

White Paper

XML-based Network Management

Phil Shafer
Senior Staff Engineer



Juniper Networks, Inc.
1194 North Mathilda Avenue
Sunnyvale, CA 94089 USA
408 745 2000 or 888 JUNIPER
www.juniper.net

Part Number : 200017-001 08/01

Contents

Executive Summary	3
Perspective	3
Network Management using XML	4
Ease of Use	4
Readily Handles Complex Data	4
Tools and Live Data	4
Configuration	5
Extensibility and Compatibility	5
XML and Surrounding Technologies	5
XML Basics	6
Transforming XML Using XSLT	7
Using XPath to Find What You Need	8
XML Tools	8
Live Data	9
XML Compatibility	9
Supporting Environment	10
XML-based RPCs	10
RPC Request and Reply	10
Session Management	11
Access Mechanisms	12
Configuration Data	13
Database Locking	14
Failure Modes	15
Example Applications	15
Hardware Inventory	15
Diagnostics	16
Vendor-neutral Configuration	16
Conclusion	17
Acronyms	17
References	18

Executive Summary

XML-based network management uses W3C's extensible markup language (XML)^[1] to encode communications data, providing an excellent mechanism for transmitting the complex data that is used to manage networking gear.

Building an API around an XML-based remote procedure call (RPC) gives a simple, extensible way to exchange this data with a device. Receiving data in XML opens easy options for handling the data using standards-based tools. XML is widely accepted in other problem domains, and free and commercial tools are emerging at an accelerating rate. Using extensible stylesheet language transformations (XSLT) enables you to recast data in differing lights and convert it to and from common formats.

XML-based RPCs provide a standards-based API to devices, allowing external applications to take full advantage of the devices' capabilities. Use of traditional device access mechanisms and communication protocols allows access to a device under any circumstance. Use of existing authentication and authorization mechanisms minimizes the amount of new code in use, reduces the risk of security problems, and increases the trust and comfort of users.

Perspective

The current industry-standard network management methods are SNMP and Expect. Their strengths and weaknesses are well known, and many are unhappy with the level of control provided.

- SNMP is widely deployed and supported. It is a reasonable technology for many network management tasks, but is not always appropriate. Many real-world tasks cannot be addressed without the implementation of proprietary MIBs, which presents problems for both MIB implementers and users.
- When SNMP cannot be used, the Expect tool is often an option that provides immediate access to any data emitted by a device. Creating regular expressions that match CLI output in both current and future vendor software releases is a *black art*. New software often breaks Expect-based tools, adding to the cost and latency of upgrading to new software and adapting the network management tools to new software versions.

Users write Expect scripts that parse CLI output. Vendors know changes to that output will break the scripts, and hence are reluctant to change output. The static nature of these displays prevents iterative improvements, even when the improvement makes Expect scripting easier. For example, changing an output string to provide consistent output style breaks existing scripts that match against the inconsistent output. One user's fix is another's bug.

The features and possibilities of XML-based network management are leading edge and have the potential to improve the interaction between network management tools and the devices they manage. XML-based network management solutions are ideal for resolving the dilemmas posed by existing technologies. The API allows complex data to be encoded in an extensible format using rules that are presented in precise detail and with consistent syntax.

Network Management using XML

To be a worthwhile tool for network management, XML must offer features not available in other management technologies. There are five categories in which XML shows such strength.

Ease of Use

XML proves to be easy to use on many counts.

- Since XML is ASCII, you can generate, parse, handle, store, retrieve, debug, and document it using normal ASCII tools.
- XML is self-describing; for instance, a tag labeled `<input-bytes>` has an obvious meaning.
- You can execute tests by typing raw XML into a connection to the server.
- You can log sessions into text files, and bug reports can include this text and be reproduced by hand.

Readily Handles Complex Data

XML is structured data. Its simple encoding allows the representation of arbitrary hierarchies of data using ASCII text in HTML-like tags. The order and structure of the elements are defined using either document type definition (DTD) or XML Schema^[2] files, which provide a formal mechanism for notification of changes in the underlying data. Using XML comparison tools against old and new XML Schema files can provide details of what features, options, or output tags were added or removed.

