



Nexpose 5.2 API v1.1 Guide

Copyright © 2012 Rapid7 LLC. Boston, Massachusetts, USA. All rights reserved. Rapid7 and Nexpose are trademarks of Rapid7, LLC. Other names appearing in this content may be trademarks of their respective owners.

Revision history

Revision Date	Description
December 19, 2006	Adding to source control.
December 20, 2006	Fixed several typos.
April 16, 2007	Formatting, copyrights and NDA adjustments. APM and Fix the Report Save Request.
June 26, 2007	Added RTF format information to the Report Adhoc Generate Request and the Report Config DTD listing.
June 27, 2007	Removed unsupported API functions for device details.
August 8, 2007	Added "vuln-potential" vuln status to ScanSummary DTD
November 6, 2007	For the SiteSaveRequest the element "adminCredentials" now requires the service attribute.
May 1, 2008	Update the Site DTD Hosts element. Add a note to the ScanConfig DTD describing how severityThreshold is currently implemented in Nexpose. Removed the "status" attribute from the Engine-ActivityResponse DTD. Removed the ScanStartRequest and the AdhocScanConfig DTD. Added additional attributes that are part of the VulnerabilityListingResponse. Removed all references to ReportSectionDefListing and ReportDbDefListing. Add the "property" element to the ReportSections in the ReportTemplate DTD and added the "name" attribute to the ReportSection element. Removed ScanTriggers from the ScanConfig DTD. Add Hosts information to the SiteDevicesScanRequest.
May 6, 2008	Updated the Nexpose version on the cover page. Added additional attributes that are part of the VulnerabilityDetailsResponse.
May7, 2008	Added riskscore to the SiteListingRequest and SiteDeviceListingRequest. Updated the SiteSummary DTD to reflect the fact that Nexpose stores the site riskscore as a computed value. Added a note in the SiteSummary DTD about how the riskscore is computed. Added comments in the VulnerabilityDetailsResponse and the VulnerabilityListingResponse DTDs mentioning the date format of the "added", "modified", and "published" attributes as ISO 8601.
August 13, 2008	Added qualys-xml attribute to ReportConfig DTD. Added XML Response DTD and Failure DTD to Appendix. Changed "Nexpose 4.6" references to "Nexpose 4.7". Edited text for grammar, tone, and typos.
August 26, 2008	Added section on system management and diagnostic functions. Replaced missing DTD references in command samples where necessary.
August 28, 2008	Corrected formatting for system management and diagnostic function commands.
September 25, 2008	Added device DTD and Email DTD to the appendix. Removed ScanTemplate DTD from the appendix because no API calls refer to it. Edited various API calls and DTDs.
October 2, 2008	Updated description for SendLog command. Annotated EngineSummary DTD.
October 7, 2008	Edited timezone and owner attributes in ReportConfig DTD.
October 14, 2008	Updated the SystemInformationRequest call.

Revision Date	Description
November 11, 2008	Updated the SystemInformationResponse call to include new thread variables.
March 30, 2009	Added Pause/Resume scheduling command
April 14, 2009	Updated values for ReportConfigSummary status attributed in ReportConfigSummary DTD.
June 18, 2009	Added site-id attribute to ScanSummary DTD.
November 6, 2009	Added sections about user management calls and error responses.
December 18, 2009	Changed version number convention and included comparison of API v1.1 and API v1.2.
January 4, 2010	Fixed typos.
February 3, 2010	Added information about multi-tenancy user and silo provisioning APIs.
June 15, 2010	Added information about API applications and requirements.
August 9, 2010	Added Code Samples section.
January 17, 2011	Expanded ReportConfig and Email DTDs with information for adding user access lists to reports.
March 14, 2011	Added notes about new features in Vulnerability Assessment APIs that are now available in API Extended v1.2.
July 11, 2011	Corrected inaccuracies throughout the document.
November 15, 2011	Changed character case usage for word "Nexpose" to reflect current branding.
March 21, 2012	Added information about vulnerability filter attribute in report-related APIs.

Contents

Revision history	3
Overview	7
Nexpose Architecture	9
Introduction.....	9
Sites.....	9
Distributed scan engines	10
Asset groups.....	11
Web console	12
Scanning	12
Reporting	13
Management and diagnostic functions	14
API functions	15
API overview.....	15
Lists of all commands in the Nexpose API	17
Session Management.....	20
Site Management	22
Device Management	29
Asset Group Management	29
Scanning	32
Vulnerability Assessment.....	38
Reporting	42
User management functions.....	52
General management and diagnostic functions	56
Error responses	60
Error responses for requests for non-existent API functions	61
Error responses common to all valid requests	61
Code samples	63
Fundamental API sequence	63
Login implementation	66
User creation implementation.....	67
Site creation implementation	68
Site listing implementation.....	69
Site scan implementation.....	70
Vulnerability details implementation.....	71
Ad hoc report generation implementation	72
Logout implementation.....	74

Appendix A: DTD Listings..... 75

- Device DTD..... 75
- SiteSummary DTD 76
- Site DTD..... 77
- AssetGroupSummary DTD 80
- AssetGroup DTD..... 81
- EngineSummary DTD..... 81
- ScanConfig DTD 82
- ScanSummary DTD 83
- ReportTemplateSummary DTD 84
- ReportTemplate DTD..... 85
- ReportConfigSummary DTD..... 86
- ReportConfig DTD 87
- Email DTD 90
- ReportSummary DTD 91
- UserConfig DTD..... 92
- User Site DTD..... 93
- User Group DTD 93
- UserSummary DTD..... 94
- AuthenticatorSummary DTD..... 95
- XMLResponse DTD..... 95
- Failure DTD..... 96

Overview

Nexpose is a unified vulnerability solution that scans networks to identify the devices running on them and to probe these devices for vulnerabilities. It analyzes the scan data and processes it for reports. You can use these reports to help you assess your network security at various levels of detail and remediate any vulnerabilities quickly.

