipment as an alternative to OSS requests, even though the same XML API commands are used.

21 CPAM OSS interface

21.1 CPAM OSS interface

21.1.1 Overview

The CPAM is an IP/MPLS control plane management system that allows service providers to offer network and service availability, and diagnose control plane configuration errors, malfunctions, and undetected routing updates. The CPAM facilitates service problem resolution over an IP/MPLS infrastructure.

The CPAM OSS interface includes CPAM packages and methods that provide a smooth integration path for the export of real-time protocol or IP-level topology to advanced OSS applications, such as TE tools.

The following CPAM features are of interest to OSS developers:

- CPAM commissioning and administrative domains
- multiple topologies
- checkpoints and real-time topology changes
- route management
- path monitoring
- fault management

OSS applications can use the XML API to create, configure, and manage CPAM objects based on the XML schema. The user privileges associated with an OSS user define the ability to configure NFM-P objects. See the chapter on NFM-P user security in the *NSP System Administrator Guide* for information about assigning permissions to NFM-P user groups and user accounts. See the *NSP NFM-P Control Plane Assurance Manager User Guide* for specific scope of command roles and the associated permissions for CPAM objects.

21.2 References

21.2.1 Overview

The following references are of interest to an OSS developer working in the traffic engineering domain:

- *NSP NFM-P Control Plane Assurance Manager User Guide*— documents the description, workflow, and configuration information about CPAM topology management, checkpoint management, path monitoring, route management, and fault management
- *NSP NFM-P Classic Management User Guide*— documents the description, workflow, and configuration information about routing and service management and integration with the CPAM
- XML API Reference—documents specific classes, attributes, and methods to configure CPAM objects using the XML API. See 21.3 “Key packages and classes” (p. 296) for the packages and classes that are relevant to a TE OSS developer.

- Chapter 8, “XML API Reference” —documents information about how to navigate the XML API Reference
- Chapter 15, “Configuration management overview” —documents configuration methods, general creation, modification, and deletion of NFM-P-managed objects, and describes deployments
- XML API SDK Sample Code Navigator—documents samples of CPAM objects

21.3 Key packages and classes

21.3.1 Overview

CAUTION

Service Disruption

The code samples in the following sections show the structures for object creation or modification, and do not include all of the properties required to create working CPAM objects.

See the XML API SDK Sample Code Navigator for working samples.

Table 21-1 CPAM packages

Package	Description
topology	For routing topology object configuration and management, such as: • administrative domains • autonomous systems • 7701 CPAA • checkpoints • alarm thresholds
monpath	For configuring monitored paths on the network. The CPAM interfaces used to monitor changes to the paths of IP routes and LSPs.
rca	For CPAM policy-driven IP/MPLS configuration audits for OSPF and IS-IS routing protocols
simulator	For defining the infrastructure used in CPAM network simulation and impact analysis
topologysim	For simulated routing topology object configuration and management, such as: • monitored paths • routers It defines the topology simulation model used for IGP and MPLS impact analysis.
multicastmgr	For managing multicast topology objects

Note: The terms distinguishedName, instanceName, and objectFullName are used synonymously in the XML API for the unique identifier of the object instance. See 10.5.4 “objectFullName element” (p. 135) in 10.1 “Schema Reference ” (p. 127) for more information about the object full name.

21.4 CPAM commissioning configuration

21.4.1 Overview

The main components of the control plane assurance management solution are:

- route analyzer component, the 7701 CPAA
- server component, the CPAM

The CPAM server component contains 7701 CPAA control frameworks, applications, and coordination functions for the distributed 7701 CPAAs. The CPAM server also provides the applications that are required to process the 7701 CPAA data.

21.4.2 CPAM commissioning configuration workflow

The following workflow describes the commissioning of the CPAM. See “CPAM commissioning configurations” in the *NSP NFM-P Control Plane Assurance Manager User Guide* for more information about CPAM commissioning.

1

Create the IGP administrative domains, or BGP autonomous systems (AS), or both, as required.

2

Assign role(s) to the 7701 CPAAs and associate every 7701 CPAA to an IGP administrative domain, or BGP AS, or both.

3

Enable the 7701 CPAA.

21.4.3 CPAM administrative domains

An administrative domain represents a logical routed network. Administrative domains are configured by the operator to reflect the logical structure of the network. The following types of administrative domains are supported:

- **IGP administrative domain**

  Represents an IGP routed network. In an IGP administrative domain, there can be both an OSPF and an IS-IS network.

- **BGP autonomous system**

  Represents either a standard BGP AS, a BGP Confederation AS, or a BGP Sub-AS (confederation member). BGP Sub-ASs are configured as members of a Confederation BGP AS.

See the *NSP NFM-P Control Plane Assurance Manager User Guide* for the rules governing the assignment of administrative domains.

Following are the minimum XML parameter tags required to develop an XML request to create an IGP administrative domain and a BGP AS.

The generic.GenericObject.configureChildInstance method is used to create an IGP administrative domain and a BGP AS. The request response returns the <objectFullName> attribute to identify the new IGP administrative domain and BGP AS, respectively.

IGP administrative domain creation parameters

- method—generic.GenericObject.configureChildInstance
- distinguishedName—tpgy-mgr. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference. In this case, it is the topology.TopologyManager class.
- childConfigInfo—creates and configures the following parameters:
  - IGP administrative domain object - topology.AutonomousSystem
  - actionMask - specifies the create operation
  - <asNumber> - (optional) assigns a numeric ID. If not specified, the ID will be zero. The asNumber and displayedName together need to be unique.
  - <displayedName> - assigns a name. The asNumber and displayedName together need to be unique.
  - <menuControl> - sets the bits for enabling the protocols for the administrative domain

Figure 21-1 IGP administrative domain creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>tpgy-mgr</distinguishedName>
   <childConfigInfo>
      <topology.AutonomousSystem>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <asNumber>10</asNumber>
         <displayedName>IGPadminDomain1</displayedName>
         <menuControl>
            <bit>ospf</bit>
            <bit>isis</bit>
            <bit>igp</bit>
         </menuControl>
      </topology.AutonomousSystem>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

BGP autonomous system creation parameters

- method—generic.GenericObject.configureChildInstance
- distinguishedName—tpgy-mgr. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference. In this case, it is the topology.TopologyManager class.
- childConfigInfo—creates and configures the following parameters:
  - BGP AS object - topology.BgpAutonomousSystem
  - actionMask - specifies the create operation

- <asNumber> - assigns a unique numeric ID.
- <asType> - sets the BGP AS type (as is standard type, confederation is confederation type)
- <displayedName> - assigns a unique name.
- <igpAdminDomain> - tpgy-mgr:name-${*displayedName*}-AS-${*asNumber*} - pointer to the IGP administrative domain. This parameter is not required for a BGP confederation AS.

Figure 21-2 Standard BGP AS creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>tpgy-mgr</distinguishedName>
   <childConfigInfo>
      <topology.BgpAutonomousSystem>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <asNumber>35</asNumber>
         <asType>as</asType>
         <displayedName>BGPAS1</displayedName>
         <igpAdminDomain>tpgy-mgr:name-IGPadminDomain1-AS-10</igpAdminDomain>
      </topology.BgpAutonomousSystem>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

Figure 21-3 BGP confederation AS creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>tpgy-mgr</distinguishedName>
   <childConfigInfo>
      <topology.BgpAutonomousSystem>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <asNumber>50</asNumber>
         <asType>confederation</asType>
         <displayedName>BGPAS2</displayedName>
      </topology.BgpAutonomousSystem>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

21.4.4 7701 CPAA configuration

You can specify the role of a 7701 CPAA to be IGP, BGP, or both. The role of the 7701 CPAA indicates to the CPAM whether information from a protocol is interpreted. An IGP administrative domain must be assigned to a 7701 CPAA with an IGP role. A BGP AS must be assigned to a 7701 CPAA with a BGP role.

7701 CPAA set role parameters

- method—generic.GenericObject.configureChildInstance
- distinguishedName—network:${*systemAddress*}:cpaa. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference. In this case, it is the topology.Cpaa class.
- configInfo—creates and configures the following parameters:
  - 7701 CPAA object - topology.Cpaa
  - actionMask - specifies the modify operation
  - <role> - sets the role of the 7701 CPAA
  - <asPointer> - tpgy-mgr:name-*${displayedName}*-AS*${asNumber}* - pointer to the administrative domain

Figure 21-4 7701 CPAA IGP role-setting request example

```
<generic.GenericObject.configureInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>network:198.51.100.70:cpaa</distinguishedName>
   <configInfo>
      <topology.Cpaa>
         <actionMask>
            <bit>modify</bit>
         </actionMask>
         <role>
            <bit>igp</bit>
         </role>
         <asPointer>tpgy-mgr:name-IGPadminDomain1-AS-10</asPointer>
      </topology.Cpaa>
   </configInfo>
</generic.GenericObject.configureInstance>
```

Set event types, keep event history, and enable 7701 CPAA parameters

- method—generic.GenericObject.configureInstance
- distinguishedName—network:${*systemAddress*}:cpaa. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference. In this case, it is the topology.Cpaa class.
- configInfo—creates and configures the following parameters:
  - 7701 CPAA object - topology.Cpaa
  - actionMask - specifies the modify operation
  - <protocolEventTypes> - sets the events that the 7701 CPAA will notify the CPAM
  - <protocolRecord> - sets the protocol events to be recorded
  - <administrativeState> - enables or disables the communication channels from the CPAM to the 7701 CPAA

The following figure shows an example of a request to set event types, keep an event history, and enable the 7701 CPAA.

Figure 21-5 7701 CPAA configuration request example

```
<generic.GenericObject.configureInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>network:198.51.100.70:cpaa</distinguishedName>
   <configInfo>
      <topology.Cpaa>
         <actionMask>
            <bit>modify</bit>
         </actionMask>
         <protocolEventTypes>
            <bit>ospf</bit>
            <bit>isis</bit>
         </protocolEventTypes>
         <protocolRecord>
            <bit>ospf</bit>
            <bit>isis</bit>
            <bit>ospfTe</bit>
         </protocolRecord>
         <administrativeState>up</administrativeState>
      </topology.Cpaa>
   </configInfo>
</generic.GenericObject.configureInstance>
```

21.5 Topology and checkpoint management

21.5.1 Overview

After the CPAM is commissioned and the proper IGP administrative domain, or BGP AS, or both, are assigned to the 7701 CPAAs and the 7701 CPAAs are up, IGP topologies are automatically populated in the CPAM.

21.5.2 Multiple topologies

The CPAM allows OSS applications to retrieve information about the following network protocol topologies:

- OSPF
- OSPFv3
- IS-IS
- IGP
- BGP
- MPLS—MPLS and LDP information is collected from the NFM-P

See “Topology management” in the *NSP NFM-P Control Plane Assurance Manager User Guide* for more information about the supported topologies.

The XML API provides a read-only interface for topology objects, such as areas, routers, links, networks, and baselines. The JMS interface notifies the XML API of any changes to the topology

model, including baselines. See Chapter 4, “Event monitoring using JMS” for information about how to connect and subscribe to JMS.

The CPAM topology package is used to manage routing topology objects and routing alarms. The CPAM supports IS-IS, OSPF, OSPFv3, and BGP.

OSPF, OSPFv3, and IS-IS topologies include information about routers, subnet objects, links, OSPF/OSPFv3 areas, IS-IS levels, and LSDB updates. The IGP topology includes information about routers, protocols for both OSPF, OSPFv3, and IS-IS, and information about IGP links. The following table shows the routing topology objects for IS-IS, OSPF, OSPFv3, and IGP.

Table 21-2 OSPF, IS-IS, and IGP routing topology objects

Object	Package and class
OSPF and OSPFv3	
Area	topology.Area
Area Link	topology.OspfLink
Router	topology.Router
Area Subnet	topology.Subnet
IS-IS	
Area	topology.Area
Domain Link	topology.IsisLink
Router	topology.Router
Domain Subnet	topology.IsisSubnet
IGP	
Link	topology.IgpLink
Router	topology.Router

The find or findToFile method, along with the appropriate class specified in Table 21-2, “OSPF, IS-IS, and IGP routing topology objects” (p. 302) , and a filter, can be used to retrieve topology information.

A real-time topology object shares the same XML API class as a checkpointed data object; therefore, in the find or findToFile method filter, you need the following to distinguish real-time data from checkpointed data.

For real-time data:

```
<equal name='isCheckpointObject' value='false'/>
```

For checkpointed data:

```
<equal name='isCheckpointObject' value='true'/>
```

Retrieving IGP topology data

- method—find

- fullClassName—the class where data is to be retrieved from. For Figure 21-6, “find request example, IGP links” (p. 302) , the class is the topology.IgpLink class.
- filter—set a filter to only return IGP link objects that satisfy the filter criteria
  - The filter criteria is to look for real-time data (not checkpointed data), with administrative domain name of igp0, and router ID of 198.51.100.57.

Figure 21-6 find request example, IGP links

```
<find xmlns="xmlapi_1.0">
   <fullClassName>topology.IgpLink</fullClassName>
   <filter>
      <and>
         <equal name="isCheckpointObject" value="false"/>
         <equal name="asName" value="igp0"/>
         <equal name="routerIdStr" value="198.51.100.57"/>
      </and>
   </filter>
</find>
```

See Chapter 13, “Inventory management” for more information about using the find or findToFile method and filters.

The MPLS topology includes information about IGP links, MPLS, RSVP, and LDP interfaces for routers.

The BGP topology includes BGP route information from a 7701 CPAA that monitors a BGP AS and maintains a BGP RIB. The 7701 CPAA supports the following BGP configurations:

- BGP AS
- BGP Confederation AS
- BGP Sub-AS

Table 21-3 BGP topology objects

Package and class	Description
topology.Asn	AS number in BGP AS path attribute
topology.AsnLink	BGP ASN link between two autonomous systems
topology.BgpOriginatedAs	BGP-originated autonomous system number
topology.BgpRibInfo	BGP RIB-IN info
topology.BgpRibInfoValue	BGP RIB info value
topology.BgpRoutesNextHop	VPN IPv4 next hop
topology.BgpRoutesRouteTarget	VPN IPv4 route target

21.5.3 Topology operations

Table 21-4 Topology operation methods

Method	Description
topology.TopologyManager.constrainedShortestPathFirstGraphExt	Performs a CSPF calculation from the specified source to the specified destination, using the specified constraints, as known by the specified IGP protocol
topology.TopologyManager.igpShortestPathFirstGraphExt	Performs an SPF calculation from the specified source to the specified destination, using OSPF, IS-IS, or OSPF and IS-IS
topology.TopologyManager.nextHopExt	Calculates the next hop from all routers in the source set to the specified destination FEC
topology.TopologyManager.shortestPathFirstGraphExt	Performs an SPF calculation from the specified source to the specified destination using the specified IGP protocol See Figure 21-7, “find request example, SPF between two NEs” (p. 304) for a method example.