Tools and Live Data

XML defines a simple encoding mechanism, but a wide range of tools and supporting standards continue to appear around it. For example, you can use

- XPath to find elements that match a particular set of criteria,
- XSLT to transform one dataset into another,
- XML-enabled databases to store and retrieve XML,
- XML comparison tools to report changes between XML documents, and
- Scalable Vector Graphics (SVG)^[5] to display graphics.

Getting a dataset in a form that can be handed directly to other applications means freedom from writing and supporting custom code to turn queries into structured results. The response to an XML-based query is structured data, and you can use this structured data as input to another application.

XSLT can even turn an XML Schema into XSLT transformation scripts, which you can then run against incoming data. This transformation could take advantage of the schema's ability to contain application-specific information to carry specific instructions into the generated XSLT scripts. As well, XSLT can generate SVGs.

Configuration

Configuration is a promising application for XML-based network management. The rich set of XML-based configuration options and features makes a perfect match for provisioning applications, centralized network databases, configuration patch files, and configuration archival.

Extensibility and Compatibility

XML has the capabilities to allow migration, enhancements, maintenance, and support for fast moving software. Old applications ignore new tags, and new applications can handle the absence of new tags. Servers can emit both new and old tags. Clients can inspect DTD or XML Schema files to determine software capabilities.

One open issue is how standardization efforts will define new tags and how clients and servers could adjust as elements migrate into standard form. Devices could provide XSLT transformations to convert the devices' XML output into standard XML. This implementation would allow support of new standards on old software releases without requiring changes on the device.

XML and Surrounding Technologies

XML's simple encoding of complex hierarchical information in extensible, self-describing tags enables you to easily generate and parse data using free available libraries for all major programming languages. XSLT transformations enable nonprogrammers to use the power and flexibility of XML to perform operations that previously required custom programming.

Example:

An example of XML encoding might be the encoding of a person's contact information.

```

<contact-information>                                <!-- Comments are encoded like this -->
<!-- An open tag begins an element,
  which might contain text, other
  elements, or both. -->
  <person/>                                           <!-- Empty tags like this represent
  flags or other Boolean
  information -->
  <last-name>Shafer</last-name><!-- Data elements contain data -->
  <first-name>Phil</first-name>
  <office>                                             <!-- Nesting elements allows
  representation of complex data
  and relationships -->
    <phone>1-888-586-4737</phone>
    <address>
      <street>940 Main Campus Drive</street>
      <city>Raleigh</city>
      <state abbreviation="NC">North Carolina</state>
      <!-- Attributes might appear on open
      or empty tags -->
    </address>                                        <!-- Close tags mark the end of
    elements -->
  </office>                                           <!-- Every element must be closed -->
</contact-information>                               <!-- XML documents and document
  fragments have one top-level
  element -->

```

XML Basics

XML has the following basic constructs.

- An element refers to a particular open tag, its matching close tag, and all content in between.
 - Open tags begin an element by giving the name of the element in greater-than and less-than signs (also called *chevrons* or *angle brackets*).
Example: `<foo>`
 - Close tags end an element by preceding the name of the element with a slash.
Example: `</foo>`
- Empty tags are equivalent to an open and close tag and consist of the element name followed by a slash.
Example: `<foo/>`
- Data is any text not otherwise part of a tag, and it normally appears between open and close tags.
Example: `<foo>bar</foo>`
- Attributes can appear in either open or empty tags, and consist of the attribute name followed by an equal sign and the quoted attribute value.
Example: `<foo bar="12"/>`
- DTDs define the syntactic relationships between the elements, specifying containment relationships between elements and attributes.
Example:

```
<!ELEMENT office (phone, address)>
    <!ELEMENT address (street, city, state)>
    <!ATTLIST state abbreviation CDATA #IMPLIED>
```
- An XML Schema performs a similar function, but is encoded in XML for simpler manipulation and extension. They enable the creation of more complex element relationships, data types, patterns, and constraints, as well as allowing insertion of application-specific information into a schema.