The vulnerability checks in Nexpose identify security weaknesses in all layers of a network computing environment, including operating systems, databases, applications, and files. Nexpose can detect malicious programs and worms, identify areas in your infrastructure that may be at risk for an attack, and verify patch updates and security compliance measures.

Nexpose APIs provide programming access to commonly used functions. You can also use the API to automate internal auditing and security activities, and integrate your environment with third-party products. There are currently two categories of APIs. They are API v1.1 and API v1.2.

Nexpose API v1.1

The Nexpose API 1.1 is available in Nexpose 4.0 or later and is broken down into the following functional categories:

- session management
- site management
- scanning

NOTE: The API does not support scan template creation.

- asset, or device, management
- asset group management
- vulnerability assessment
- most reporting functions

NOTE: An additional reporting function is available in API v1.2.

The requests made to the Nexpose API 1.1 are validated with DTDs documented in this guide.

Nexpose API v1.2

The Nexpose Extended API 1.2 provides extended functionality available in Nexpose 4.8 or later.

It is broken down into the following functional categories:

- scan engine management
- report listing

NOTE: All other reporting functions are available in API v1.1.

- ticketing
- multi-tenant user management
- silo profile management
- silo management
- role management
- scan engine pool management

NOTE: API 1.1 Session Management is required for API 1.2 functions.

The requests made to the Nexpose API 1.2 are documented in the *Nexpose API v1.2 Guide* and can be validated with the XML schemas provided in the package `Nexpose_Extended_API_XMLSchemas_v1.2.zip`.

You can download all documentation and schemas from the *Support* page in Help.

Nexpose Architecture

Understanding the Nexpose architecture will help you make to make the best use of the functions in the API.

Introduction

The Nexpose system consists of two main components: scan engines and a security console. One or more Nexpose Scan Engines (NSEs) search networks to discover devices and the processes running on them, such as operating systems, programs, and databases. The scan engines then test discovered assets for vulnerabilities, patches, and other security-related factors. A Nexpose Security Console collects, analyzes, and stores the scan data, and it generates reports and vulnerability remediation procedures. Additionally, the console controls the scan engines and provides a Web-accessible user interface for managing all Nexpose functions.

An organization can deploy scan engines within its network or outside its firewall. It also can use Hosted Scanning Engines that are located in Rapid7 data centers.

The simplest Nexpose configuration consists of a single scan engine and the security console on one host.

Sites

A site is a logical group of assets assembled for a scan by a specific, dedicated scan engine. The grouping principle may be something meaningful to you, such as a common geographic location or a range of IP addresses. Or, you may organize a site for a specific type of scan.

For example, a company sets up Nexpose in a Boston location. The Nexpose Global Administrator, whose logon name is corp_admin wants to scan two sets of assets at different times and with different scanning parameters. So, he sets up two sites:

- **BOS_Servers** includes Web and database servers.
- **BOS_Workstations** includes the workstations.

The global administrator is in charge of scanning both sites.

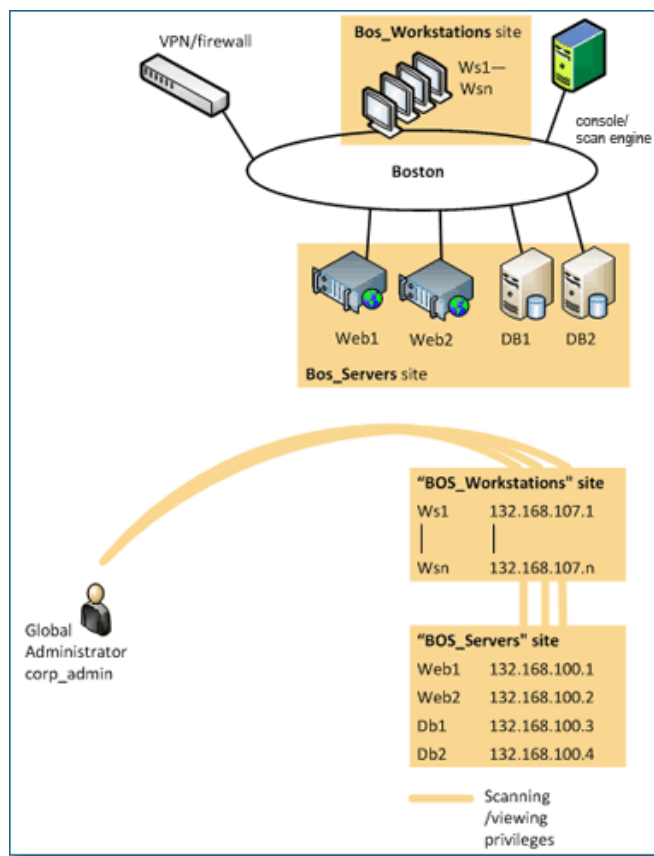


Figure 1: The initial implementation with two sites

For more information about setting up sites and asset groups, see the *Nexpose Administrator's Guide*, which you can download from the *Support* page of Help.