Perform SPF calculation

- method — topology. TopologyManager.shortestPathFirstGraphExt
- <igpAdminDomain> — tpgy-mgr:name-${*displayedName*}-AS-${*asNumber*}, which is a pointer to the IGP administrative domain to perform the SPF in. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference. In this case, it is the topology.AutonomousSystem class.
- <protocol>—sets the protocol to run SPF on
- <sourceType>—sets the address type of the source
- <source>— ets the source address
- <sourceLen>—sets the length of the source address
- <destType>—sets the address type of the destination
- <dest>—sets the destination address
- <destLen>— ets the length of the destination address

Figure 21-7 find request example, SPF between two NEs

```
<topology.TopologyManager.shortestPathFirstGraphExt xmlns="xmlapi_1.0">
   <igpAdminDomain>tpgy-mgr:name-AdminDomain1-AS-1</igpAdminDomain>
   <protocol>ospf</protocol>
   <sourceType>ipv4</sourceType>
   <source>198.51.100.71</source>
   <sourceLen>32</sourceLen>
   <destType>ipv4</destType>
   <dest>198.51.100.72</dest>
   <destLen>32</destLen>
</topology.TopologyManager.shortestPathFirstGraphExt>
```

See the XML API Reference for descriptions of the input parameters. See the XML API SDK Sample Code Navigator under the Configuration/CPAM directory for other sample topology operation scripts and the results after executing these types of scripts.

21.5.4 Topology checkpoints

The CPAM can be used to analyze the effect of a network failure on services, IP paths, and LSPs, as well as configuration changes, dynamic changes such as LSP configuration, and reroutes during a specified time period. To analyze these effects, checkpoints can be configured and used to compare historical topology changes in a specified interval on the IGP topology.

A checkpoint object is a snapshot of a real topology object at a specific time. When you apply a checkpoint to a real network object, all of the properties of the real object at checkpoint time—for example, metric and bandwidth on IGP links—are copied to the checkpointed object.

After you have set up your network and the network is operational, you can checkpoint the network to create a snapshot of the current state and compare it with checkpoints collected at different times. An OSPF topology checkpoint is created for all of the OSPF areas in an IGP administrative domain. An ISIS routing topology checkpoint is created for all of the ISIS routing domains—Level 2 or Level 1—in an IGP administrative domain.

See “Topology management” in the *NSP NFM-P Control Plane Assurance Manager User Guide* for more information about topology checkpoints and objects that are included in OSPF and IS-IS checkpoints.

21.5.5 Checkpoint configuration workflow

The following workflow describes the configuration of checkpoints and the usage of checkpointed data.

1

Create topology checkpoints for an administrative domain.

2

Retrieve checkpointed data.

3

Compare checkpointed data.

4

Schedule automatic checkpoint creations.

21.5.6 Checkpoint creation

You can create checkpoints using the following methods:

- using the generic configure method and the appropriate classes, such as topology.OspfCheckpoint and topology.IsisCheckpoint. This method creates a specific type, OSPF or IS-IS, of checkpoint in an IGP administrative domain. A name and description can be set for the checkpoint.

- using the topology.Checkpoint.autoCreateCheckpoints method. This method creates all supported types, such as OSPF, IS-IS, of checkpoints at once, or creates a combination of the supported types of checkpoints in an IGP administrative domain.

Using the generic configure method

- method—generic.GenericObject.configureChildInstance
- distinguishedName—tpgy-mgr:name-${*displayedName*}-AS-${*asNumber*}. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference.
- childConfigInfo—creates and configures the following parameters:
  - Checkpoint object:
    - for OSPF checkpoint - topology.OspfCheckpoint
    - for IS-IS checkpoint - topology.IsisCheckpoint
  - actionMask - specifies the create operation
  - <id> - (optional) assigns a unique ID. If not specified, the NFM-P automatically assigns the ID.
  - <displayedName> - assigns a unique name
  - <description> - describes the checkpoint

Using the autoCreateCheckpoints method

- method—topology.Checkpoint.autoCreateCheckpoints
- <igpAdminDomain> — tpgy-mgr:name-${*displayedName*}-AS-${*asNumber*} is a pointer to the IGP administrative domain that is being checkpointed. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference.
- <protocolMask>—sets the protocols. Set all the applicable bits.
  - <bit> - topology.CheckPoint bitmask value

See the XML API Reference for descriptions of the input parameters. See the XML API SDK Sample Code Navigator under the Configuration/CPAM directory for a sample of these scripts.

21.5.7 Checkpointed data retrieval

After checkpoints are set, you can retrieve the checkpointed topology objects, such as routers, IGP links, and subnets. The following table shows the topology checkpoint information that can be retrieved.

Table 21-5 Checkpoint objects

Object	Package and class
OSPF	
Area	topology.Area
Area Link	topology.OspfLink
Router	topology.Router
Area Subnet	topology.Subnet
IS-IS	

Table 21-5 Checkpoint objects (continued)

Object	Package and class
Area	topology.Area
Domain Link	topology.IsisLink
Router	topology.Router
Domain Subnet	topology.IsisSubnet
Non-routed	
Link	topology.NonRoutedLink
Subnet	topology.NonRoutedSubnet

You can retrieve checkpointed data using the following methods:

- Using the topology.Checkpoint.getCpTopology method. This method retrieves only the objects that exist in the specified checkpoint.
- Using the find or findToFile method with the appropriate classes specified in Table 21-5, “Checkpoint objects” (p. 306) , and a filter. This method retrieves objects in different checkpoints depending on what is specified in the filter.

Using the getCpTopology method to retrieve checkpointed data

- method—topology.Checkpoint.getCpTopology
- <checkpointId> — ID of the checkpoint from where to retrieve data
- <asNumber>—IGP administrative domain number of the checkpoint
- <asName> — IGP administrative domain name of the checkpoint
- <protocol>—protocol to which the checkpoint belongs, which is ospf for OSPF, or isis for IS-IS.
- <fullClassName> — list of checkpointed objects, from Table 21-5, “Checkpoint objects” (p. 306) , separated by a comma. For example, topology.Router, topology.OspfLink
- <resultFilter>—(optional) filter for reducing the scope of the returned information

The following figure shows an example of a request that uses the getCpTopology method to retrieve checkpointed data.

Figure 21-8 Checkpointed data retrieval request example

```
<topology.Checkpoint.getCpTopology xmlns="xmlapi_1.0">
   <checkpointId>1</checkpointId>
   <asNumber>1</asNumber>
   <asName>AdminDomain-1</asName>
   <protocol>ospf</protocol>
   <fullClassName>topology.Router</fullClassName>
   <resultFilter/>
</topology.Checkpoint.getCpTopology>
```

See the XML API Reference for descriptions of the input parameters.

A checkpointed object shares the same XML API class as a real-time topology object; therefore, in the find or findToFile method filter, you need the following to distinguish checkpointed data from real-time data.

For real-time data:

```
<equal name='isCheckpointObject' value='false'/>
```

For checkpointed data:

```
<equal name='isCheckpointObject' value='true'/>
```

Using the find method to retrieve checkpointed data

- method—find
- fullClassName—the class where data is to be retrieved from. For Figure 21-9, “Checkpointed OSPF link retrieval request example” (p. 307) , the class is the topology.OspfLink class.
- filter—sets a filter to only return OSPF link objects that satisfy the filter criteria
  - The filter criteria is to look for checkpointed data, with administrative domain name of igp0, and router ID of 198.51.100.57.

The following figure shows an example of a request to find checkpointed OSPF links that match a filter.

Figure 21-9 Checkpointed OSPF link retrieval request example

```
<find xmlns="xmlapi_1.0">
   <fullClassName>topology.OspfLink</fullClassName>
   <filter>
      <and>
         <equal name="isCheckpointObject" value="true"/>
         <equal name="asName" value="igp0"/>
         <equal name="routerIdStr" value="198.51.100.57"/>
      </and>
   </filter>
</find>
```

See Chapter 13, “Inventory management” for more information about using the find and findToFile method and filters.

21.5.8 Checkpoint comparison

You can use the CPAM to diagnose problems in the network by comparing two checkpoints of the network and viewing the information about the differences in configuration and topology changes.

When you compare two checkpoints, you can specify whether the comparison is of checkpointed areas, routers, links, or subnets, or a combination of these objects. The first checkpoint that you select is the basis for the comparison. You can swap the order of the checkpoints so that the second checkpoint that you select becomes the basis for the comparison. You can specify a filter to limit what comparison results are returned.

Comparing checkpointed data

- method—generic.GenericObject.triggerIncrementalRequest

- <aInIncrementalContext>—Application specific configuration parameters are set inside this tag.
  - <generic.IncrementalContext> - structure that contains the parameters for this method
  - <jmsClientId> - sets the JMS client ID. The ID can be used to filter JMS events for the result.
  - <handlerName> - specifies the application to run. In this case, the application is the compare object that needs to be set to GEN_COMPARE_OBJECTS.
  - <appDefinedContext> - sets the parameters for the handler
    - <generic.ComparisonParam>—structure for the parameters for the comparison
      - <comparisonFilter> - reduces scope of comparison
      - <base> - basis of comparison
      - <compareTo> - basis if being compared to this object

Figure 21-10 Checkpoint comparison request example

```
<generic.GenericObject.triggerIncrementalRequest xmlns="xmlapi_1.0">
   <aInIncrementalContext>
      <generic.IncrementalContext>
         <jmsClientId>JMS_client@n</jmsClientId>
         <handlerName>GEN_COMPARE_OBJECTS</handlerName>
         <appDefinedContext>
            <generic.ComparisonParam>
               <comparisonFilter>
                  <generic.ComparisonFilter>
                     <classesToProcess/>
                     <includeOnlyDiffs>true</includeOnlyDiffs>
                  </generic.ComparisonFilter>
               </comparisonFilter>
               <base>tpgy-mgr:name-igp0-AS-0:OspfCheckpoint-1</base>
               <compareTo>tpgy-mgr:name-igp0-AS-0:OspfCheckpoint-3</compareTo>
            </generic.ComparisonParam>
         </appDefinedContext>
      </generic.IncrementalContext>
   </aInIncrementalContext>
</generic.GenericObject.triggerIncrementalRequest>
```

See the XML API Reference for descriptions of the input parameters.

The result of the comparison is retrieved through the JMS interface. An OSS needs to subscribe to JMS to receive the result. See Chapter 4, “Event monitoring using JMS” for information about how to connect and subscribe to JMS.

For a response, look for the JMS client ID that you specified in the request in Figure 21-10, “Checkpoint comparison request example” (p. 309) in the ALA_clientId attribute in the JMS message header and also filter for the IncrementalRequestEvent in the JMS message header.

Figure 21-11 IncrementalRequestEvent JMS header example

```
<header xmlns="xmlapi_1.0">
   <eventName>IncrementalRequestEvent</eventName>
   <ALA_category>GENERAL</ALA_category>
```

```
   <ALA_OLC>0</ALA_OLC>
   <ALA_isVessel>false</ALA_isVessel>
   <ALA_allomorphic/>
   <MTOSI_osTime>1300820624174</MTOSI_osTime>
   <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
   <MTOSI_objectName/>
   <ALA_clientId>JMS_client@n</ALA_clientId>
   <MTOSI_objectType>IncrementalRequestEvent</MTOSI_objectType>
   <ALA_eventName>IncrementalRequestEvent</ALA_eventName>
</header>
```

See the XML API SDK Sample Code Navigator under the Configuration/CPAM directory for a sample of the complete JMS event in response to the request in Figure 21-10, “Checkpoint comparison request example” (p. 309) .

21.5.9 Checkpoint schedule policies

Checkpoints can be automatically created using a schedule. Scheduled checkpoint creation requires a checkpoint schedule policy and checkpoint scheduled task. See “Topology checkpoints” in the *NSP NFM-P Control Plane Assurance Manager User Guide* for information about how to configure checkpoint schedule policies and scheduled tasks, and for considerations specific to checkpoint scheduled task creation. See “NFM-P-based schedules” in the *NSP NFM-P Classic Management User Guide* for general information about schedules and scheduled tasks.

The following are the classes for scheduling checkpoints and defining schedule policies:

- **topology.CheckpointScheduledTask**
  represents a specialized Checkpoint Scheduled Task used for performing scheduled checkpoints
- **topology.CpSchedulePolicy**
  defines the policy for creating scheduled checkpoints

21.6 Route management

21.6.1 Overview

A managed route is an IP path from a specified source router to a specified destination router. An IP path represents one or more GRE or LDP tunnels, and loose-path RSVP LSPs, of the same source and destination. Managed routes can span multiple areas and, therefore, multiple 7701 CPAAs.

Multiple managed routes are stored by the route manager object and any changes to the paths of managed routes are monitored via the JMS event channel. A separate route manager must be created for each OSS application. This separate route allows for separate sets of managed routes for different applications. JMS events may be filtered by application.

The route manager and its set of managed routes are persisted. The paths of the managed route are not persisted.

21.6.2 Route management workflow

The following workflow describes how to set up to monitor managed routes.

1

Create a managed route client.

2

Create a request to add or register managed routes.

3

Monitor managed route changes via the JMS interface.

4

Retrieve managed routes information.

21.6.3 Route management configuration

The XML API supports the creation of managed route client, creation of request to register managed routes, and retrieval of managed routes information.

The following examples show the minimum XML parameter tags required to develop an XML request to create a managed route client.

The generic.GenericObject.configureChildInstance method is used to create a managed route client, which uses the topology.TopologyManager class singleton as the parent. The request response returns the <objectFullName> attribute to identify the new managed route object.

Managed route client creation parameters

- method—generic.GenericObject.configureChildInstance
- distinguishedName—tpgy-mgr. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference. In this case, it is the topology.TopologyManager class.
- childConfigInfo—creates and configures the following parameters:
  - Route manager object- topology.RouteManager
  - actionMask - specifies the create operation
  - <applicationName> - assigns a unique name

Figure 21-12 Managed route client creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>tpgy-mgr</distinguishedName>
   <childConfigInfo>
      <topology.RouteManager>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <applicationName>TestRoutes</applicationName>
      </topology.RouteManager>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

The following examples show the minimum XML parameter tags required to develop an XML request to register a managed route.