- XML Namespaces allow the syntactic and semantic definition of elements (and their attributes and data values) to vary by associating portions of documents with different Namespace identifiers. You can reference elements in another Namespace by preceding the element name with the prefix to which the Namespace identifier has been mapped. The `xmlns` attribute provides this mapping.

Example:

In this example, the `container` element defines Namespace as alias `junos`, which is tied to the unique identifying string `http://juniper.net/junos.dtd`. Elements under the `container` element can refer to tags in that Namespace using the alias as a prefix.

```
<container xmlns:junos="http://juniper.net/junos.dtd">
  <junos:feature/> <!-- tag in another Namespace -->
</container>
```

- XML documents begin with a processing instruction that contains the version of XML used in the document (currently fixed at 1.0) and the character set encoding scheme, normally either `us-ascii`, `iso-8859-1`, or `UTF-8` for unicode.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
```

XML is simply structured text. XML is easy to generate, easy to understand, easy to parse, and easy to debug. You can create XML text using text editors and `printf()` calls, parse it using open-source libraries, debug it using the `p` command in `gdb`, transport it in e-mail, store it in text files, manipulate it using traditional Unix tools, indent and print it using simple rules, and transmit it over normal interprocess communication mechanisms.

Transforming XML Using XSLT

XML forms the nucleus of a set of standards. The most applicable for network management is the extensible stylesheet language (XSL)^[3]. Originally conceived as a stylesheet language for rendering XML in browser windows, XSL also functions as a transformation language (XSLT) that can transform XML into either different XML or text. Common transformations are as follows.

- Converting XML to XHTML^[4], an XML form of HTML to be displayed in browsers
- Turning data into text reports or HTML Web pages
- Sorting and re-ordering data
- Inverting indices or reversing the hierarchy of tags in a document
- Discarding uninteresting information, such as zero values or counters that are not being monitored
- Adding additional information, such as turning elements into hyperlinks to additional information
- Changing one vocabulary of XML into another vocabulary

XSLT is not a programming language, though it has some mechanisms for pseudo-programming. There are IDEs that allow nonprogrammers to build XSLT transformations, and then test and debug them against real data. You need not be a programmer to construct transformations to manipulate data.

Using XPath to Find What You Need

Another XML-related standard is XPath, which describes a set of elements within an XML document. XPath allows the specification of a path through the document's element hierarchy, as well as a mechanism for selecting elements from that hierarchy.

Example:

This XPath expression matches all `name` elements that are parented by `physical-interface` that are in turn parented by `interface-information` elements.

```
/interface-information/physical-interface/name
```

You can add a `predicate` expression that further limits the set of matching elements. These expressions appear in square brackets after an element of the attribute name.

Example:

This expression matches all `status` elements that are parented by an `environment-item` whose `name` attribute is `Power Supply B`.

```
environment-item[@name="Power Supply B"]/status
```

XSLT uses XPath to describe the nodes upon which transformations are to be performed.

Example:

```
<xsl:template match="environment-item[status != 'OK']">
  <!-- XSL statements here would be applied to an environment-item
       element that contains a status element whose value is not
       equal to "OK" -->
</xsl:template>

<xsl:template match="*[temperature/@junos:celsius > 40]">
  <!-- XSL statements here would be applied to an element that
       contains a temperature element whose junos:celsius
       attribute is greater than 40. (The greater than symbol
       must be escaped as '&gt;' to avoid ambiguity.) -->
</xsl:template>
```

XML Tools

XML and its surrounding technologies are supported by common tools across a variety of platforms and programming languages. Finding XML-aware programmers and XML-capable tools is becoming easier. New versions of many databases and applications are supporting XML formats and features.

Stylesheet coders need not be programmers. XML IDEs provide integrated source, stylesheet, and output viewing and manipulation. XSL provides simple, compact descriptions of the tags being matched and the operations being performed on them.

New XML technologies are emerging from the W3C's working groups at great speed. The XML Query Language^[6] and XML Encryption working groups are discussing technology that could be directly applicable to the next-generation of XML-based network management.

Live Data

Data encapsulated in XML can be easily handled. Instead of using a proprietary parser to handle requests and replies, you can use any open-source XML parser. The use of common XML parser APIs allows migration of source code between parsers.

Hierarchical data lends itself to tree representations, a familiar programming construct. The W3C's document object model (DOM)^[7] provides a simple language-independent API for traversing and manipulating trees of XML elements. While DOM imposes memory constraints that limits its utility with large data sets, it is easy to learn and suitable for many tasks.

Data in XML can be exchanged between XML tools, allowing data retrieved from a device to be

- Processed by an XSLT transformation,
- Stored in an XML database,
- Rendered into a Web page,
- Compared with previously recorded datasets,
- Transformed into a report detailing any important differences, and
- E-mailed to a notification list.

Simply stated, data encoded in XML is live data and can be shared between XML-capable tools.

XML Compatibility

XML provides excellent forward and backward compatibility of data in three ways.

- First, you can introduce elements for new features without impairing an application's ability to parse old elements.

Example:

```
<foo>
  <older-feature>here</older-feature>
  <new-feature>works</new-feature>
</foo>
```

- Secondly, when an element's meaning needs to change, a device can generate two elements in parallel; the old element can contain the old value and the new element can contain the new value.

Example:

```
<foo>
  <old-style>value</old-style>
  <new-style><new-value-tag/></new-style>
</foo>
```

- Finally, applications and devices can use the document version number to detect incompatibilities and either adjust their behavior or abort the operation.

Example:

```
<foo-protocol version="5.1">
```

Supporting Environment

XML is a W3C standard and forms the center of a number of related standards and technologies. An active community of developers contributes to open-source software and lively discussions. Software vendors are rolling XML into their applications and APIs. The Apache group has several XML-related projects. Microsoft supports XML in IE5 and is embracing it with a vengeance. IBM's Alphaworks site has a number of XML-related projects. Companies are jumping on the XML bandwagon, not just to be trendy, but because they see the power that encoding data in XML gives them and their customers. Big companies (such as Oracle and SUN) and little companies (such as Neocore and XMLSPY) are all generating powerful tools for XML. Common APIs like DOM and SAX allow code to migrate between XML tool chains.

XML-based RPCs

An XML-based RPC consists of a request and the corresponding response, transmitted during a connection-oriented session using any transport protocol, including ssh, telnet, ssl, or a serial console connection.

RPC Request and Reply

A request consists of an `<rpc>` element containing a method element and any arguments for that method.

Example:

```
<!-- This RPC invokes the 'get-interface-information' method with
the interface-name of fxp0 (the management interface on a
Juniper Networks router) -->
<rpc>
  <get-interface-information>
    <interface-name>fxp0</interface-name>
    <media/>
    <brief/>
  </get-interface-information>
</rpc>

<!-- This RPC invokes the 'get-chassis-inventory' method
with no arguments -->
<rpc>
  <get-chassis-inventory/>
</rpc>

<!-- A more complex RPC -->
<rpc>
  <get-ospf-database-information>
    <!-- order of elements under an rpc does not matter -->
    <summary/> <!-- Booleans encode as empty tags -->
    <lsa-id>152.1.4.128</lsa-id> <!-- keyword value pairs -->
    <advertising-router>152.1.255.163</advertising-router>
    <area>152.14.0.0</area>
    <instance>big</instance>
  </get-ospf-database-information>
</rpc>
```

The response to an `<rpc>` is an `<rpc-reply>` tag containing any output the RPC generates.

Example:

```
<rpc-reply>
  <interface-information xmlns=".../junos-interface.dtd">
    <physical-interface>
      <name>fxp0</name>
      <admin-status junos:format="Enabled">up</admin-status>
      <oper-status>up</oper-status>
      <if-type>Ethernet</if-type>
      <link-level-type>Ethernet</link-level-type>
      <mtu>1514</mtu>
      <speed>100mbps</speed>
      <clocking>Unspecified</clocking>
      <if-device-flags>
        <ifdf-present/>
        <ifdf-running/>
      </if-device-flags>
      <if-config-flags>
        <iff-snmp-traps/>
      </if-config-flags>
    </physical-interface>
  </interface-information>
</rpc-reply>
```

Attributes given in the `<rpc>` open tag are echoed on the `<rpc-reply>` open tag, so the caller can insert additional information in the reply.

Example:

Entering the following text

```
<rpc host="my-router.foo-foo.com"
  date="Wed July 24" timing="daily">
  <get-something-interesting/>
</rpc>
```

Returns the following data.