Distributed scan engines

Distributing multiple scan engines promotes fault tolerance and improves scanning performance while conserving bandwidth. It is a best practice to deploy at least one scan engine at each physical location, where it can scan assets locally. This frees up bandwidth for more remote connections. Also, installing scan engines locally, behind firewalls, removes the need for firewall rule exceptions.

Agentless operation

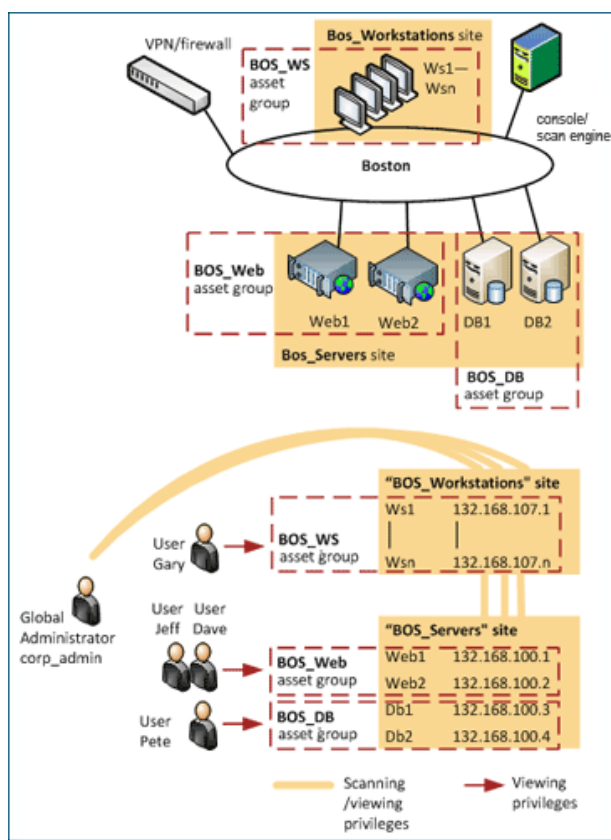
Nexpose scans exclusively over the network, using common Windows and UNIX network protocols to gain access to systems. It does not require agent software to be installed on the assets targeted for scanning. Agentless architecture lowers the total cost of ownership (TCO) of Nexpose and avoids potential security and stability issues associated with agents.

Asset groups

An asset group is a collection of assets, but unlike a site, it is not defined for scanning. An asset group typically is assigned to a nonadministrative Nexpose user, who views scan reports about that group in order to perform any necessary remediation.

Using the example of the Boston company in the *Sites* section, the global administrator, who has control of the entire Nexpose system, wants to delegate teams for remediating vulnerabilities on the Web servers, database servers, and workstations. So, he creates three asset groups.

- BOS_Web includes the two Web servers. Two nonadministrative users, Jeff and Dave, who handle Web server maintenance and troubleshooting at the Boston location, have access to this group.
- BOS_DB includes the two database servers. A nonadministrative user, Pete, who is a database manager, has access to this group.
- BOS_WS includes all workstations. A nonadministrative user, Gary, who troubleshoots the workstations, has access to this group.



For more information about setting up sites and asset groups, see the *Nexpose Administrator's Guide*, which you can download from the *Support* page of Help.

Web console

Each scan engine is controlled by a security console, which can be located anywhere on the network. The console communicates with the engines via encrypted SSL sessions over a defined Transmission Control Protocol (TCP) port. Engines talk only to the console, they do not talk to other engines.

In order to manage scans and view results, users log on to the security console interface using a Web browser over HTTPS (secure encrypted HTTP). The only software required for using the console is a Web browser.

User access control

The security console requires users to log on with a Nexpose user name and password. This authentication occurs over HTTPS, so it is entirely encrypted. The authentication database is stored in an encrypted format locally on the console server. Nexpose does not store or transmit passwords in plaintext.

Upon logging on to Nexpose, a user sees only information to which he or she has been granted access by a Nexpose Global Administrator. A given user can have access to one or more entire sites, one or more assets within a site, or one or more asset groups. The Global Administrator can control access to sensitive security information by granting fine-grained, “need-to-know” user permissions.

Scanning

A scan includes one or more of the following phases

Phase	What the scan engine does in this phase
Device discovery	Locates active devices on the network.
Service discovery	Determines the types of services running on devices found to be active on the network.
Access discovery	Scans active devices to determine configurations, including operating system, hardware, service and installed software.
Vulnerability assessment	Scans active devices for known vulnerabilities.

Device discovery

In device discovery, the first phase of a scan, the scan engine maps out the network and locates the active assets.

The scan engine can discover devices using ICMP echo requests, or by sending TCP packets to one or more ports in what is effectively a mini port scan. Systems responding to these packets are marked as active and will be included in subsequent scan phases.