Managed route registration parameters

- method—topology.RouteManager.registerRoutes
- <instanceFullName>—tpgy-mgr:application-${*applicationName*}, which is the FDN of the managed route client
- <routeKeySet>—the set of route keys to register
  - <topology.RouteKey> - creates and configures the following route key parameters:
    - sourceType
    - source
    - sourceLen
    - destType
    - dest
    - destLen

Figure 21-13 Add or register managed routes request example

```
<topology.RouteManager.registerRoutes xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <instanceFullName>tpgy-mgr:application-TestRoutes</instanceFullName>
   <routeKeySet>
      <topology.RouteKey>
         <sourceType>ipv4</sourceType>
         <source>198.51.100.45</source>
         <sourceLen>32</sourceLen>
         <destType>ipv4</destType>
         <dest>198.51.100.47</dest>
         <destLen>32</destLen>
      </topology.RouteKey>
   </routeKeySet>
</topology.RouteManager.registerRoutes>
```

The route can be deregistered by using the topology.RouteManager.deregisterRoutes method with the same parameters as the topology.RouteManager.registerRoutes method. See the XML API Reference for descriptions of the input parameters. See the XML API SDK Sample Code Navigator under the Configuration/CPAM directory for a sample of this script.

21.6.4 Route management retrieval

The route can be retrieved by using the topology.RouteManager.retrieveRoutes method with the same parameters as the topology.RouteManager.registerRoutes method. See the XML API Reference for descriptions of the input parameters. See the XML API SDK Sample Code Navigator under the Configuration/CPAM directory for a sample of this script.

The following figure shows an example of a response from the topology.RouteManager.retrieveRoutes method.

Figure 21-14 Managed route retrieval response example

```
<topology.RouteManager.retrieveRoutesResponse xmlns="xmlapi_1.0">
   <result>
      <item>
         <key>
            <topology.RouteKey>
               <sourceType>ipv4</sourceType>
               <source>198.51.100.45</source>
               <sourceLen>32</sourceLen>
               <destType>ipv4</destType>
               <dest>198.51.100.47</dest>
               <destLen>32</destLen>
            </topology.RouteKey>
         </key>
         <value>
            <topology.RouteResult>
               <errorCode>noError</errorCode>
               <errorMessage>No Error</errorMessage>
               <type>standard</type>
               <vertices>
                  <topology.RouterVertex>
                     <fdn>tpgy-mgr:name-igp1-AS-0:router-595137581</fdn>
                     <egressEdges>
                        <topology.LinkEdge>
                           <destFdn>tpgy-mgr:name-igp1-AS-0:router-595137583</
destFdn>
                           <fdn>tpgy-mgr:name-igp1-AS-0:link-start-tpgy-mgr%name
-igp1-AS-0%router-595137581-ip-10.45.47.1-if-0-pointToPoint-to-tpgy-mgr%name-igp1-
AS-0%router-595137583</fdn>
                        </topology.LinkEdge>
                     </egressEdges>
                  </topology.RouterVertex>
                  <topology.RouterVertex>
                     <fdn>tpgy-mgr:name-igp1-AS-0:router-595137583</fdn>
                     <egressEdges/>
                  </topology.RouterVertex>
               </vertices>
               <detailedErrors/>
            </topology.RouteResult>
         </value>
      </item>
   </result>
</topology.RouteManager.retrieveRoutesResponse>
```

21.7 Path and prefix monitoring

21.7.1 Overview

OSS applications that are integrated with the CPAM can be used to monitor IGP paths between any two routers or BGP prefixes available to the CPAM. When a network topology changes, such as a link metric or state change, the system evaluates whether the routes of any registered path are affected. In this case, new routes are recorded and the OSS clients are immediately informed. If there is no route for a monitored path as a result of a topology change, a record is logged.

The CPAM supports the following types of path monitoring:

- **IP network path monitoring**
  Monitors the IP forwarding path between two system IP addresses and supports both unidirectional and bidirectional IP path monitoring within a single IGP administrative domain.
- **LSP path monitoring**
  Monitors the path (active, secondary, and primary) of a specified LSP, and supports both unidirectional and bidirectional LSP path monitoring.
- **P2MP LSP path monitoring**
  Monitors the path (active, secondary, and primary) of a P2MP LSP.
- **BGP monitored prefix**
  Monitors BGP IPv4, IPv6, VPN-IPv4, and VPN-IPv6 prefixes visible to CPAM.

See “Path and prefix monitoring” the *NSP NFM-P Control Plane Assurance Manager User Guide* for more information about path monitoring.

21.7.2 Path monitoring workflow

The following workflows describe how to configure and view IP and LSP path monitors.

For an IP path monitor:

1. Create an IP path monitor between two routers. The monitor can be a unidirectional or bidirectional monitor.
2. Monitor the IP network path.
3. View any changes on the path.

For a LSP path monitor:

1. Create a LSP path monitor from an NFM-P managed dynamic LSP. It can be a unidirectional or bidirectional monitor.
2. Monitor the LSP path.
3. View any changes on the path.

For a BGP monitored prefix:

1. Create a BGP monitored prefix for a supported BGP prefix.
2. Monitor the BGP prefix.
3. View any change of BGP prefix status.

21.7.3 Packages and classes

The monpath package is used to monitor changes to the paths of IP routes, LSPs, and the topology package used to monitor BGP prefixes. The changes are recorded, as they occur, by CPAM IP and LSP path monitors, and BGP monitored prefixes. The following table shows the classes for configuring and retrieving monitored objects.

Table 21-6 Monitor objects

Object	Package and class
IP network path monitoring	
Unidirectional IP path monitor	monpath.MonitoredIpPath
Bidirectional IP path monitor	monpath.BidirMonitoredIpPath
IP path record - instance of a path route at a specific instance	monpath.IpPathRecord
IP path segment - a hop from one router to another	monpath.IpPathSegment
LSP path monitoring	
Unidirectional LSP path monitor	monpath.MonitoredLspPath
Bidirectional LSP path monitor	monpath.BidirMonitoredLspPath
LSP path record - instance of a path route at a specific instance	monpath.LspPathRecord
LSP path segment - a hop from one router to another	monpath.LspPathSegment
P2MP LSP path monitoring	
P2MP LSP path monitor	monpath.MonitoredP2MPLspPath
P2MP S2L path monitor	monpath.MonitoredS2LPath
BGP monitored prefix	
BGP monitored prefix	topology.BgpMonitoredPrefix
BGP monitored Prefix status record	topology.BgpMonPrefixStatus

21.7.4 Path and prefix monitoring creation

The XML API supports the creation, deletion, and modification of IP and LSP path monitors, and BGP monitored prefixes. The following example shows the minimum XML parameter tags required to develop an XML request to create a unidirectional IP path monitor.

The generic.GenericObject.configureChildInstance method is used to create an IP path monitor. The request response returns the <objectFullName> attribute to identify the new IP path monitor.

`IP path monitor creation parameters`

- method—generic.GenericObject.configureChildInstance
- distinguishedName—monitored-path-mgr. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference. In this case, it is the monpath.MonitoredPathManager class.

- childConfigInfo—creates and configures the following parameters:
  - IP path monitor object - monpath.MonitoredIpPath
  - actionMask - specifies the create operation
  - <source> - source address of the monitored path
  - <dest> - destination address of the monitored path
  - <administrativeState> - enables or disables the path monitoring

Figure 21-15 Unidirectional IP path monitor creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>monitored-path-mgr</distinguishedName>
   <childConfigInfo>
      <monpath.MonitoredIpPath>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <source>198.51.100.36</source>
         <dest>198.51.100.35</dest>
         <administrativeState>up</administrativeState>
      </monpath.MonitoredIpPath>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

LSP path monitors can also be created using the monpath.MonitoredPathManager.createMonitoredPath method. This method can be used to create multiple monitored paths from the passed-in LSPs or service tunnels.

LSP path monitor creation using createMonitoredPath method

- method—monpath.MonitoredPathManager.createMonitoredPath
- <sourceObjectNameSet>—the set of LSP or service tunnel object pointers to create monitored LSP paths from.
  - <pointer> - FDN of a dynamic LSP or service tunnel - add one for each LSP or service tunnel
- <isBidirectional>—whether to create bidirectional monitored paths - true or false
- <pathType>—sets the path types (active, primary, secondary). Set all applicable bits.
  - <bit> - monpath.PathTypeBitMask bitmask value

Figure 21-16 Multiple LSP path monitor creation request example

```
<monpath.MonitoredPathManager.createMonitoredPath>
   <deployer>immediate</deployer>
   <sourceObjectNameSet>
      <pointer>lsp:from-198.51.100.35-id-1</pointer>
      <pointer>lsp:from-198.51.100.32-id-4</pointer>
   </sourceObjectNameSet>
   <isBidirectional>false</isBidirectional>
   <pathType>
      <bit>active</bit>
```

```
   </pathType>
</monpath.MonitoredPathManager.createMonitoredPath>
```

21.7.5 Path monitoring methods

Path monitoring can be manually triggered by using the captureCurrentPath method. The method captures the state of the path at the moment the method is executed.

Each of the classes that can be used to create an IP or LSP path monitor has its own captureCurrentPath method. For example, the captureCurrentPath method for a unidirectional IP path monitor is under the monpath.MonitoredIpPath called monpath.MonitoredIpPath.captureCurrentPath.

Capture current path method

- method—monpath.MonitoredIpPath.captureCurrentPath
- instanceFullName—monitored-path-mgr:ip-path-src-${*sourceType*}-${*source*}-${*sourceLen*}-dest-${*destType*}-${*dest*}-${*destLen*}. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference. In this case, it is the monpath.MonitoredIpPath class.

Figure 21-17 Current unidirection IP path capture request example

```
<monpath.MonitoredIpPath.captureCurrentPath xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <instanceFullName>monitored-path-mgr:ip-path-src-ipv4-198.51.100.36-32-dest-ipv4-
198.51.100.35-32</instanceFullName>
</monpath.MonitoredIpPath.captureCurrentPath>
```

See the XML API Reference for descriptions of the input parameters.

21.7.6 Monitored path information retrieval

The find or findToFile method, along with the appropriate class for path record and segment specified in Table 21-6, “Monitor objects” (p. 315) , and a filter, can be used to retrieve path monitor data.

Retrieving path record data

- method—find
- fullClassName—the class from where data is to be retrieved. For Figure 21-18, “IP path record retrieval request example” (p. 317) , the class is the monpath.IpPathRecord class.
- filter—sets a filter to only return IP path record objects that satisfy the filter criteria
  - The filter criteria is to look for source of 198.51.100.36, and destination of 198.51.100.37.

The following figure shows an example of a request to find IP path records that match a filter.

Figure 21-18 IP path record retrieval request example

```
<find xmlns="xmlapi_1.0">
   <fullClassName>monpath.IpPathRecord</fullClassName>
   <filter>
      <and>
```

```
         <equal name="source" value="198.51.100.36"/>
         <equal name="dest" value="198.51.100.37"/>
      </and>
   </filter>
</find>
```

See Chapter 13, “Inventory management” for more information about using the find and findToFile method and filters.

21.7.7 BGP monitored prefix

The following figure shows an example XML request to create a BGP monitored IPv4 prefix.

Figure 21-19 BGP monitored IPv4 prefix creation request example

```
<generic.GenericObject.configureChildInstance>
   <deployer>immediate</deployer>
      <synchronousDeploy>true</synchronousDeploy>
      <deployRetries>1</deployRetries>
      <clearOnDeployFailure>true</clearOnDeployFailure>
      <distinguishedName>tpgy-mgr</distinguishedName>
      <childConfigInfo>
      <topology.BgpMonitoredPrefix>
      <actionMask>
      <bit>create</bit>
      </actionMask>
      <bgpASPointer>tpgy-mgr:AS-50:AS-65050</bgpASPointer>
      <prefRd></prefRd>
      <displayedName></displayedName>
      <description></description>
      <alarmSuppress>
      <bit>asPath</bit>
      <bit>redLoss</bit>
      <bit>unreachable</bit>
      </alarmSuppress>
      <prefAddr>2.0.0.0</prefAddr>
      <asPathLenThreshold>1</asPathLenThreshold>
      <prefAddrType>ipv4</prefAddrType>
      <prefLen>32</prefLen>
      <alarmThreOverride>
      <bit>asPath</bit>
      <bit>redLoss</bit>
      </alarmThreOverride>
      <redLossThreshold>1</redLossThreshold>
      <administrativeState>up</administrativeState>
      <prefType>ipv4</prefType>
      <rdType>none</rdType>
      </topology.BgpMonitoredPrefix>
      </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

The following xml tags describe the inputs to this request:

- distinguishedName - CPAM AS Object FDN
- bgpASPointer - CPAM AS object FDN
- prefRD - route-distinguisher monitored Prefix for VPN-IPv4, VPN-IPv6
- displayedName - displayed name of monitored prefix
- description - description of monitored prefix
- prefAddr - IP address of the monitored Prefix
- prefAddrType - Prefix address type ipv4, ipv6
- prefLen - Prefix length
- prefType - BGP prefix type ipv4,ipv6,vpnIpv4 or vpnIpv6
- administrativeState - up or down
- rdType - BGP prefix route distinguisher type none (IPv4,IPv6) or type0,type1 or type2 for VPN-IPv4, VPN-IPv6
- alarmSupress (optional) - if BGP monitored Prefix Alarms are configured, the user can supress the BGP monitored prefix alarms specified by the bit flag:
  - aspath - supress BGP monitored prefix alarm topology.BgpAsPathLenPerMonPrefThresholdReached
  - redloss - supress BGP monitored prefix alarm topology.BgpMonPrefRedundancyLossThresholdReached
  - unreachable - supress BGP monitored prefix alarm topology.BgpMonPrefixUnreachable
- alarmThreOverride (optional) - for given BGP monitored Prefix, override the following alarms by set by bit flags of value:
  - asPath - override alarm topology.BgpAsPathLenPerMonPrefThresholdReached
  - redLoss - override alarm topology.BgpMonPrefRedundancyLossThresholdReached
- redLossThreshold (optional) - if the alarmThreOverride with bit flag redLoss is set, the user can specify the threshold override value for alarm topology.BgpMonPrefRedundancyLossThresholdReached
- asPathLenThreshold (optional) - if the alarmThreOverride with bit flag asPath is set, the user can specify the threshold override value for alarm topology.BgpAsPathLenPerMonPrefThresholdReached

If the user wants to manually capture a monitored prefix status record, the following XML request can be used:

Figure 21-20 Manually capture a monitored prefix status record request example