```
<rpc-reply host="my-router.foo-foo.com"
  date="Wed July 24" timing="daily">
  <something-interesting>
    <awesome-data-here/>
  </something-interesting>
</rpc-reply>
```

Session Management

An XML-based network management session is a series of `<rpc>` elements and corresponding `<rpc-reply>` elements contained in an XML-based element. You can open sessions over any connection-oriented transport protocol, while allowing the time span of the connection to control issues like database locking and stateful operations. The session begins with the simultaneous exchange of a two-part handshake between the client and server. The first part is the XML processing instruction.

Example:

This instruction establishes the session as an XML document, with the current version of XML and the character set encoding scheme.

```
<?xml version="1.0" encoding="us-ascii"?>
```

The endpoints also exchange version and release information as attributes of the open tag for the XML session.

Example:

```
<junoscript version="1.0" release="5.0R1">
```

XML requires that a document have a single top-level tag containing the entire document contents. By placing the entire session inside the session element, you can allow either side of the session to be treated as a complete document. Additionally, each `<rpc>` or `<rpc-reply>` element can be treated as a document fragment.

The client and server use the version and release information to determine session compatibility. Typically, this decision is left to the party with the higher version number because a 1.0 client cannot deduce its capacity to interoperate with a 2.0 server. If the version difference is considered too large or drastic, the session can be aborted. If there are significant protocol or content differences, the higher numbered version party can adapt to the lower version requirements. This version-compatibility logic should never be required since all future work should always be backwards compatible, but it is laid as groundwork for unforeseen future changes.

When either party is ready for the session to be torn down, it can send a close session tag, closing the session element and terminating the connection.

Access Mechanisms

The API needs only a connection-oriented data path, making it suitable for a number of transport protocols. An `ssh`, a `telnet`, or a serial console connection can easily carry an XML-based network management session. Once a connection is established, the client uses the `xml-mode` command to convert a login session into an XML-based network management session. The client and server exchange the initial handshake, after which further communications are in XML-based RPCs.

By using existing access mechanisms, you can leverage the functionality (security, existing clients, and servers) of those mechanisms, while also letting customers use well-known and trusted protocols. The features and problems of `ssh` are well understood; the encryption and authentication schemes are well tested and in use throughout the world. An XML-specific access mechanism requires testing and experience before being considered trustworthy. In addition, using normal device login mechanisms allows support for all authentication modules currently supported under the JUNOS Internet software (RADIUS, TACACS+, local password) and all existing features built upon them (such as `allow` and `deny` command sets).

Configuration Data

The XML API presents configuration data in the same hierarchy as the native JUNOS UI, with simpler encoding rules. The UI attempts to make output more palatable, but a client needs only simple, exact rules.

The JUNOS UI uses nested braces to represent hierarchy.

Example:

```
protocols {
  bgp {
    group local {
      neighbor 10.0.0.1 {
        local-address 10.0.0.2;
      }
    }
  }
}
```

In contrast, the XML API uses hierarchies of elements.

Example:

```
<configuration>
  <protocols>
    <bgp>
      <group>
        <name>local</name>
        <neighbor>
          <name>10.0.0.1</name>
          <local-address>10.0.0.2</local-address>
        </neighbor>
      </group>
    </bgp>
  </protocols>
</configuration>
```

The entire configuration appears inside the `<configuration>` element. This element can be received as output from the `<get-configuration>` RPC and passed as input to the `<load-configuration>` RPC.

Example:

This example allows a client to download a portion of the configuration database, modify it, and reload it using the same syntax and elements.

This RPC requests a portion of the configuration using an element hierarchy under the `<configuration>` element.