You may wish to disable device discovery when scanning assets in a DMZ or any other area with strict protection, such as a firewall that drops blocked packets. When you disable device discovery, Nexpose uses port scan results found in the discovery phase to determine which hosts are active. If any ports are found to be open on an asset, Nexpose will mark that asset “alive.”

Service Discovery

In the service discovery phase the scan engine maps out the network services running on the active assets.

You can tune service discovery to enable or disable TCP and User Datagram Protocol (UDP) port scans. You can specify which ports to scan, including default port lists or all possible ports (1–65,535). Additionally, you can change the method of TCP port scanning to use full connections, half-open (SYN) scans, or other variations.

Once Nexpose determines a port to be open, it performs a protocol handshake on that port to verify the type of service running on it. Doing so allows Nexpose to determine if a service is running, even if it is not on the expected port. For example, an HTTP server may be running on port 1234, as opposed to the standard HTTP port 80.

Asset inventory

Once Nexpose knows the network layout with active assets and services, it can perform an asset inventory to determine the configuration of many system components:

- operating system type and version (for example, Microsoft Windows XP SP2)
- system configuration
- hardware type (for example, Cisco 2621)
- service type and version (for example, Apache 2.0.54)
- service configuration
- installed software (for example, Mozilla Firefox 1.0.5)
- software configuration

Vulnerability assessment

In the vulnerability assessment phase, Nexpose scans active devices for known vulnerabilities.

Nexpose vulnerability checks cover known vulnerabilities in a broad range of products. Additionally, Nexpose's Web spidering feature can discover vulnerabilities caused by Web application developers. The spider can search a Web site for common programming errors and backup copies of scripts that may divulge sensitive information.

You can specify certain vulnerabilities or vulnerability types for discovery. Nexpose includes default scan templates with predefined vulnerability check settings. You also can custom-define your own vulnerability checks.

Reporting

You can create reports based on Nexpose scan data in PDF, HTML, XML, and plain text formats. Nexpose also can export data to most database systems or to structured file formats, such as XML, QualysXML, and CSV.

Configuring a report involves several steps:

- selecting a report template
- specifying sites, asset groups, or assets to include in the report
- selecting delivery options, such as e-mail to all authorized users
- scheduling when to generate the report

You can use Nexpose default report templates, which include predefined settings for level of technical data, specific information for certain compliance audits, export format, and other features. See the *Nexpose Reporting Guide* for sample reports and export formats.

You also can create custom report templates.

Report sections

Each Nexpose report template consists of sections that include specific types of information. When you create a custom report, you can choose from a list of sections to generate information exactly according to your needs. Examples of report sections include *Discovered System Information*, *Discovered Vulnerabilities*, *Risk Assessment*, and *Remediation Plan*.

See the user's guide for a complete list of report sections, including descriptions and visual samples.

Management and diagnostic functions

You can use the Nexpose logging and system reporting functions to monitor internal activity and troubleshoot problems. Additionally, you can induce Nexpose to restart and to obtain required software updates when necessary.

API functions

The Nexpose API provides access to a subset of the full feature set that is available in the security console Web interface. The range of API access depends on the user privileges assigned to the logon credentials.

You may access the API using encrypted Hypertext Transfer Protocol over a Secure Socket Layer connection. The API supports HTTP 1.0 and 1.1 syntax. For data exchange, you may use the Extensible Markup Language (XML) as defined by the W3C (<http://www.w3.org/TR/REC-xml>).

API overview

You access the API through a URL of the form:

```
https://<host>:<port>/api/api-version/xml
```

The application connecting to Nexpose must use HTTPS to engage the console. The application must then log on with valid Nexpose credentials. Upon successful logon, Nexpose returns a session ID to the application. Use the session ID for subsequent requests rather than resubmitting the credentials. The following is a typical login sequence:

1. Open an HTTPS connection to the Web console, usually on port 3780.
1. Construct a LoginRequest XML request containing valid Nexpose credentials.
2. Verify that the Content-type HTTP header is set to “text/xml”.
3. Send the XML request to `https://<host>:<port>/api/1.1/xml` using HTTP POST Method.
4. Parse the returned LoginResponse.
5. If the success attribute is set to 1, extract the session-id attribute for use in subsequent requests.
6. If the success attribute is set to 0, extract the Failure information and report it.

The session-id is subject to timeout from inactivity regardless of how much work Nexpose is performing. You can specify the timeout period on the *Nexpose Security Console Configuration* page of the Web interface. See the *Nexpose Administrator's Guide* for details.

All subsequent requests must include the appropriate session-id in their respective request XML structure. This inclusion will allow the API program to perform actions on behalf of the credentials specified.

If the API request results in a failure, the response XML document will have the success attribute set to 0 and the Failure element will be returned. The format of the Failure element is as follows:

```
<!-- The failure description, consisting of one or more message and/or exception -->
<!ELEMENT Failure ((message|Exception)*)>

<!-- the message describing the failure -->
<!ELEMENT message (#PCDATA)>
  <!-- the source of the message, such as the module that caused the error -->
  <!ATTLIST message source CDATA #IMPLIED>
  <!-- the source specific message code -->
  <!ATTLIST message code CDATA #IMPLIED>

<!-- the exception causing the failure -->
<!ELEMENT Exception (message, stacktrace?)>
  <!-- the name of the Exception class (for Java or C++ exceptions) -->
  <!ATTLIST Exception name CDATA #IMPLIED>
<!ELEMENT stacktrace (#PCDATA)>
```

As the success and failure information is stored within the returned XML document, all requests processed by the Nexpose API will return HTTP status code 200. Any other status code implies a problem on the Nexpose server. Common causes of server errors include an older version of Nexpose that do not have API support built-in, out of memory conditions, etc.