```
<topology.BgpMonitoredPrefix.captureCurrentPrefix xmlns="xmlapi_1.0">
   <deployer>immediate</deployer><!--Type:deployer-->
   <synchronousDeploy>true</synchronousDeploy><!--Type:synchronousDeploy-->
   <clearOnDeployFailure>true</clearOnDeployFailure><!--Type:clearOnDeployFailure-->

   <deployRetries>1</deployRetries><!--Type:deployRetries-->
   <deployRetryInterval>60</deployRetryInterval><!--Type:deployRetryInterval-->
   <taskDescription></taskDescription><!--Type:taskDescription-->
```

```
   <instanceFullName>tpgy-mgr:AS-50:AS-65050:type-ipv4-rdtype-none-rd--addr-2.0.0.
0-len-32</instanceFullName><!--Type:instanceFullName-->
</topology.BgpMonitoredPrefix.captureCurrentPrefix>
```

- instanceFullName - monitored Prefix FDN

21.8 Fault management

21.8.1 Overview

The fault management system that the CPAM provides includes:

- RCA audit policy for identifying problems or errors in protocol-related and control plane configuration
- CPAM alarms

21.8.2 RCA audit policies

The CPAM allows you to perform on-demand verifications of the IP/MPLS configuration of IS-IS and OSPF routing protocols on NFM-P-managed NEs. RCA audit policies identify problems or errors in protocol-related and control plane configurations.

RCA audit polices can be created for IGP administrative domains to verify the configuration of network objects on NFM-P-managed routers. For each audit policy created, one or more policy entries can be specified to define the scope of configuration that is verified by the audit and the severity of the related problem.

See “RCA audit policies” in the *NSP NFM-P Control Plane Assurance Manager User Guide* for more information about RCA audit policies.

21.8.3 RCA audit policies workflow

The following workflow provides how to create and use RCA audit policies.

1

Create an RCA audit policy. Configure the attributes that are verified for each RCA audit policy entry.

2

Run the RCA audit policy on a specific object.

3

Identify the problems and correct the configuration.

21.8.4 RCA audit policies creation

The rca package is used for policy-driven IP/MPLS configuration audits for OSPF and IS-IS routing protocols.

When creating an RCA audit policy without specifying any audit policy entry, by default, all entries are created and enabled for an RCA audit. If you want to only audit specific attributes, you would need to create audit policy entries for those attributes.

The following example shows the minimum XML parameter tags required to develop an XML request to create an OSPF RCA audit policy with an audit policy entry to audit only the MTU attribute of OSPF interfaces.

The generic.GenericObject.configureChildInstance method is used to create an RCA audit policy. The request response returns the <objectFullName> attribute to identify the new RCA audit policy.

RCA audit policy creation parameters

- method—generic.GenericObject.configureChildInstance
- distinguishedName—rca-mgr. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference. In this case, it is the rca.RcaManager class.
- childConfigInfo—creates and configures the following parameters:
  - RCA audit policy object- rca.AuditPolicy
  - actionMask - specifies the create operation
  - <id> - (optional) assigns a unique numeric ID. If not specified, the NFM-P automatically assigns the ID.
  - <policyName> - ospf for OSPF type RCA audit policy or isis for IS-IS type RCA audit policy
  - <children-Set>—tag to enclose children objects
    - RCA audit policy entry object - rca.AuditPolicyEntry
    - actionMask - specifies the modify operation
    - <objectFullName> - rca-mgr:ID-*${*id}*:enId-*${*entryId}* - FDN of the RCA audit policy entry object
    - <mismatchAttributes>—set of attributes to configure to enable or disable audit for and the severity of the related problem.
      - Mismatch attribute set - rca.MismatchAttribute
        - attributeName - OSPF interface attributes. See ospf.Interface class in the XML API Reference.
        - severity - sets severity, such as info, minor, major, critical, etc. See the XML API Reference for values.
        - enabled - true or false

Figure 21-21 OSPF RCA audit policy creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>rca-mgr</distinguishedName>
   <childConfigInfo>
      <rca.AuditPolicy>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <id>100</id>
         <policyName>ospf</policyName>
```

```
         <children-Set>
            <rca.AuditPolicyEntry>
               <actionMask>
                  <bit>modify</bit>
               </actionMask>
               <objectFullName>rca-mgr:ID-100:enId-1</objectFullName>
               <mismatchAttributes>
                  <rca.MismatchAttribute>
                     <attributeName>mtu</attributeName>
                     <severity>critical</severity>
                     <enabled>true</enabled>
                  </rca.MismatchAttribute>
               </mismatchAttributes>
            </rca.AuditPolicyEntry>
         </children-Set>
      </rca.AuditPolicy>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

21.8.5 RCA audit policies configuration check method

RCA audits can be manually triggered by using the rca.RcaManager.checkConfig method. The method validates the configuration of OSPF or IS-IS routing protocols set by the RCA audit policy.

Check config method

- method—rca.RcaManager.checkConfig
- <fdn>—tpgy-mgr:name-${*displayedName*}-AS-${*asNumber*}:protocol-${*protocol*}-area-${*areaId*}, which is a pointer to the OSPF area to perform the check configuration for. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference. In this case, it is the topology.Area class.
- <fdnPolicy>—rca-mgr:ID-${**Id*}, which is a pointer to the RCA audit policy. In this case, it is the rca.AuditPolicy class.
- <aInContext>—specifies the rca.Context. The value for this context is null for an OSPF or IS-IS RCA audit, but the tags need to be specified to make the request valid.
- <resultFilter>—(optional) filter for reducing the scope of the returned information

Figure 21-22 RCA audit execution request example

```
<rca.RcaManager.checkConfig xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <fdn>tpgy-mgr:name-AdminDomain1-AS-1:protocol-ospf-area-0.0.0.0</fdn>
   <fdnPolicy>rca-mgr:ID-100</fdnPolicy>
   <aInContext>
      <rca.Context/>
   </aInContext>
   <resultFilter/>
</rca.RcaManager.checkConfig>
```

If there are problems with the configuration, the response is similar to the example in the following figure.

Figure 21-23 RCA audit problem response example

```
<rca.RcaManager.checkConfigResponse xmlns="xmlapi_1.0">
   <problemList>
      <rca.Problem>
         <lastTimeChanged>1344305122231</lastTimeChanged>
         <id>notFound</id>
         <causedByObjectFullNames>
            <pointer>network:198.51.100.32:router-1:ospf-v2:areaSite-0.0.0.0:interface-
5</pointer>
         </causedByObjectFullNames>
         <isFixable>false</isFixable>
         <severity>indeterminate</severity>
         <probableCause>unknown</probableCause>
         <description>Far-End Not found</description>
         <ignoreFix>false</ignoreFix>
         <applicationEnum>unspecified</applicationEnum>
         <solution/>
         <policyFdn>rca-mgr:ID-100</policyFdn>
         <deploymentState>0</deploymentState>
         <objectFullName>network:198.51.100.32:router-1:ospf-v2:areaSite-0.0.0.0:
interface-5:cause-unknown-notFound</objectFullName>
         <name>cause-unknown-notFound</name>
         <selfAlarmed>false</selfAlarmed>
         <children-Set/>
      </rca.Problem>

      <rca.Problem>
         <lastTimeChanged>1344305122232</lastTimeChanged>
         <id>aggregated</id>
         <causedByObjectFullNames>
            <pointer>ospf:area-0.0.0.0-version-2</pointer>
         </causedByObjectFullNames>
         <isFixable>false</isFixable>
         <severity>indeterminate</severity>
         <probableCause>aggregatedCause</probableCause>
         <description>Aggregated Cause</description>
         <ignoreFix>false</ignoreFix>
         <applicationEnum>unspecified</applicationEnum>
         <solution/>
         <policyFdn>rca-mgr:ID-100</policyFdn>
         <deploymentState>0</deploymentState>
         <objectFullName>ospf:area-0.0.0.0-version-2:cause-aggregatedCause-aggregated
</objectFullName>
         <name>cause-aggregatedCause-aggregated</name>
         <selfAlarmed>false</selfAlarmed>
```

```
        <children-Set/>
      </rca.Problem>
   </problemList>
</rca.RcaManager.checkConfigResponse>
```

21.8.6 CPAM alarms

In addition to routing alarms generated by routers, there are threshold reaching alarms generated by the 7701 CPAA. The alarm generation process uses the routing data collected by the 7701 CPAA. The generated alarms are sent to the CPAM route controller. The alarms are then available to an OSS application in the same manner that alarms generated in the NFM-P are sent to the XML API clients.

See Chapter 11, “Fault management” for more information about retrieving alarms using the OSS interface.

All CPAM-related JMS event messages are part of the CPAM event category, for example, ALA_category = CPAM. When retrieving CPAM threshold alarms via JMS, set ALA_category=CPAM as part of the filter.

The following are not included in the CPAM category:

- Configuration changes
  - interfaces
  - protocol
  - hardware (cards, daughter cards, ports)
  - MPLS
  - LDP

See “Threshold reaching alarms” in the *NSP NFM-P Control Plane Assurance Manager User Guide* for more information about the CPAM threshold reaching alarms for BGP, IS-IS, and OSPF.

The following table shows some of the CPAM alarms threshold package and class for setting alarm threshold attributes for the specific alarm. This list is not complete. See the XML API Reference under the topology package for a complete list.

Table 21-7 CPAM alarm threshold objects

Routing alarm type	Package and class
ISIS LSP Alarm Threshold	topology.IsisLspAlarmThreshold
ISIS Reachability Threshold	topology.IsisReachabilityAlarmThreshold
OSPF LSA Rate Alarm Threshold	topology.OspfLsaRatePerRouterAlarmThreshold
OSPF LSA Alarm Threshold	topology.OspfLsaPerRouterAlarmThreshold
BGP Route Rate Threshold Per RT Threshold	topology.BgpRouteRateThresholdPerRTargetAlarmThreshold
BGP AS Path Length Per Monitored Prefix Threshold	topology.BgpAsPathLenPerMonPrefAlarmThreshold
BGP Packet Rate Threshold	topology.BgpPktRateAlarmThreshold
BGP Route Count Threshold	topology.BgpRouteCountAlarmThreshold

Table 21-7 CPAM alarm threshold objects (continued)

Routing alarm type	Package and class
BGP Route Flap Rate Threshold	topology.BgpRouteFlapRateAlarmThreshold
BGP Route Rate Threshold	topology.BgpRouteRateAlarmThreshold
BGP Monitored Prefix Flap Rate Threshold Reached	topology.BgpMonitorPrefixFlapRateThresholdReached
BGP monitored Prefix Unreachable	topology.BgpMonPrefixUnreachable
BGP Prefix Monitor Redundancy Loss	topology.BgpMonPrefRedundancyLossThresholdReached

The following are the mandatory XML parameter tags required to develop an XML request to create and set a BGP alarm threshold.

The generic.GenericObject.configureChildInstance method is used to create a BGP route count alarm threshold. The request response returns the <objectFullName> attribute to identify an alarm threshold.

Alarm threshold creation parameters

- method—generic.GenericObject.configureChildInstance
- distinguishedName—tpgy-rtr-alarm-mgr. The FDN can be determined by looking at the Parent Hierarchy of the class to be created in the XML API Reference. In this case, it is the topology.RoutingAlarmManager class.
- childConfigInfo—creates and configures the following parameters:
  - BGP route count alarm threshold object - topology.BgpRouteCountAlarmThreshold
  - actionMask - specifies the create operation
  - <cpaaPointer> - network:${*systemAddress*}:cpaa - pointer to the 7701 CPAA
  - <alarmName> - sets the alarm name for this threshold. In Figure 21-24, “BGP route count alarm threshold creation request example” (p. 325) , the alamr name is the BgpRouteCountThresholdReached alarm. See the XML API Reference under the fm.alarmName type for a complete list of alarm names. Search for package=topology for CPAM alarms.
  - <threshold> - sets the threshold
  - <administrativeState> - enables or disables monitoring of alarm threshold

Figure 21-24 BGP route count alarm threshold creation request example

```
<generic.GenericObject.configureChildInstance xmlns="xmlapi_1.0">
   <deployer>immediate</deployer>
   <distinguishedName>tpgy-rtr-alarm-mgr</distinguishedName>
   <childConfigInfo>
      <topology.BgpRouteCountAlarmThreshold>
         <actionMask>
            <bit>create</bit>
         </actionMask>
         <cpaaPointer>network:198.51.100.37:cpaa</cpaaPointer>
         <alarmName>topology.BgpRouteCountThresholdReached</alarmName>
         <threshold>1000</threshold>
```