```
<rpc>
  <get-configuration>
    <configuration>
      <protocols>
        <bgp/> <!-- Requesting only 'protocols bgp'
                    configuration -->
      </protocols>
    </configuration>
  </get-configuration>
</rpc>
```

The reply is given as the same element hierarchy under a `<configuration>` element.

```
<rpc-reply>
  <configuration>
    <protocols>
      <bgp>
        <!-- Complete BGP configuration here -->
      </bgp>
    </protocols>
  </configuration>
</rpc>
```

The API uses attributes to carry flags for configuration statements. Boolean attributes are encoded as `<tag flag="flag"/>` to allow for future extensions. The `inactive` and `active` attributes turn on and off the `inactive` flag on the configuration statement, effectively commenting out an entire hierarchy of statements, which is the equivalent of the JUNOS `activate` and `deactivate` commands.

The `replace` attribute functions as the `replace` tag in the JUNOS software, telling the software to remove the contents of a statement before inserting the incoming data. The `delete` attribute directs the software to remove a statement from the configuration. These four attributes can work together to generate a patch file for a configuration.

Example:

```
<configuration>
  <protocols>
    <bgp active="active"/>      <!-- Activate bgp configuration -->
    <ospf replace="replace">  <!-- Old ospf config is discarded -->
      <new-configuration-here/> <!-- Replacement configuration -->
    </ospf>
    <isis delete="delete"/>  <!-- Remove all isis configuration -->
    <pim>
      <add-this-to-pim/>      <!-- Merge in this configuration -->
    </pim>
  </protocols>
</configuration>
```

Database Locking

An XML API can provide an exclusive lock to the configuration database to avoid conflicts between scripts and human users. For example, the JUNOS software uses a commit-based configuration model, where multiple configuration changes can be committed as a group. A client script running unattended needs to be able to avoid committing partial changes being entered by users or another script.

An exclusive lock on the database is granted using the `<lock-configuration/>` RPC. The lock is granted only if no other user or client holds an exclusive lock and if there are no pending configuration changes. Users browsing the configuration database in the CLI are not forcibly removed from the configuration editor, but are unable to make changes to the database once an exclusive lock is granted. The exclusive lock remains in place until either the session is terminated or the `<unlock-configuration/>` RPC is requested.

Failure Modes

The XML API reports errors using an element that can appear at any point during the RPC, and the client must decide whether the entire `<rpc-reply>` should be discarded.

Example:

```
<junos:error>
  <source>routing</source> <!-- The routing protocol daemon -->
  <filename>incoming.conf</filename>
  <line-number>415</line-number>
  <column>16</column>
  <token>bad-value</token>
  <edit-path>protocols bgp</edit-path>
  <statement>peer-as bad-value;</statement>
  <message>Invalid value 'bad-value' for peer-as</message>
</junos:error>
```

Example Applications

The power of XML lies in your ability to manipulate the data with both specific mechanisms (like custom software looking for specific elements) and generic mechanisms (like XSLT transformations that sort, filter, mutate, re-order, and render the data).

Hardware Inventory

The `<get-chassis-inventory/>` RPC returns a complete list of hardware, organized by geographic location within the device's chassis, with a hierarchy that looks like the following.

```
<rpc-reply host="rtr1.foo-foo.com"> <!-- attributes given by caller -->
  <chassis-inventory>
    <chassis> <!-- might appear multiple times -->
      <chassis-module> <!-- might appear multiple times -->
        <name>FPC 1</name>
        <fpc>1</fpc>
        <version>REV 01</version>
        <part-number>710-001292</part-number>
        <serial-number>AE3916</serial-number>
        <chassis-sub-module> <!-- might appear multiple times -->
          <name>PIC 3</name>
          <pic>3</pic>
          <version>REV 07</version>
          <part-number>750-002303</part-number>
          <serial-number>AS3329</serial-number>
          <description>4x F/E, 100 BASE-TX</description>
        </chassis-sub-module>
      </chassis-module>
    </chassis>
  </chassis-inventory>
</rpc-reply>
```

An application would perform this RPC across a set of routers and concatenate the replies into a single XML document. This document could then be run through an XSLT transformation to convert it into a more interesting order, such as the following.