If you use a command that is not listed in the in *Nexpose Administrator's Guide*, Nexpose will return the XMLResponse.

For a sample implementation of some of the API functionality, see the *Code samples* section.

Lists of all commands in the Nexpose API

Session management commands

Command	Description
Login	Log on to the security console and establish a session.
Logout	Log off from the security console, freeing the session and all related resources.

Site management commands

Command	Description
SiteListing	Provide a list of all sites the user is authorized to view or manage.
SiteConfig	Provide the configuration of the site, including its associated assets.
SiteSave	Save changes to a new or existing site.
SiteDelete	Delete the specified site and all associated scan data.
SiteScan	Scan the specified site.
SiteScanHistory	Provide a list of all previous scans of the site.
SiteDeviceListing	Provide a list of all of the assets in a site. If no site-id is specified, then this will return all of the assets for the scan engine, grouped by site-id.
SiteDevicesScan	Scan a specified subset of site assets.

Asset management commands

Command	Description
DeviceDelete	Delete the specified asset.

Asset group management commands

Command	Description
AssetGroupListing	Provide a list of all asset groups the user is authorized to view or manage.
AssetGroupConfig	Provide the configuration of the asset group, including its associated devices.
AssetGroupSave	Save changes to a new or existing asset group.
AssetGroupDelete	Delete the specified asset group and all associated scan data.

Scan commands

NOTE: The API does not support scan template creation.

Command	Description
EngineListing	Provide a list of all scanning engines managed by the security console.
EngineActivity	Provide a list of current scan activities for a specific scan engine.
ScanActivity	Provide a list of current scan activities across all scan engines managed by the security console.
ScanPause	Pause a running scan.
ScanResume	Resume a running scan.
ScanStop	Stop a running scan.
ScanStatus	Check the current status of a scan.
ScanStatistics	Get scan statistics, including node and vulnerability breakdowns.

Vulnerability assessment commands

Command	Description
VulnerabilityListing	Provide a list of vulnerabilities checked by Nexpose. NOTE: Additional features for this API are now available in API Extended v1.2. Refer to the Nexpose <i>Extended API v1.2 Guide</i> for more information.
VulnerabilityDetails	Provide the full details of a vulnerability, including its description, cross-references, and solution. NOTE: Additional features for this API are now available in API Extended v1.2. Refer to the Nexpose <i>Extended API v1.2 Guide</i> for more information.

Reporting commands

Command	Description
ReportTemplateListing	Provide a list of all report templates the user can access on the security console.
ReportTemplateConfig	Retrieve the configuration for a report template.
ReportTemplateSave	Save the configuration for a report template.
ReportListing	Provide a listing of all report definitions the user can access on the security console.
ReportHistory	Provide a history of all reports generated with the specified report definition.
ReportConfig	Retrieve the configuration for a report definition.
ReportSave	Save the configuration for a report definition.
ReportGenerate	Generate a new report using the specified report definition.
ReportDelete	Delete a previously generated report or report definition.
ReportAdhocGenerate	Generate a report once using a simple configuration, and send it back in a multipart mime response.

User management commands

Command	Description
UserListing	Provide a list of user accounts and information about those accounts.
UserAuthenticator	Provide a list of user authentication sources.
UserConfig	List information about a given user account.
UserSave	Create a new user account, or update the settings for an existing account.
UserDelete	Delete a user account. Note that you cannot delete a user account that is associated with reports or tickets.

General management and diagnostic commands

Command	Description
ConsoleCommand	Execute an arbitrary Nexpose console command that is supplied as text via an API parameter. The Nexpose console commands are documented in the Nexpose <i>Administrator's Guide</i> . If you use a command that is not listed in the in Nexpose <i>Administrator's Guide</i> , Nexpose will return the XMLResponse.
SystemInformation	Obtain Nexpose system data, such as total RAM, free RAM, total disk space, free disk space, CPU speed, number of CPU cores, and other vital information.
StartUpdate	Induce Nexpose to retrieve required updates and restart if necessary.
Restart	Induce Nexpose to restart.
SendLog	Output diagnostic information into log files, zip the files, and encrypt the archive with a PGP public key that is provided as a parameter for the API call. Then, either e-mail this archive to an address that is specified as an API parameter, or upload the archive using HTTP or HTTPS to a URL that is specified as an API parameter. If you do not specify a key, the SendLogRequest uses a Rapid7 default key.

Session Management

Login

Log on to the security console and establish a session.