```
         <administrativeState>up</administrativeState>
      </topology.BgpRouteCountAlarmThreshold>
   </childConfigInfo>
</generic.GenericObject.configureChildInstance>
```

A OSS developer best practices

A.1 Recommendations

A.1.1 Communicating with the NFM-P

Nokia recommends that an OSS use the JMS event stream for ongoing monitoring of alarms and network objects, and the XML API only for on-demand information retrieval when synchronization is required. See Chapter 4, “Event monitoring using JMS” and Chapter 5, “XML requests” for more information about JMS and HTTP communication with the NFM-P.

A.1.2 Time synchronization

Nokia strongly recommends that the time between an NFM-P main server and the OSS clients is synchronized using a timing synchronization protocol such as NTP for the following reasons:

- To facilitate the troubleshooting of OSS client and NFM-P interaction
- To prevent the JMS consumer on the OSS client from discarding keep-alive messages

A.1.3 Data retrieval using filters and resultFilter elements

Nokia recommends using the filter and resultFilter elements to reduce the number of objects and object properties that a request returns, and to speed data retrieval. See the following for information about filters and resultFilters:

- A.3.5 “Inventory retrieval” (p. 331) in A.3 “HTTP communication” (p. 328)
- A.4 “JMS communication” (p. 332)
- Chapter 4, “Event monitoring using JMS” , Chapter 5, “XML requests” , Chapter 11, “Fault management” , Chapter 13, “Inventory management” , and Chapter 14, “Accounting, performance, and flow monitoring”

A.2 Recommended durable JMS client operation

A.2.1 Steps

1

Establish a JMS connection and subscription with the NFM-P; listen to JMS events, and buffer events locally. Nokia recommends that the OSS maintain two local buffer queues:

- One queue for critical system-related events that the OSS can use to monitor the state of the JMS connection. Examples of critical system-related events includes the KeepAliveEvent, SystemInfoEvent, JmsMissedEvent, and TerminateClientSessionEvent.
- One queue for inventory or alarm events in which the OSS is interested.

Note: The OSS must not begin processing the events from the second buffer until Step 3 is successfully completed.

2

Initiate retrieval of inventory, the latest alarm list, or statistics and OAM test results using HTTP(S), or initiate retrieval of statistics and OAM log files via (S)FTP if any of the following occurs:

- The IP address of the Primary NFM-P Server in the SystemInfoEvent is different from that of the last known Primary NFM-P Server.
- The jmsStartTime or sysStartTime in the SystemInfoEvent are different from the last-recorded jmsStartTime and sysStartTime.
- A JmsMissedEvent notification is received.

Note: While inventory is being retrieved, the OSS must continue to monitor the first buffer queue, as described in Step 1 . If the OSS detects connectivity loss with the NFM-P, or a loss of JMS events, the OSS should terminate the current inventory retrieval process, if possible. The internal buffer queues should be cleared. Perform exit procedures to gracefully unsubscribe and disconnect from the XML API. Then repeat the start-up procedures from Step 1 .

3

After the inventory retrieval completes successfully, start listening to the events from the second buffer queue described in Step 1 . Continue to monitor the first buffer queue.

4

If the OSS has to restart, the OSS should perform exit procedures to gracefully unsubscribe and disconnect from the XML API, then repeat the start-up procedures from Step 1 .

The OSS may need to restart if the OSS detects one of the following:

- Messages have been lost (JmsMissedEvents).
- The connection to the NFM-P fails (JMSonException).
- There is silence on the JMS channel for an unreasonable time period, which may indicate a loss of communication with the NFM-P.

See A.4.3 “Monitoring a JMS connection” (p. 332) in A.4 “JMS communication” (p. 332) for more information about JMS connection monitoring.

END OF STEPS

A.3 HTTP communication

A.3.1 Connections

HTTP connection limit

The XML API supports a limited number of concurrent HTTP connections. The exact limit depends on the available system resources. See the *NSP Planning Guide* for the platform requirements regarding the HTTP connection limit.

The OSS requests of a specific NFM-P user have a priority that is assigned during user or user group configuration. The prioritization may cause some lower-priority requests to time out in a high-volume environment. Nokia recommends that you design each OSS application to include a configurable request duration timeout in order to avoid request timeouts.

Nokia recommends that only one HTTP connection is used for XML SOAP requests from each OSS. If more than one HTTP connection is required, the number of concurrent HTTP connections must be configurable in the OSS application so that system integrators can manage the HTTP connections across multiple applications.

HTTP security

The NFM-P supports both secure and non-secure communication over HTTP. Nokia recommends that an OSS application support both protocols.

A.3.2 Sending SOAP requests

Inventory commands

The NFM-P can concurrently execute a maximum of five concurrent find and findToFile OSS requests. An OSS can potentially consume all available HTTP connections if the number of simultaneous inventory requests is not configurable. When an OSS must send simultaneous inventory requests, Nokia recommends that the number be configurable at run time.

Streaming SOAP XML commands

An efficient way to perform multiple XML API requests is to use SOAP streaming. This eliminates the overhead of establishing HTTP connections for every XML API request.

In SOAP streaming, the OSS client first opens a HTTP connection, encapsulates the XML API command into a SOAP envelope, and posts the request to the XML API. After the NFM-P server response is received by the OSS client and while keeping the connection open, the next request, encapsulated into a SOAP envelope, can be posted using the same HTTP connection. See Chapter 5, “XML requests” and Chapter 9, “XML message structure” for more information about HTTP connection and XML message structures.

Nokia recommends that an OSS client use SOAP streaming. An OSS client using SOAP streaming should implement a configurable idle timer to close the HTTP connection when not in use, to reduce the number of consumed HTTP connections.

Avoid configuration with children hierarchy

It is more difficult to troubleshoot a failure that follows an XML API request with multiple configuration operations in comparison to an XML API request with a single configuration operation. To execute corrective action, for example to reverse a partial configuration, it is important to first precisely identify which part of the configuration request failed.

Nokia recommends that each configuration request contain no more than one configuration operation. For example, to configure a service, each service object should be configured individually in its own XML API request, as opposed to configuring all service objects in a single, complex XML API request.

Avoid configuration with children hierarchy

It is more difficult to troubleshoot a failure that follows an XML API request with multiple configuration operations in comparison to an XML API request with a single configuration operation. To execute corrective action, for example to reverse a partial configuration, it is important to first precisely identify which part of the configuration request failed.

Nokia recommends that each configuration request contain no more than one configuration operation. For example, to configure a service, each service object should be configured individually in its own XML API request, as opposed to configuring all service objects in a single, complex XML API request.

Response processing

Depending on the network and the XML request, the size of the XML response may be quite large. Nokia recommends that OSS software process the XML response as it is streamed. This avoids the need to buffer the entire XML response and the OSS software is able to process the response without having to wait for the entire response to be generated.

A.3.3 Monitoring the NFM-P main server

Nokia recommends that you use JMS to monitor the NFM-P; see Chapter 4, “Event monitoring using JMS” for more information.

Nokia also recommends that you monitor the NFM-P HTTP and SOAP responses to ensure that the requests are correctly processed. The following table lists and describes the response error messages.

Table A-1 HTTP error messages

Error message	Recommended action
Connection refused	The NFM-P is not fully initialized; try again later.
HTTP code 405	The standby main server has received the request; redirect the request to the primary main server.
HTTP code 905	The maximum number of concurrent OSS sessions for one user is reached; try again later.
HTTP code 503	The HTTP connection limit is reached; try again later.

A.3.4 Failure handling

An OSS may or may not be designed to perform automatic rollback from failure. Nokia recommends that the OSS application should log the failure details for reference, which can be used later for manual troubleshooting and error recovery. Failed deployments must be cleared because uncleared deployments consume resources on the NFM-P server. The total number of deployments on the NFM-P server is finite. If all of the deployments are being consumed on the NFM-P server, any future network configurations are blocked.

Nokia recommends the following deployment-handling strategy:

- Do not change the default value (false) of the <clearOnDeployFailure> parameter. The OSS should retrieve the failure details from the deployment (use the

generic.GenericObject.getDeployer(s) method), then immediately delete the deployment (use the generic.GenericObject.clearDeployer method).

- Do not change the default value (0) of the <deployRetries> and <deployRetryInterval> parameters.

See 15.3 “Deployments” (p. 209) for more information about deployments and deployment failure handling procedures.

A.3.5 Inventory retrieval

When obtaining a full inventory from the NFM-P, all objects of a specified class should be retrieved in a single request. Requests should not be hierarchical. For example, children objects should be retrieved in separate requests from the parent. Filters and result filters can be used to achieve these goals.

Filters

When only a subset of a particular object class is required, filters can be used to limit the objects being returned. Filters should only specify attributes that are part of the object being requested. Filters that depend on attributes of children classes should be avoided.

Result filters

Result filters should be used to limit the attributes that are returned. Only those attributes that are of interest to the OSS should be included in the filter. To uniquely identify objects at a later time, the fully distinguished name (<objectFullName>), which is guaranteed unique, should be used.

find and findToFile methods

The find and findToFile methods can be used to retrieve an inventory of general objects. Table A-2, “find and findToFile comparison” (p. 331) lists the advantages and disadvantages of each method.

Note: Nokia does not recommend making find or findToFile requests on a large scale network, compared to a smaller scope, because requests could take longer for larger data sets. Frequent large find or findToFile operations require more system resource so these requests may need to be timed for appropriate daily windows. If an application needs updates more frequently, event notification can be used by applications without extracting full inventory.

Note: Alarm retrieval requires the findFaults method.

Table A-2 find and findToFile comparison

Method	Advantages	Disadvantages
find	• Easy to send requests and parse results using HTTP • Results can be parsed as they become available	• Subject to failure due to network conditions (latency, dropped packets)

Table A-2 find and findToFile comparison (continued)

Method	Advantages	Disadvantages
findToFile	• Can resume file transfer after network interruptions • Does not require HTTP connection to remain open (requires asynchronous requests used in conjunction with JMS notifications)	• Requires waiting for entire response before results can be parsed

A.4 JMS communication

A.4.1 JMS connection limit

The NFM-P supports a maximum of 10 or 20 concurrent JMS connections, depending on the available system resources. See the *NSP Planning Guide* for the platform sizing requirements regarding the JMS connection limit.

A JMS connection attempt when the limit is reached generates the following exception:

```
Exception: XML API Maximum Connected Clients
```

Nokia recommends that an OSS should use only one JMS connection per JMS topic. When an OSS client needs to receive events from multiple JMS topics, Nokia recommends subscribing to the 5620-SAM-topic-xml-filtered topic, which supports advanced message filtering and allows an OSS to receive any required events. See 4.6 “JMS message filtering” (p. 37) for information about advanced message filtering.

A.4.2 JMS filtering

To reduce the JMS event traffic volume, an OSS must use a JMS filter to specify the events that the OSS requires. A JMS filter must allow critical system events to pass. When an OSS client subscribes to the 5620-SAM-topic-xml-filtered topic, the client can use an advanced message filter to specify event messages. See 4.6 “JMS message filtering” (p. 37) for information about advanced message filtering.

A.4.3 Monitoring a JMS connection

To ensure that the JMS connection is active, all incoming events including the following events must be monitored:

- KeepAliveEvent
- TerminateClientSessionEvent
- StateChangeEvent (JMSMissedEvent and SystemInfoEvent)

If no events are received for two minutes, the status of the NFM-P server must be checked.

See 4.8 “Connection monitoring and error recovery” (p. 57) for more information.

A.4.4 JMS acknowledgment mode

The NFM-P supports the following acknowledgment modes:

- AUTO_ACKNOWLEDGE
- DUPS_OK_ACKNOWLEDGE

Nokia recommends that OSS clients use DUPS_OK_ACKNOWLEDGE mode in a network environment with high latency to improve JMS message throughput. OSS clients using this mode must be able to handle the potential of duplicate JMS messages. See 4.8 “Connection monitoring and error recovery” (p. 57) for more information.

A.4.5 Use the 5620-SAM-topic-xml-filtered JMS topic

Nokia recommends that OSS clients use the 5620-SAM-topic-xml-filtered topic for the following benefit:

- The advance message filter selects messages using the JMS header field as well as the message body and object properties.
- The result filter selects only the required attributes to be sent to OSS clients.
- OSS clients can adjust the advance JMS filter dynamically without disconnecting the JMS connection.
- Multiple events are sent using the event vessel message to enhance events delivery throughput.

A.4.6 Advanced message filtering

The JMS 5620-SAM-topic-xml-filtered topic supports advanced message filtering. Nokia recommends the following:

- In the advanced message filter, the “classname” should be included in the extra tag to facilitate processing of attribute value change events.
- Because message filter elements are limited, consider the “in” filter element to eliminate the need for using logical “or” with numerous leaf filter elements.

A.5 Statistics data retrieval with registerLogToFile and find/findToFile

A.5.1 Overview

It is recommended to use the registerLogToFile method for ongoing statistics retrieval, especially in an environment where a large number of statistics are to be collected; the find and findToFile methods are not recommended in networks with high statistics collection rates. See Chapter 14, “Accounting, performance, and flow monitoring” and the *NSP Planning Guide* for information about statistics collection using the XML API. See 14.8 “Statistics retrieval using registerLogToFile” (p. 183) for specific information about using the registerLogToFile method.

An accounting or performance statistics file is specific to a statistics class, and is stored in a specified directory on an NFM-P server. When a file is created and ready for retrieval, the NFM-P sends a LogFileAvailableEvent. A file generated using the method is deleted after 20 minutes by default. You must ensure that adequate file space is available for the generated files, which may be deleted earlier if space is not available.

By default, JMS client inactivity checking is enabled, and a registration is removed after 15 minutes if the associated JMS client is disconnected. If a prolonged OSS outage is expected, you can modify the client timeout accordingly. For example, before an OSS upgrade, it is recommended to

increase the timeout so that statistics files continue to be generated during the outage associated with the upgrade. See 14.8 “Statistics retrieval using registerLogToFile” (p. 183) for configuration information.

A JMS subscription is optional when using the registerLogToFile method. To avoid registerLogToFile client deregistration because of JMS client inactivity, you can disable the JMS client inactivity check. In such a scenario, the registering application is responsible for ensuring that JMS clients are deregistered as required in order to allow the NFM-P to reclaim resources. See A.6 “To configure the registerLogToFile or registerSasLogToFile client inactivity check” (p. 334) for configuration information.

Note: The find and findToFile methods are not recommended as a means to recover from prolonged outages.

The find and findToFile methods can be used to retrieve specific statistics on demand, for example, during troubleshooting when statistics matching a set of criteria from a certain time are required. It is recommended that OSS clients avoid invoking find/findToFile frequently without filters or result filters.

The find and findToFile methods are subject to performance limitations in NFM-P deployments that include an auxiliary database. Specifically, if the results contain data from children classes, responses can take up to 20 times longer than in a similarly sized NFM-P environment that does not include an auxiliary database.

Delays may be minimized in the following ways:

- Use the <children/> resultFilter expression to explicitly exclude child class data, which is often not the target of interest.
- Create request filters that reduce the overall number of objects returned. For Statistics and OAM test results, use timeCaptured to limit the amount of logRecords for retrieval. Multiple timestamps are provided by statistic log records. The NFM-P optimizes the filtering based on timeCaptured, so timeCaptured should be used for find and findToFile queries filtering on time.
- Create requests that target the child class directly, where data may be retrieved without explicit association to the parent object.

See 6.8 “Statistics filtering” (p. 84) for result filter and request filter information.

A.6 To configure the registerLogToFile or registerSasLogToFile client inactivity check

A.6.1 Purpose

By default, the client inactivity check is enabled. In order to avoid client deregistration after a period of client inactivity, you can disable the inactivity check for clients that use one or both of the following methods:

- registerLogToFile
- registerSasLogToFile

Perform this procedure to enable or disable client inactivity checking.

CAUTION

Service Disruption

Modifying the server configuration can have serious consequences including service disruption.

Contact technical support before you attempt to modify the server configuration.

Note: You must perform the procedure on each main server in the NFM-P system.

A.6.2 Steps

1

Log in to the main server station as the nsp user.

2

Navigate to the /opt/nsp/nfmp/server/nms/config directory.

3

Create a backup copy of the nms-server.xml file.

4

Open the nms-server.xml file using a plain-text editor such as vi.

5

To configure the registerLogToFile inactivity check, perform the following steps.

1. Locate the section that begins with following XML tag:

   `<logToFile`

2. Edit the following line in the section to read:

   `enableJmsClientInactivityCheck="`*`value`*`"`

   where *value* is true, which enables the check, or false, which disables the check

6

To configure the registerSasLogToFile inactivity check, perform the following steps.

1. Locate the section that begins with following XML tag:

   `<saslogtofile`

2. Edit the following line in the section to read:

   `enableSasJmsClientInactivityCheck="`*`value`*`"`

   where *value* is true, which enables the check, or false, which disables the check

7

Save and close the nms-server.xml file.

8

Enter the following:

```
bash$ /opt/nsp/nfmp/server/nms/bin/nmsserver.bash read_config ↵
```

The main server configuration is updated.

9

Close the console window.

END OF STEPS

B JMS events

B.1 JMS events

B.1.1 Overview

This appendix lists and describes the NFM-P JMS event classes that are available for monitoring by OSS clients.

For information about JMS client configuration and subscribing to NFM-P JMS topics, see Chapter 4, “Event monitoring using JMS” .

B.1.2 JMS XML schemas

JMS XML schemas describe the structure and allowed elements of a document, similar to a DTD.

The NFM-P documentation contains information about schemas, objects, methods, parameters, and changes between releases.

Table B-1 JMS XML schema files

File	Description
xmlApiJms.xsd	JMS general messages
xmlApiJmsHeader.xsd	JMS messages for filtering
xmlApiJmsTypes.xsd	Types used in the JMS header and body
xmlApiJmsBody.xsd	Event body format

Some events contain complete objects defined in the standard NFM-P XML schemas. See B.2 “JMS event classes” (p. 337) for more information. See Chapter 3, “Communicating with the NFM-P” for information about the W3C standards for XML.

B.2 JMS event classes

B.2.1 JMS event classes

The following table lists the JMS event classes. Each class name is a link to an event message example.

Table B-2 JMS event classes

Event class	Description
“AlarmStatusChangeEvent” (p. 339)	Indicates an alarm status change
“AttributeValueChangeEvent” (p. 339)	Indicates a change to one or more parameter in the database
“DBActivityEvent example” (p. 341)	Indicates database switchover or failover activity

Table B-2 JMS event classes (continued)

Event class	Description
"DBConnectionStateChangeEvent example" (p. 341)	Indicates a change in the database connection state
"DBErrorEvent example" (p. 342)	Indicates a new database error or the correction of an earlier error
"DBProxyStateChangeEvent example" (p. 342)	Indicates a database proxy status change
"DeployerEvent example" (p. 343)	Indicates the success or failure of an asynchronous deployment request; is available only in XML topics
"EventVessel example" (p. 343)	Indicates that the message is an event vessel that contains one or more events
"ExceptionEventXMLFormat example" (p. 344)	Indicates the occurrence of an exception
"FileAvailableEvent example" (p. 345)	Indicates that a file is available for retrieval; applies to asynchronous and synchronous findToFile requests
"IncrementalRequestEvent example" (p. 346)	Generated for activity related to generic.GenericObject.triggerIncrementalRequest
"KeepAliveEvent example" (p. 346)	A keep-alive message that is sent periodically to ensure that the server is running and can communicate with the OSS client
"LogFileAvailableEvent example" (p. 347)	Indicates that an accounting statistics file is ready for retrieval
"LogRemoteFileAvailableEvent example" (p. 347)	Notify Call Trace data file availability on the NE, not on NFM-P. The event includes the filename, NE FTP server IP address, and the NE ID.
"ObjectCreationEvent example" (p. 348)	Identifies a newly created object; contains the object details
"ObjectDeletionEvent example" (p. 349)	Identifies a newly deleted object; contains the object details
"PolicyDistributionEvent example" (p. 350)	Indicates the status of a policy distribution task to NEs. If the request is from XML API, a task monitor is added to the distribute task. Once the task is completed, the event is sent. The maximum number of sites per event is 1000. As a result, there may be multiple completed events, each with up to 1000 sites.
"RelationshipChangeEvent example" (p. 351)	Indicates that a relationship has changed between objects
"ScriptExecutionEvent example" (p. 351)	Indicates script execution by an OSS or GUI client. If specified, the message is sent only to the client that executed the script. Identifies script execution success or failure, the result full name, and the location of the file that contains the result.
"StateChangeEvent" (p. 352)	Indicates a server change, such as reloaded alarm information or changed alarm count An example of an important state change event is the SystemInfoEvent, which is sent each time an OSS client creates a new subscription or connects to a subscription.
"StatsEvent example" (p. 354)	Indicates statistics-collection activity; includes the statistics type, collection time, and the NE identifier, and marks the start or end of polling for an NE, as indicated by the <state> parameter
"TerminateClientSessionEvent example" (p. 355)	Identifies a client session to close for security reasons

Table B-2 JMS event classes (continued)

Event class	Description
"XMLFilterChangeEvent example" (p. 355)	Indicates a successful XML filter change, as requested by the OSS client

AlarmStatusChangeEvent

Figure B-1 AlarmStatusChangeEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <MTOSI_osTime>1138396703631</MTOSI_osTime>
         <ALA_clientId />
         <MTOSI_objectType>AlarmStatusChange</MTOSI_objectType>
         <MTOSI_NTType>NT_ATTRIBUTE_VALUE_CHANGE</MTOSI_NTType>
         <ALA_category>FAULT</ALA_category>
         <ALA_allomorphic>TiEvent</ALA_allomorphic>
         <MTOSI_objectName>svc-mgr:service-3:10.1.1.143</MTOSI_objectName>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <alarmStatusChangeEvent>
            <objectFullName>svc-mgr:service-3:10.1.1.143</objectFullName>
            <attribute>
               <attributeName>alarmStatus</attributeName>
               <value>2</value>
            </attribute>
         </alarmStatusChangeEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

AttributeValueChangeEvent

Figure B-2, "AttributeValueChange message example" (p. 340) shows a JMS message example for the AttributeValueChangeEvent class. An AttributeValueChangeEvent consists of a pointer with a list of changed attributes; the event includes the old and new attribute values. For enum and bitmask attribute types, the message includes the old and new text strings associated with the old and new attribute values, as shown in the example.

For more information about the attributes, or parameters, of specific classes, see the XML API Reference. For parameters that are enumerators, the integer value is used for AttributeValueChangeEvents, while the string value is used in ObjectCreationEvents.

Figure B-2 AttributeValueChange message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <MTOSI_osTime>1138123820148</MTOSI_osTime>
         <ALA_clientId />
         <MTOSI_objectType>netw.NetworkElement</MTOSI_objectType>
         <MTOSI_NTType>NT_ATTRIBUTE_VALUE_CHANGE</MTOSI_NTType>
         <ALA_category>GENERAL</ALA_category>
         <ALA_allomorphic>netw.NetworkElement</ALA_allomorphic>
         <MTOSI_objectName>network:10.1.186.218</MTOSI_objectName>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <attributeValueChangeEvent>
            <objectFullName>network:10.1.186.218</objectFullName>
            <attribute>
               <attributeName>resyncStatus</attributeName>
               <newValue>
                  <int>7</int>
               </newValue>
               <newValueString>requested</newValueString>
               <oldValue>
                  <int>4</int>
               </oldValue>
               <oldValueString>failed</oldValueString>
            </attribute>
            <attribute>
               <attributeName>lastTimeResyncStarted</attributeName>
               <newValue>
                  <long>1138123819632</long>
               </newValue>
               <oldValue>
                  <long>1138123519629</long>
               </oldValue>
            </attribute>
         </attributeValueChangeEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

DBActivityEvent example

Figure B-3 DBActivityEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <MTOSI_osTime>1138395709210</MTOSI_osTime>
         <ALA_clientId />
         <MTOSI_objectType>DBActivityEvent</MTOSI_objectType>
         <MTOSI_NTType>NT_STATE_CHANGE</MTOSI_NTType>
         <ALA_category>DATABASE</ALA_category>
         <ALA_allomorphic />
         <MTOSI_objectName />
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <dbActivityEvent>
            <state>failoverEnd</state>
         </dbActivityEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

DBConnectionStateChangeEvent example

Figure B-4 DBConnectionStateChangeEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <MTOSI_osTime>1138395709210</MTOSI_osTime>
         <ALA_clientId />
         <MTOSI_objectType>DBConnectionStateChangeEvent</MTOSI_objectType>
         <MTOSI_NTType>NT_STATE_CHANGE</MTOSI_NTType>
         <ALA_category>DATABASE</ALA_category>
         <ALA_allomorphic />
         <MTOSI_objectName />
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <dbConnectionStateChangeEvent>
            <state>connectionUp</state>
         </dbConnectionStateChangeEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

DBErrorEvent example

Figure B-5 DBErrorEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <MTOSI_osTime>1138395709210</MTOSI_osTime>
         <ALA_clientId />
         <MTOSI_objectType>DBErrorEvent</MTOSI_objectType>
         <MTOSI_NTType>NT_STATE_CHANGE</MTOSI_NTType>
         <ALA_category>DATABASE</ALA_category>
         <ALA_allomorphic />
         <MTOSI_objectName />
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <dbErrorEvent>
            <state>clear</state>
            <error />
         </dbErrorEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

DBProxyStateChangeEvent example

Figure B-6 DBProxyStateChangeEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <MTOSI_osTime>1138395709210</MTOSI_osTime>
         <ALA_clientId />
         <MTOSI_objectType>DBProxyStateChangeEvent</MTOSI_objectType>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <ALA_category>DATABASE</ALA_category>
         <ALA_allomorphic />
         <MTOSI_objectName />
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <dbProxyStateChangeEvent>
            <state>databaseProxyPrimaryUp</state>
         </dbProxyStateChangeEvent>
      </jms>
   </SOAP:Body>
```

```
</SOAP:Envelope>
```

DeployerEvent example

Figure B-7 DeployerEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <MTOSI_osTime>1165863370530</MTOSI_osTime>
         <ALA_clientId />
         <MTOSI_objectType>DeployerEvent</MTOSI_objectType>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <ALA_category>GENERAL</ALA_category>
         <ALA_allomorphic />
         <MTOSI_objectName />
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <deployerEvent>
            <requestID>JMS_client@n</requestID>
            <successList>
               <deployerId>Default.DeployerBank:depl-486</deployerId>
            </successList>
            <failedList>
               <deployerId>Default.DeployerBank:depl-487</deployerId>
            </failedList>
         </deployerEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

EventVessel example

Figure B-8 EventVessel message example

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>EventVessel</eventName>
         <ALA_category>GENERAL</ALA_category>
         <ALA_vesselSize>3</ALA_vesselSize>
         <ALA_isVessel>true</ALA_isVessel>
         <MTOSI_osTime>1308318280367</MTOSI_osTime>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <ALA_eventName>EventVessel</ALA_eventName>
```

```
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <eventVessel>
            <attributeValueChangeEvent>
               <objectFullName>faultManager:network@192.168.101.13|alarm-31-4-24
</objectFullName>
               <attribute>
                  <attributeName>numberOfOccurencesSinceClear</attributeName>
                  <newValue>
                     <long>3</long>
                  </newValue>
                  <oldValue>
                     <long>2</long>
                  </oldValue>
               </attribute>
            </attributeValueChangeEvent>
            <statsEvent>
               <state>begin</state>
               <networkElement>192.168.101.13</networkElement>
               <statsType>accounting</statsType>
               <time>1308256029287</time>
            </statsEvent>
            <statsEvent>
               <state>end</state>
               <networkElement>192.168.101.13</networkElement>
               <statsType>accounting</statsType>
               <time>1308256029302</time>
            </statsEvent>
         </eventVessel>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

ExceptionEventXMLFormat example

Figure B-9 ExceptionEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <MTOSI_osTime>1174926836987</MTOSI_osTime>
         <ALA_clientId />
         <MTOSI_objectType>aengr.Policy</MTOSI_objectType>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <ALA_category>GENERAL</ALA_category>
         <ALA_allomorphic />
         <MTOSI_objectName>Access Egress:2100</MTOSI_objectName>
```

```
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <ExceptionEventXMLFormat>
            <className>aengr.Policy</className>
            <operation>distribute on node 10.168.1.91</operation>
            <objectName>Access Egress:2100</objectName>
            <requestID>JMS_client@n</requestID>
            <errorMessage>[ app: autoconfigChild ] [ class: aengr.Policy ] [
instance: network:10.168.1.91:Access Egress:2100 ] [ descr: invalid action
bitmask: 3 : Object deletion is in progress, the object cannot be configured
:Parent = network:10.168.1.91:Access Egress:2100: child = queue-1 ]</errorMessage>
         </ExceptionEventXMLFormat>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

FileAvailableEvent example

Figure B-10 FileAvailableEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <MTOSI_osTime>1138129016027</MTOSI_osTime>
         <ALA_clientId />
         <MTOSI_objectType>FileAvailableEvent</MTOSI_objectType>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <ALA_category>GENERAL</ALA_category>
         <ALA_allomorphic />
         <MTOSI_objectName />
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <fileAvailableEvent>
            <fileName>equipmentPhysicalPort.xml</fileName>
         </fileAvailableEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

IncrementalRequestEvent example

Figure B-11 IncrementalRequestEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>IncrementalRequestEvent</eventName>
         <ALA_category>GENERAL</ALA_category>
         <ALA_OLC>0</ALA_OLC>
         <ALA_isVessel>false</ALA_isVessel>
         <ALA_allomorphic/>
         <MTOSI_osTime>1276545012449</MTOSI_osTime>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <MTOSI_objectName/>
         <ALA_clientId/>
         <MTOSI_objectType>IncrementalRequestEvent</MTOSI_objectType>
         <ALA_eventName>IncrementalRequestEvent</ALA_eventName>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <IncrementalRequestEvent>
            <requestID>JMS_client@n</requestID>
            <incrementalAction>RE_EVAL_MSAPS_ON_MSAP_POLICY</incrementalAction>
            <serviceType>default</serviceType>
            <status>finished</status>
            <originalParam>
               <string>msapPolicy:msappolicy</string>
            </originalParam>
            <payload>
               <boolean>true</boolean>
            </payload>
         </IncrementalRequestEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

KeepAliveEvent example

Figure B-12 KeepAliveEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <MTOSI_osTime>1137166251642</MTOSI_osTime>
         <ALA_clientId />
         <MTOSI_objectType>KeepAliveEvent</MTOSI_objectType>
         <MTOSI_NTType>NT_HEARTBEAT</MTOSI_NTType>
```