```
<inventory-list>
  <inventory-item>          <!-- might appear multiple times -->
    <part-number>750-002303</part-number>
    <serial-number>AS3329</serial-number>
    <description>4x F/E, 100 BASE-TX</description>
    <location>
      <host>rtrl.foo-foo.com</host>
      <fpc>1</fpc>
      <pic>3</pic>
      <reported>2001-07-25</reported>
    </location>
  </inventory-item>
</inventory-list>
```

You can convert this document fragment into XHTML and then display it on a Web page. Alternatively, you can convert it into another XML dialect and load it into a spreadsheet. You can save it to an XML database or into a relational database using the database's XML interface. You could also load `<inventory-list>` from yesterday's run and compared it to today's output using an XML comparison program and render that result into text to be mailed out to a notification mailing list.

Diagnostics

You might have a small set of commands to run on a device to get an overview of its status or to quickly narrow the scope of a problem. You likely must then dig through the output of those commands and separate the interesting data from the noise.

As an alternative, a diagnostic application could determine what is wrong with a device by emitting the RPC equivalent of those commands, assembling the output into a single XML document, and passing that document through an XSLT transformation. The XSLT transformation could discard the uninteresting elements and assign priorities to the remaining elements according to the design of the individual network. You could put the results into a Web page, save them to a database, or compare them to a recent diagnosis retrieved from the same database. You could also put the differences into e-mail and dispatch them to a notification mailing list. You could apply this same scheme to an entire network of devices to determine what is wrong with a network.

Vendor-neutral Configuration

One promising application for XML-based network management is configuration; nothing else offers the easy access to complex data. However, a single vendor solution to provisioning in a large multivendor networks only addresses a portion of the problem. The central issue is having a vendor-independent or vendor-neutral configuration XML Schema, and supporting XSLT transformations that translate a centralized database of information into individual configuration files or XML document fragments that can be easily loaded throughout the network.

Given the rate and style of feature implementation among the vendor community, creating a single must-fit-all configuration schema might not be realistic, but a vendor extension to translations to and from a meta configuration is an achievable goal. Using XML, XSLT, and related tool sets bring this idea from an implausible concept into the realm of possibility.

Conclusion

XML is becoming widely accepted as XML tools are quickly appearing in the industry. XML provides a simple way of encoding complex network management data using existing tools. XML-based RPCs enable you to exchange data with devices in a simple and scalable manner that is standards based and aligned with emerging technology.

Acronyms

API	application programming interface
ASCII	American Standard Code for Information Interchange
CLI	command-line interface
DOM	document object model
DTD	document type definitions
HTML	hypertext markup language
IDE	integrated development environment
MIB	Management Information Base
RADIUS	Remote Authentication Dial-in User Service
RPC	remote procedure call
RFC	Request for Comments
SAX	simple API for XML
SVG	Scalable Vector Graphics
TACACS	Terminal Access Controller Access Control System
UI	user interface
XML	extensible markup language
XSL	extensible stylesheet language
XSLT	extensible stylesheet language transformations
XHTML	extensible hypertext markup language

References

- [1] <http://www.w3c.org/TR/REC-xml>
- [2] <http://www.w3.org/TR/xmlschema-0/>
- [3] <http://www.w3c.org/TR/xslt>
- [4] <http://www.w3.org/TR/xhtml1/>
- [5] <http://www.w3.org/TR/SVG/>
- [6] <http://www.w3.org/TR/xquery/>
- [7] <http://www.w3.org/TR/REC-DOM-Level-1>

For more information, see www.juniper.net or contact your nearest Juniper Networks sales representative.

Copyright © 2001, Juniper Networks, Inc. All rights reserved. Juniper Networks is registered in the U.S. Patent and Trademark Office and in other countries as a trademark of Juniper Networks, Inc. Internet Processor, Internet Processor II, JUNOS, JUNOScript, M5, M10, M20, M40, and M160 are trademarks of Juniper Networks, Inc. All other trademarks, service marks, registered trademarks, or registered service marks are the property of their respective owners.

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.