LoginRequest

If no silo-id is specified, the user's silo will be set to the user's default silo, if it exists. If the silo-id is not specified, and no silos are defined for the user, then the login fails, unless the user is a super-user.

```
<!DOCTYPE LoginRequest [  
<!ELEMENT LoginRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST LoginRequest sync-id CDATA #IMPLIED>  
  <!-- the user id to login with -->  
  <!ATTLIST LoginRequest user-id CDATA #REQUIRED>  
  <!-- the password to login with -->  
  <!ATTLIST LoginRequest password CDATA #REQUIRED>  
  <!-- the silo to log into -->  
  <!ATTLIST LoginRequest silo-id CDATA #IMPLIED>  
>
```

LoginResponse

```
<!DOCTYPE LoginResponse [  
<!ELEMENT LoginResponse (Failure?)>  
  <!-- the session id to be used with all subsequent requests -->  
  <!ATTLIST LoginResponse session-id CDATA #REQUIRED>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST LoginResponse success (0|1) #REQUIRED>  
>
```

Logout

Log off from the security console, freeing the session and all related resources.

LogoutRequest

```
<!DOCTYPE LogoutRequest [  
<!ELEMENT LogoutRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST LogoutRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST LogoutRequest session-id CDATA #REQUIRED>  
>
```

LogoutResponse

```
<!DOCTYPE LogoutResponse [  
<!ELEMENT LogoutResponse (Failure?)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST LogoutResponse success (0|1) #REQUIRED>  
>
```

Site Management

SiteListing

Provide a list of all sites the user is authorized to view or manage.

SiteListingRequest

```
<!DOCTYPE SiteListingRequest [  
<!ELEMENT SiteListingRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST SiteListingRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST SiteListingRequest session-id CDATA #REQUIRED>  

```

SiteListingResponse

```
<!DOCTYPE SiteListingResponse [  
<!ELEMENT SiteListingResponse (Failure|SiteSummary*)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST SiteListingResponse success (0|1) #REQUIRED>  
<!-- See the SiteSummary DTD for more details -->  

```

SiteConfig

Provide the configuration of the site, including its associated assets.

SiteConfigRequest

```
<!DOCTYPE SiteConfigRequest [  
<!ELEMENT SiteConfigRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST SiteConfigRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  

```

SiteConfigResponse

```
<!DOCTYPE SiteConfigResponse [  
<!ELEMENT SiteConfigResponse (Failure|Site)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST SiteConfigResponse success (0|1) #REQUIRED>  
  
  <!-- See the Site DTD for more details -->  
  

```

SiteSave

Save changes to a new or existing site.

SiteSaveRequest

```
<!DOCTYPE SiteSaveRequest [  
<!ELEMENT SiteSaveRequest (Site)>  
  <!-- user-defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST SiteSaveRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  

```

SiteSaveResponse

```
<!DOCTYPE SiteSaveResponse [  
<!ELEMENT SiteSaveResponse (Failure?)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST SiteSaveResponse success (0|1) #REQUIRED>  
  <!-- the newly assigned site ID (unchanged for existing sites) -->  

```

SiteDelete

Delete the specified site and all associated scan data.

If you have a scan in progress or a paused scan, you cannot delete the site in which that scan was initiated. If you send SiteDeleteRequest with a paused or in-progress scan, Nexpose will return an error response. For more information, see *Error responses for SiteDelete*.

It is a best practice to send SiteScanHistoryRequest first to determine if any scans are paused or running. See *SiteScanHistory* (on page 26).

To stop a paused or running scan, send ScanStopRequest. See *ScanStop* (on page 36).

When you are certain that no scans are running or paused, send SiteDeleteRequest.

SiteDeleteRequest

```
<!DOCTYPE SiteDeleteRequest [
<!ELEMENT SiteDeleteRequest EMPTY>
  <!-- user defined synchronization token id used to avoid duplicate requests -->
  <!ATTLIST SiteDeleteRequest sync-id CDATA #IMPLIED>
  <!-- the current session id -->
  <!ATTLIST SiteDeleteRequest session-id CDATA #REQUIRED>
  <!-- the ID of the site to delete -->
  <!ATTLIST SiteDeleteRequest site-id CDATA #REQUIRED>
]>
```

SiteDeleteResponse

```
<!DOCTYPE SiteDeleteResponse [
<!ELEMENT SiteDeleteResponse (Failure?)>
  <!-- set to 1 upon success, 0 otherwise -->
  <!ATTLIST SiteDeleteResponse success (0|1) #REQUIRED>
]>
```

SiteScan

Scan the specified site.

SiteScanRequest

```
<!DOCTYPE SiteScanRequest [  
<!ELEMENT SiteScanRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST SiteScanRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST SiteScanRequest session-id CDATA #REQUIRED>  
  <!-- the ID of the site to scan -->  
  <!ATTLIST SiteScanRequest site-id CDATA #REQUIRED>  

```

SiteScanResponse

```
<!DOCTYPE SiteScanResponse [  
<!ELEMENT SiteScanResponse (Failure|(Scan+))>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST SiteScanResponse success (0|1) #REQUIRED>  
<!ELEMENT Scan EMPTY>  

```

SiteScanHistory

Provide a list of all previous scans of the site.