```
         <ALA_category>GENERAL</ALA_category>
         <ALA_allomorphic />
         <MTOSI_objectName />
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <keepAliveEvent />
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

LogFileAvailableEvent example

Figure B-13 LogFileAvailableEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>LogFileAvailable</eventName>
         <MTOSI_osTime>1184770068031</MTOSI_osTime>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <MTOSI_objectType>LogFileAvailableEvent</MTOSI_objectType>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <ALA_category>GENERAL</ALA_category>
         <ALA_allomorphic />
         <MTOSI_objectName />
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <logFileAvailableEvent>
            <fileName>client_directory\NE_IP_address_1184770068031.xml</fileName>
            <serverIpAddress>server_IP_address</serverIpAddress>
            <routerId>NE_IP_address</routerId>
         </logFileAvailableEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

LogRemoteFileAvailableEvent example

Figure B-14 LogRemoteFileAvailableEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>LogRemoteFileAvailable</eventName>
         <ALA_isVessel>false</ALA_isVessel>
```

```
         <MTOSI_objectType>LogRemoteFileAvailableEvent</MTOSI_objectType>
         <JMSXDeliveryCount>0</JMSXDeliveryCount>
         <ALA_allomorphic />
         <ALA_OLC>0</ALA_OLC>
         <ALA_category>GENERAL</ALA_category>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <ALA_eventName>LogRemoteFileAvailable</ALA_eventName>
         <MTOSI_osTime>1527258136621</MTOSI_osTime>
         <MTOSI_objectName/>
         <ALA_clientId/>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <logRemoteFileAvailableEvent>
            <fileName>/shared/restfm/traces/IMSI_12345678901_TRACE-ID_13_2018-05-25-
17-22-14-125.pcap</fileName>
            <serverIpAddress>135.121.103.12</serverIpAddress>
            <neId>135.121.103.12</neId>
         </logRemoteFileAvailableEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

ObjectCreationEvent example

Figure B-15 ObjectCreationEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <MTOSI_osTime>1138128165344</MTOSI_osTime>
         <ALA_clientId />
         <MTOSI_objectType>aclfilter.MacFilterEntry</MTOSI_objectType>
         <MTOSI_NTType>NT_OBJECTCREATION</MTOSI_NTType>
         <ALA_category>GENERAL</ALA_category>
         <ALA_allomorphic>aclfilter.MacFilterEntry</ALA_allomorphic>
         <MTOSI_objectName>MAC Filter:3:entry-1</MTOSI_objectName>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <objectCreationEvent>
            <aclfilter.MacFilterEntry>
               <frameType>unspecified</frameType>
               <sourceMacAddress>10-00-00-00-00-10</sourceMacAddress>
               <sourceMacAddressMask>00-00-00-00-00-00</sourceMacAddressMask>
               <destinationMacAddress>10-00-00-00-00-10</destinationMacAddress>
```

```
               <destinationMacAddressMask>00-00-00-00-00-00</destinationMacAddressMask>

               <dot1pValue>notSet</dot1pValue>
               <dot1pMask>unspecified</dot1pMask>
               <ethernetType>-1</ethernetType>
               <dsap>-1</dsap>
               <dsapMask>-1</dsapMask>
               <ssap>-1</ssap>
               <ssapMask>-1</ssapMask>
               <snapPid>-1</snapPid>
               <snapOui>off</snapOui>
               <action>default</action>
               <logId>0</logId>
               <administrativeState>notInService</administrativeState>
               <containingPolicyDisplayedName>ACL Mac Filter 2</containingPolicyDisplayedNam

               <containingPolicyId>3</containingPolicyId>
               <policyType>macAcl</policyType>
               <isLocal>false</isLocal>
               <siteId>0.0.0.0</siteId>
               <siteName>N/A</siteName>
               <displayedName>Filter 10-00-00-00-00-10</displayedName>
               <description>Filter Entry # 1</description>
               <id>1</id>
               <globalPolicy>N/A</globalPolicy>
               <deploymentState>0</deploymentState>
               <objectFullName>MAC Filter:3:entry-1</objectFullName>
               <name>Filter 10-00-00-00-00-10</name>
               <selfAlarmed>false</selfAlarmed>
               <children-Set />
            </aclfilter.MacFilterEntry>
         </objectCreationEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

ObjectDeletionEvent example

Figure B-16 ObjectDeletionEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>ObjectDeletion</eventName>
         <MTOSI_osTime>1224518670453</MTOSI_osTime>
         <ALA_clientId/>
         <MTOSI_NTType>NT_OBJECTDELETION</MTOSI_NTType>
         <MTOSI_objectType>equipment.PhysicalPort</MTOSI_objectType>
         <ALA_category>EQUIPMENT</ALA_category>
```

```
         <ALA_isVessel>false</ALA_isVessel>
         <ALA_allomorphic>equipment.PhysicalPort</ALA_allomorphic>
         <ALA_eventName>ObjectDeletion</ALA_eventName>
         <ALA_span>:0:</ALA_span>
         <MTOSI_objectName>network:15.1.1.89:shelf-1:cardSlot-1:card:daughterCardSlot-
1:daughterCard:port-53</MTOSI_objectName>
         <ALA_OLC>2</ALA_OLC>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <objectDeletionEvent>
            <objectFullName>network:15.1.1.89:shelf-1:cardSlot-1:card:daughterCardSlot-
1:daughterCard:port-53</objectFullName>
         </objectDeletionEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

PolicyDistributionEvent example

Figure B-17 PolicyDistributionEvent message example

```
<PolicyDistributeEvent isAlarm="no">
   <properties>
      <eventName>PolicyDistributeStatusEvent</eventName>
      <ALA_OLC>0</ALA_OLC>
      <ALA_category>GENERAL</ALA_category>
      <ALA_isVessel>false</ALA_isVessel>
      <MTOSI_osTime>1365047500446</MTOSI_osTime>
      <ALA_allomorphic/>
      <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
      <MTOSI_objectName>rpCommunity:testcommunity</MTOSI_objectName>
      <ALA_clientId>JMS_client@n</ALA_clientId>
      <MTOSI_objectType>rp.Community</MTOSI_objectType>
      <ALA_eventName>PolicyDistributeStatusEvent</ALA_eventName>
   </properties>
   <Body>
      <PolicyDistributeEventXMLFormat>
         <className>rp.Community</className>
         <operation>distribute</operation>
         <objectFullName>rpCommunity:testcommunity</objectFullName>
         <DistributionStatus>
            <policy.DistributeSiteStatus>
               <siteId>198.51.100.43</siteId>
               <status>Failed</status>
            </policy.DistributeSiteStatus>
            <policy.DistributeSiteStatus>
               <siteId>198.51.100.83</siteId>
```

```
               <status>Succeeded</status>
            </policy.DistributeSiteStatus>
         </DistributionStatus>
      </PolicyDistributeEventXMLFormat>
   </Body>
</PolicyDistributeEvent>
```

RelationshipChangeEvent example

Figure B-18 RelationshipChangeEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <MTOSI_osTime>1138386054650</MTOSI_osTime>
         <ALA_clientId />
         <MTOSI_objectType>epipe.Epipe</MTOSI_objectType>
         <MTOSI_NTType>ALA_RELATIONSHIPCHANGE</MTOSI_NTType>
         <ALA_category>SERVICE</ALA_category>
         <ALA_allomorphic />
         <MTOSI_objectName>svc-mgr:service-3</MTOSI_objectName>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <relationshipChangeEvent>
            <objectFullName>svc-mgr:service-3</objectFullName>
            <relationship>
               <changeType>added</changeType>
               <fromObjectName>svc-mgr:service-3</fromObjectName>
               <fromObjectClass>epipe.Epipe</fromObjectClass>
               <toObjectName>subscriber:1</toObjectName>
               <toObjectClass>subscr.Subscriber</toObjectClass>
            </relationship>
         </relationshipChangeEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

ScriptExecutionEvent example

Figure B-19 ScriptExecutionEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <MTOSI_osTime>1173717919050</MTOSI_osTime>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <MTOSI_objectType>ScriptExecutionEvent</MTOSI_objectType>
```

```
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <ALA_category>GENERAL</ALA_category>
         <ALA_allomorphic />
         <MTOSI_objectName />
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <scriptExecutionEvent>
            <targetScriptResultFullName>script-manager:script-4:target-script-6:target-
script-result-1173467653236</targetScriptResultFullName>
            <fileName>scriptResult/target-script-6_1173467653236.txt</fileName>
            <status>Unknown</status>
            <errorMessage>Trying to connect to an already connected session.</
errorMessage>
            <failedCommands />
         </scriptExecutionEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

StateChangeEvent

Table B-3 State change events

Event type	Description
AlarmInformationLoaded	Sent to indicate a change in the status of the alarm list; see the XML schema for the possible status values
AlarmCountChanged	Sent when the number of alarms crosses above or below the maximum number of alarms defined in the server configuration
JmsMissedEvents	Sent to inform both durable and non-durable subscribers that events have been missed
SystemInfoEvent	Sent each time a client connects to a new subscription, or reconnects to an existing durable subscription

Figure B-20 JmsMissedEvents message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>JmsMissedEvents</eventName>
         <MTOSI_osTime>1222891102168</MTOSI_osTime>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <MTOSI_objectType>StateChangeEvent</MTOSI_objectType>
         <ALA_category>GENERAL</ALA_category>
         <ALA_isVessel>false</ALA_isVessel>
         <ALA_allomorphic/>
```

```
         <ALA_eventName>JmsMissedEvents</ALA_eventName>
         <MTOSI_objectName/>
         <ALA_OLC>0</ALA_OLC>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <stateChangeEvent>
            <eventName>JmsMissedEvents</eventName>
            <state>jmsMissedEvents</state>
         </stateChangeEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

The following table lists the properties in a System Information StateChangeEvent message.

Table B-4 StateChangeEvent SystemInfoEvent properties

Property name	Description
jmsStartTime	UNIX time of most recent JMS-server restart; long integer
sysPrimaryIp	IP address or hostname (if client hostname was configured via samconfig) of primary main server; string[1]
sysStandbyIp	IP address or hostname (if client peer-server hostname was configured via samconfig) of standby main server in redundant NFM-P deployment; string[1]
sysStartTime	UNIX time of most recent main-server restart; long integer
sysType	NFM-P system type; string, value is redundant or standalone

Notes:

1. When not using hostnames in a NAT environment, the public IP address(es) are used.

The SystemInfoEvent is sent to a JMS client's subscription after the OSS establishes a JMS connection with the NFM-P.

The SystemInfoEvent sysPrimaryIp tells the OSS which NFM-P server is the current primary. An OSS with a durable JMS subscription should initiate retrieval of inventory or alarms using HTTP if any of the following occurs:

- The current sysPrimaryIp in the SystemInfoEvent is different from that of the last recorded sysPrimaryIp.
- The jmsStartTime or sysStartTime in the SystemInfoEvent are different from the last recorded jmsStartTime and sysStartTime.

See A.2 "Recommended durable JMS client operation" (p. 327) for more information about the recommended start-up procedure for an OSS with a durable JMS subscription to the XML API.

Figure B-21 SystemInfoEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <eventName>SystemInfoEvent</eventName>
         <MTOSI_osTime>1224527204299</MTOSI_osTime>
         <ALA_clientId>JMS_client@n</ALA_clientId>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <MTOSI_objectType>StateChangeEvent</MTOSI_objectType>
         <ALA_category>GENERAL</ALA_category>
         <ALA_isVessel>false</ALA_isVessel>
         <ALA_allomorphic/>
         <ALA_eventName>SystemInfoEvent</ALA_eventName>
         <MTOSI_objectName/>
         <ALA_OLC>0</ALA_OLC>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <stateChangeEvent>
            <eventName>SystemInfoEvent</eventName>
            <state>systemInfo</state>
            <sysStandbyIp>192.168.182.253</sysStandbyIp>
            <sysPrimaryIp>192.168.169.94</sysPrimaryIp>
            <jmsStartTime>1224525628189</jmsStartTime>
            <sysStartTime>1224525628189</sysStartTime>
            <sysType>Redundant</sysType>
         </stateChangeEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

StatsEvent example

Figure B-22 StatsEvent message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <MTOSI_osTime>1170864259082</MTOSI_osTime>
         <ALA_clientId />
         <MTOSI_objectType>StatsEvent</MTOSI_objectType>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <ALA_category>STATISTICS</ALA_category>
         <ALA_allomorphic />
         <MTOSI_objectName />
      </header>
   </SOAP:Header>
```

```
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <statsEvent>
            <state>end</state>
            <networkElement>10.1.200.71</networkElement>
            <statsType>accounting</statsType>
            <time>1170864259082</time>
         </statsEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

TerminateClientSessionEvent example

Figure B-23, “TerminateClientSession message example” (p. 354) shows a JMS message example for the TerminateClientSessionEvent class.

Note: The JMS message terminateClientSessionEvent is broadcast to all sessions.

Figure B-23 TerminateClientSession message example

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <MTOSI_osTime>1138395709210</MTOSI_osTime>
         <ALA_clientId />
         <MTOSI_objectType>TerminateClientSession</MTOSI_objectType>
         <MTOSI_NTType>NT_STATE_CHANGE</MTOSI_NTType>
         <ALA_category>GENERAL</ALA_category>
         <ALA_allomorphic />
         <MTOSI_objectName />
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <terminateClientSessionEvent>
            <clientId>ossi@1</clientId>
         </terminateClientSessionEvent>
      </jms>
   </SOAP:Body>
</SOAP:Envelope>
```

XMLFilterChangeEvent example

Figure B-24 XMLFilterChangeEvent message example

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
```

```
      <header xmlns="xmlapi_1.0">
         <eventName>XMLFilterChangeEvent</eventName>
         <ALA_isVessel>false</ALA_isVessel>
         <MTOSI_objectType>XMLFilterChangeEvent</MTOSI_objectType>
         <JMSXDeliveryCount>0<JMSXDeliveryCount>
         <ALA_allomorphic/>
         <ALA_OLC>0</ALA_OLC>
         <ALA_category>GENERAL</ALA_category>
         <MTOSI_NTType>ALA_OTHER</MTOSI_NTType>
         <ALA_eventName>XMLFilterChangeEvent</ALA_eventName>
         <MTOSI_osTime>1536329823494</MTOSI_osTime>
         <MTOSI_objectName/>
         <ALA_clientId>AdvancedJMSFilter@1</ALA_clientId>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <jms xmlns="xmlapi_1.0">
         <xmlFilterChangeEvent>
            <filter-Set>
            <filter>
            <or>
            <equal name="className" value="fm.AlarmObject"/>
            <equal name="className" value="fm.AlarmNoteObject"/>
            <or>
            <filter>
            </filter-Set>
            <resultFilter-Set>
            <resultFilter class="fm.AlarmNoteObject">
            <attribute>isAckNote</attribute>
            <attribute>objectFullName</attribute>
            <attribute>status</attribute>
            </resultFilter>
            <resultFilter class="fm.AlarmObject">
            <attribute>severity</attribute>
            <attribute>affectedObjectFullName</attribute>
            <attribute>numberOfOccurences</attribute>
            <attribute>alarmClassTag</attribute>
            <attribute>firstTimeDetected</attribute>
            <attribute>additionalText</attribute>
            <attribute>operatorAssignedUrgency</attribute>
            <attribute>alarmName</attribute>
            <attribute>objectFullName</attribute>
            <attribute>acknowldegedBy</attribute>
            </resultFilter>
            <resultFilter-Set>
            <extraTags>
            </tag name="MTOSI_osTime"/>
            </tag name="fullClassName"/>
            </tag name="ALA_category"/>
```

```
            </extraTags>
          </xmlFilterChangeEvent>
       </jms>
    </SOAP:Body>
</SOAP:Envelope>
```

C Troubleshooting

C.1 Troubleshooting client OSS application problems

C.1.1 Introduction

The following procedures describe how to troubleshoot OSS application-specific problems.