SiteScanHistoryRequest

```
<!DOCTYPE SiteScanHistoryRequest [  
  <!ELEMENT SiteScanHistoryRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST SiteScanHistoryRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST SiteScanHistoryRequest session-id CDATA #REQUIRED>  
  <!-- the ID of the site to retrieve the scan history for -->  
  <!ATTLIST SiteScanHistoryRequest site-id CDATA #REQUIRED>  
>
```

SiteScanHistoryResponse

```
<!DOCTYPE SiteScanHistoryResponse [  
  <!ELEMENT SiteScanHistoryResponse (Failure|ScanSummary*)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST SiteScanHistoryResponse success (0|1) #REQUIRED>  
  
  <!-- See the ScanSummary DTD for more details -->  
>
```

SiteDeviceListing

Provide a list of all of the assets in a site. If no site-id is specified, then this will return all of the assets for the scan engine, grouped by site-id.

SiteDeviceListingRequest

```
<!DOCTYPE SiteDeviceListingRequest [  
<!ELEMENT SiteDeviceListingRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST SiteDeviceListingRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST SiteDeviceListingRequest session-id CDATA #REQUIRED>  
  <!-- the ID of the site to retrieve the device listing for -->  
  <!ATTLIST SiteDeviceListingRequest site-id CDATA #IMPLIED>  
>
```

SiteDeviceListingResponse

```
<!DOCTYPE SiteDeviceListingResponse [  
<!ELEMENT SiteDeviceListingResponse (Failure|SiteDevices*)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST SiteDeviceListingResponse success (0|1) #REQUIRED>  
<!ELEMENT SiteDevices (device*)>  
  <!-- See the device DTD for more details -->  
  <!ATTLIST SiteDevices site-id CDATA #REQUIRED>  
>
```

SiteDevicesScan

Scan a specified subset of site assets.

SiteDevicesScanRequest

```
<!DOCTYPE SiteDevicesScanRequest [  
  <!ELEMENT SiteDevicesScanRequest (Devices?,Hosts?)>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST SiteDevicesScanRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST SiteDevicesScanRequest session-id CDATA #REQUIRED>  
  <!-- the ID of the site whose devices are to be scanned -->  
  <!ATTLIST SiteDevicesScanRequest site-id CDATA #REQUIRED>  
  
  <!ELEMENT Devices (device+)>  
  <!-- See the device DTD for more details -->  
  
  <!ELEMENT Hosts (range|hosts)+>  
  <!-- IPv4 address range of the form 10.0.0.1 -->  
  <!ELEMENT range EMPTY>  
    <!ATTLIST range from CDATA #REQUIRED>  
    <!ATTLIST range to CDATA #IMPLIED>  
  <!-- named host (usually DNS or Netbios name -->  
  <!ELEMENT host (#PCDATA)>  
]>
```

SiteDevicesScanResponse

```
<!DOCTYPE SiteDevicesScanResponse [  
  
  <!ELEMENT SiteDevicesScanResponse (Failure|(Scan+))>  
    <!-- set to 1 upon success, 0 otherwise -->  
    <!ATTLIST SiteDevicesScanResponse success (0|1) #REQUIRED>  
  <!ELEMENT Scan EMPTY>  
    <!-- the scan ID, upon successful start -->  
    <!ATTLIST Scan scan-id CDATA #REQUIRED>  
    <!-- the engine the scan was dispatched to -->  
    <!ATTLIST Scan engine-id CDATA #REQUIRED>  
]>
```

Device Management

DeviceDelete

Delete the specified asset.

DeviceDeleteRequest

```
<!DOCTYPE DeviceDeleteRequest [  
<!ELEMENT DeviceDeleteRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST DeviceDeleteRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST DeviceDeleteRequest session-id CDATA #REQUIRED>  
  <!-- the id of the device to remove -->  
  <!ATTLIST DeviceDeleteRequest device-id CDATA #IMPLIED>  
>
```

DeviceDeleteResponse

```
<!DOCTYPE DeviceDeleteResponse [  
<!ELEMENT DeviceDeleteResponse (Failure?)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST DeviceDeleteResponse success (0|1) #REQUIRED>  
>
```

Asset Group Management

AssetGroupListing

Provide a list of all asset groups the user is authorized to view or manage.

AssetGroupListingRequest

```
<!DOCTYPE AssetGroupListingRequest [  
<!ELEMENT AssetGroupListingRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST AssetGroupListingRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST AssetGroupListingRequest session-id CDATA #REQUIRED>  
>
```

AssetGroupListingResponse

```
<!DOCTYPE AssetGroupListingResponse [  
<!ELEMENT AssetGroupListingResponse (Failure|AssetGroupSummary*)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST AssetGroupListingResponse success (0|1) #REQUIRED>  
<!-- See the AssetGroupSummary DTD for more details -->  
>
```

AssetGroupConfig

Provide the configuration of the asset group, including its associated devices.

AssetGroupConfigRequest

```
<!DOCTYPE AssetGroupConfigRequest [  
<!ELEMENT AssetGroupConfigRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST AssetGroupConfigRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST AssetGroupConfigRequest session-id CDATA #REQUIRED>  
  <!-- the ID of the group to retrieve the config for -->  
  <!ATTLIST AssetGroupConfigRequest group-id CDATA #REQUIRED>  
>
```

AssetGroupConfigResponse

```
<!DOCTYPE AssetGroupConfigResponse [  
<!ELEMENT AssetGroupConfigResponse (Failure|AssetGroup)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST AssetGroupConfigResponse success (0|1) #REQUIRED>  
  
<!-- See the AssetGroup DTD for more details -->  
>
```

AssetGroupSave

Save changes to a new or existing static asset group.