Note: See the *NSP Troubleshooting Guide* for more information about how to troubleshoot common client issues.

Table C-1 Troubleshooting client OSS application problems procedures

Procedure
C.2 “The OSS client cannot communicate with the server” (p. 359)
C.3 “An attempt to log in to the NFM-P server fails” (p. 361)
C.4 “Unable to perform an action using the XML API” (p. 361)
C.5 “Receive insufficient privileges to perform this operation on an object exception when performing an action on an NFM-P object” (p. 362)
C.6 “Receive an authorization failure to access an object exception when performing an action on an NFM-P object” (p. 363)
C.7 “Identifying XML messages from specific users” (p. 363)
C.8 “Receive a java.lang.UnsupportedClassVersionError when sending scripts using an OSS client” (p. 365)
C.9 “Receive a java.net.ConnectException when sending scripts using an OSS Client” (p. 366)
C.10 “The OSS client cannot connect with the HTTP or JMS server” (p. 366)
C.11 “The OSS client cannot perform find or findTofile requests” (p. 367)

C.2 The OSS client cannot communicate with the server

C.2.1 Introduction

The following XML API ping sample can be used to test whether the OSS application can access the NFM-P server. Information in the SOAP/XML request includes:

- standard SOAP user and password encoding
- the ping command to look for the release of XML on the server, which in this case, is Release 1.0

An exception response to a failed ping may result in many types of return messages, for example:

- socket timeouts
- HTTP errors
- connection exceptions

C.2.2 Steps

1

Run a ping command similar to the following in Figure C-1, “Ping request example” (p. 359) .

Figure C-1 Ping request example

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <security>
            <user>username</user>
            <password>password</password>
         </security>
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <ping xmlns="xmlapi_1.0"/>
   </SOAP:Body>
</SOAP:Envelope>
```

2

Review the three-digit HTTP response code.

- 200 to 299—indicates success; client-server communication is not the problem
- 400 to 499—indicates that the error is on the client side; the request contains incorrect XML syntax, or cannot be fulfilled
- 500 to 599—indicates that the error is on the server; the request is not the problem

3

Check the XML API response for any indication of other communication problems. Figure C-2, “Ping response message” (p. 360) is a successful response message to the ping. The response indicates the correct release 1.0 of XML on the server that responds to the ping.

Figure C-2 Ping response message

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <requestID>XML_API_client@n</requestID>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <pingResponse xmlns="xmlapi_1.0"/>
```

```
   </SOAP:Body>
</SOAP:Envelope>
```

END OF STEPS

C.3 An attempt to log in to the NFM-P server fails

C.3.1 Steps

1

Send an OSS request to the NFM-P.

2

If you receive the SOAP fault message in Figure C-3, “SOAP fault message” (p. 360) , check the following:

- incorrect username or password
- user does not exist
- username or password missing
- password not hashed; see 3.4 “Secure communication” (p. 25) for information about MD5-hashed passwords

Figure C-3 SOAP fault message

```
<SOAP:Fault>
   <faultcode>SOAP:Client</faultcode>
   <faultstring>[security] Login failure.</faultstring>
   <faultactor>XmlApi</faultactor>
   <detail>
      <requestID>XML_API_client@n</requestID>
   </detail>
</SOAP:Fault>
```

END OF STEPS

C.4 Unable to perform an action using the XML API

C.4.1 Steps

1

Check with your administrator to determine whether you have sufficient privileges to perform the action. You receive the message in Figure C-4, “User security fault message” (p. 362) if you are not assigned the appropriate scope of command to use the XML API.

Figure C-4 User security fault message

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <requestID>XML_API_client@n</requestID>
         <requestTime>Apr 1, 2014 2:35:46 PM</requestTime>
         <responseTime>Apr 1, 2014 2:35:46 PM</responseTime>
      </header>
   </SOAP:Header>
   <SOAP:Fault>
      <faultcode>SOAP:Client</faultcode>
      <faultstring>[security] Users require OSS Management privileges to use XML
API</faultstring>
      <faultactor>XmlApi</faultactor>
      <detail>
         <requestID>XML_API_client@n</requestID>
      </detail>
   </SOAP:Fault>
</SOAP:Envelope>
```

2

Some actions are not available to an OSS application; for example, a GUI-specific action such as changing a display preference. For such an action, you must use a GUI client.

END OF STEPS

C.5 Receive insufficient privileges to perform this operation on an object exception when performing an action on an NFM-P object

C.5.1 Troubleshooting option

You receive the following exception if you are not assigned the appropriate scope of command for an object:

```
[app:generic] [class:GenericObject] [instance:instance:object] [cause:Method Invocation Failed (13)
] [descr:Insufficient privileges to perform this operation. User does not have create-update-delete
access to class  classname] [nested exception:null]
```

Check with your administrator to determine whether you are assigned the appropriate scope of command to allow you to access the NFM-P object.

C.6 Receive an authorization failure to access an object exception when performing an action on an NFM-P object

C.6.1 Troubleshooting option

You receive the following exception if you are not assigned the appropriate span of control for an object:

```
[ app: generic ] [ class: generic.GenericObject ] [ instance: - unknown- ] [ descr: SecurityManager:
Authorization Failure: User does not have access to objectFullName. Object is not in user's span.]
[ exceptionClass: SecurityException ]
```

Check with your administrator to determine whether you are assigned the appropriate span of control to allow you to access the NFM-P object.

C.7 Identifying XML messages from specific users

C.7.1 Introduction

Perform the following procedure to enable the logging of the XML API request, response, and exception messages on the NFM-P. The entries in the XML message log files can help you identify the XML requests of a user, multiple users, or of all of the users. See 9.1.6 “Logging XML requests, responses, and exceptions” (p. 120) in Chapter 9, “XML message structure” for more information about the logging of XML messages.

You must update the nms-sever.xml file on each NFM-P main server with DEBUG logging to view OSS template information in NFM-P server logs.

CAUTION

Service Disruption

Contact your Nokia technical support representative before you attempt to modify the nms-server.xml file.

Modifying the nms-server.xml file can have serious consequences that can include service disruption.

CAUTION

Service Disruption

The purpose of the XML message log is to enable developers to evaluate the interaction of a third-party application with the XML API. The logging consumes system resources and may degrade performance in a live system.

It is strongly recommended that you enable the logging only in an OSS application development environment.

C.7.2 Steps

1

Log in to the main server station as the nsp user.

2

Navigate to the /opt/nsp/nfmp/server/nms/config directory.

3

Create a backup copy of the nms-server.xml file.

4

Open the nms-server.xml file using a plain-text editor.

5

Locate the <systemOssiLog> element.

6

Modify the attributes of the <systemOssiLog> element to enable the log option for an individual user or multiple users. The NFM-P creates a unique log file for each HTTP request and response associated with each user. The request log contains the body of the SOAP message. The response log contains the entire SOAP envelope of the response.

a. To log the XML messages for all users, edit the section to read:

```
<systemOssiLog
        allUsers="yes"
        path="../log/all_users"/>
```

The above example creates files named ossi*user*Request*n*.log and ossi*user*Response*n*+.log in the /opt/nsp/nfmp/server/nms/log/individual directory

where

user is the NFM-P user name

n is the incremental request number

b. To log the XML messages for a single user, edit the section to read:

```
<systemOssiLog
        allUsers="no"
        timeToKeepLogFile="minutes"
        user="yes"
        path="../log/individual"/>
```

where

minutes is the time, in minutes, that the NFM-P retains the log files; when the parameter is absent, the default is 1440, which is equivalent to one day

user is the NFM-P user name

The above example creates files named ossi*user*Request*n*.log and ossi*user*Response*n*.log in the /opt/nsp/nfmp/server/nms/log/individual directory

where

user is the NFM-P user name

n is the incremental request number

7

To change the log file retention period, add the "timeToKeepLogFile="*minutes*" attribute to the <systemOssiLog> element:

```
<systemOssiLog
        .
        .
        .
        "timeToKeepLogFile="minutes" />
```

where *minutes* is the number of minutes to retain log files

When the timeToKeepLogFile attribute is not entered in the <systemOssiLog> element, the default is 24 h.

8

Save and close the nms-server.xml file.

9

Open a console window.

10

Navigate to the /opt/nsp/nfmp/server/nms/bin directory.

11

Enter the following at the prompt:

```
bash$ ./nmsserver.bash read_config ↵
```

The main server reads the nms-server.xml file and puts the configuration changes into effect.

END OF STEPS

C.8 Receive a java.lang.UnsupportedClassVersionError when sending scripts using an OSS client

C.8.1 Notification example

This error occurs when you install Oracle after installing Java. Verify that the right Java version, as described in 4.15 “To determine the Java version” (p. 65), is in the path. The following error message is an example of the notification that your receive when you send script to the NFM-P via an OSS client:

```
Exception in thread "main" java.lang.UnsupportedClassVersionError: OSS Client (Unsupported major.minor
version 48.0)
at java.lang.ClassLoader.defineClass0(Native Method)
at java.lang.ClassLoader.defineClass(ClassLoader.java:495)
at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:110)
at java.net.URLClassLoader.defineClass(URLClassLoader.java:251)
at java.net.URLClassLoader.access$1(URLClassLoader.java:217)
at java.net.URLClassLoader$1.run(URLClassLoader.java:198)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(URLClassLoader.java:192)
at java.lang.ClassLoader.loadClass(ClassLoader.java:300)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:290)
at java.lang.ClassLoader.loadClass(ClassLoader.java:256)
at java.lang.ClassLoader.loadClassInternal(ClassLoader.java:316)
```

C.9 Receive a java.net.ConnectException when sending scripts using an OSS Client

C.9.1 Notification example

You must verify that the correct NFM-P server is configured in the OSS client. By default, the NFM-P server points to localhost. The following error message is an example of the notification that you receive when you send scripts to the NFM-P via an OSS client:

```
Exception in thread "main" java.net.ConnectException: Connection refused: connect
at java.net.PlainSocketImpl.socketConnect(Native Method)
at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:171)
at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:158)
at java.net.Socket.connect(Socket.java:452)
at java.net.Socket.connect(Socket.java:402)
at java.net.Socket.(Socket.java:309)
at java.net.Socket.(Socket.java:124)
at org.apache.commons.httpclient.protocol.DefaultProtocolSocketFactory.createSocket
(DefaultProtocolSocketFactory.java:118)
at org.apache.commons.httpclient.HttpConnection.open(HttpConnection.java:683)
at org.apache.commons.httpclient.HttpClient.executeMethod(HttpClient.java:662)
at org.apache.commons.httpclient.HttpClient.executeMethod(HttpClient.java:529)
at OSSclient.main(OSSclient.java:136)
```

C.10 The OSS client cannot connect with the HTTP or JMS server

C.10.1 Steps

1

Verify that the OSS client uses the correct server IP address and port.

2

If connectivity problems remain, perform the following:

- Use an ICMP ping to verify that you can ping the server from the OSS client station.
- For XML requests, perform an XML ping. See 5.1 “HTTP communication with the NFM-P” (p. 67) for more information.
- For JMS problems:
  - Verify that the samOSS.jar file used by the OSS client application is from the NFM-P release that is on the server. The Implementation-Version in the META-INF/MANIFEST.MF file in the samOss.jar indicates the NFM-P release that the jar file is from.
  - Try to create the same JMS connection using JMSTest. See 4.12 “To compile and connect an NFM-P JMS client” (p. 62) for information.
  - Verify that the NFM-P credentials that the OSS JMS client uses are correct.
  - Check for any indication of errors in the NFM-P JMS server log about the OSS client connection attempt.
- Verify that the required ports are open between the OSS client station and the server. See the *NSP Planning Guide* for firewall configuration information.

END OF STEPS

C.11 The OSS client cannot perform find or findTofile requests

C.11.1 Steps

1

If you receive an exception indicating that the number of inventory connections has reached the maximum, the request has been rejected. This occurs when there are five concurrent find or findToFile requests. There is no indication of the number of concurrent requests. The OSS client must pace the requests.

Figure C-5 find method limit exception

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <requestID>XML_API_client@n</requestID>
         <requestTime>Apr 15, 2010 4:23:38 PM</requestTime>
         <responseTime>Apr 15, 2010 4:23:38 PM</responseTime>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <findResponse xmlns="xmlapi_1.0">
         <result/>
      </findResponse>
      <XMLException xmlns="xmlapi_1.0">
```

```
         <description>The Inventory connections reached its maximum please retry
later</description>
         <line>1</line>
         <column>1078</column>
      </XMLException>
   </SOAP:Body>
</SOAP:Envelope>
```

Figure C-6 findToFile method limit exception

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <header xmlns="xmlapi_1.0">
         <requestID>XML_API_client@n</requestID>
         <requestTime>Apr 15, 2010 4:39:35 PM</requestTime>
         <responseTime>Apr 15, 2010 4:39:35 PM</responseTime>
      </header>
   </SOAP:Header>
   <SOAP:Body>
      <findToFileException xmlns="xmlapi_1.0">
         <description>javax.xml.stream.XMLStreamException: The Inventory
connections reached its maximum please retry later</description>
      </findToFileException>
   </SOAP:Body>
</SOAP:Envelope>
```

2

Retry sending the request when the request volume is lower.

END OF STEPS