AssetGroupSaveRequest

```
<!DOCTYPE AssetGroupSaveRequest [  
<!ELEMENT AssetGroupSaveRequest (AssetGroup, Failure?)>  
  <!-- user-defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST AssetGroupSaveRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST AssetGroupSaveRequest session-id CDATA #REQUIRED>  
  <!-- each asset group defines assets within it -->  
  <!ELEMENT AssetGroup (Devices)>  
  <!-- set to -1 to create a new group, or a positive integer to update an existing  
group -->  
  <!ATTLIST AssetGroup id CDATA #REQUIRED>  
  <!ATTLIST AssetGroup name CDATA #REQUIRED>  
  <!ATTLIST AssetGroup description CDATA #IMPLIED>  
  <!-- container element for asset inclusions -->  
  <!ELEMENT Devices (device+)>  
  <!ELEMENT device EMPTY>  
  <!-- the identifier of the asset to include in the group -->  
  <!ATTLIST device id CDATA #REQUIRED>  
>
```

AssetGroupSaveResponse

```
<!DOCTYPE AssetGroupSaveResponse [  
<!ELEMENT AssetGroupSaveResponse (Failure?)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST AssetGroupSaveResponse success (0|1) #REQUIRED>  
  <!-- the newly assigned group ID (unchanged for existing groups) -->  
  <!ATTLIST AssetGroupSaveResponse group-id CDATA #REQUIRED>  
>
```

AssetGroupDelete

Delete the specified asset group and all associated scan data.

AssetGroupDeleteRequest

```
<!DOCTYPE AssetGroupDeleteRequest [  
<!ELEMENT AssetGroupDeleteRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST AssetGroupDeleteRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST AssetGroupDeleteRequest session-id CDATA #REQUIRED>  
  <!-- the ID of the group to delete -->  
  <!ATTLIST AssetGroupDeleteRequest group-id CDATA #REQUIRED>  
>
```

AssetGroupDeleteResponse

```
<!DOCTYPE AssetGroupDeleteResponse [  
<!ELEMENT AssetGroupDeleteResponse (Failure?)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST AssetGroupDeleteResponse success (0|1) #REQUIRED>  
>
```

Scanning

EngineListing

Provide a list of all scanning engines managed by the security console.

EngineListingRequest

```
<!DOCTYPE EngineListingRequest [  
<!ELEMENT EngineListingRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST EngineListingRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST EngineListingRequest session-id CDATA #REQUIRED>  
>
```

EngineListingResponse

```
<!DOCTYPE EngineListingResponse [  
<!ELEMENT EngineListingResponse (Failure|EngineSummary*)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST EngineListingResponse success (0|1) #REQUIRED>  
  
  <!-- See the EngineSummary DTD for more details -->  
  
>
```

EngineActivity

Provide a list of current scan activities for a specific scan engine.

EngineActivityRequest

```
<!DOCTYPE EngineActivityRequest [  
<!ELEMENT EngineActivityRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST EngineActivityRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST EngineActivityRequest session-id CDATA #REQUIRED>  
  <!-- the id of the engine to query -->  
  <!ATTLIST EngineActivityRequest engine-id CDATA #REQUIRED>  
>
```

EngineActivityResponse

```
<!DOCTYPE EngineActivityResponse [  
<!ELEMENT EngineActivityResponse (Failure|ScanSummary*)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST EngineActivityResponse success (0|1) #REQUIRED>  
  <!-- current status of the scan engine -->  
  
  <!-- See the ScanSummary DTD for more details -->  
  
>
```

ScanActivity

Provide a list of current scan activities across all scan engines managed by the security console.

ScanActivityRequest

```
<!DOCTYPE ScanActivityRequest [  
<!ELEMENT ScanActivityRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST ScanActivityRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST ScanActivityRequest session-id CDATA #REQUIRED>  
>
```

ScanActivityResponse

```
<!DOCTYPE ScanActivityResponse [  
<!ELEMENT ScanActivityResponse (Failure|ScanSummary*)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST ScanActivityResponse success (0|1) #REQUIRED>  
  
  <!-- See ScanSummary DTD for more details -->  
>
```

ScanPause

Pause a running scan.

ScanPauseRequest

```
<!DOCTYPE ScanPauseRequest [  
<!ELEMENT ScanPauseRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST ScanPauseRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST ScanPauseRequest session-id CDATA #REQUIRED>  
  <!-- the ID of the scan -->  
  <!ATTLIST ScanPauseRequest scan-id CDATA #REQUIRED>  
>
```

ScanPauseResponse

```
<!DOCTYPE ScanPauseResponse [  
<!ELEMENT ScanPauseResponse (Failure?)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST ScanPauseResponse success (0|1) #REQUIRED>  
>
```

ScanResume

Resume a running scan.

ScanResumeRequest

```
<!DOCTYPE ScanResumeRequest [  
<!ELEMENT ScanResumeRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST ScanResumeRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST ScanResumeRequest session-id CDATA #REQUIRED>  
  <!-- the ID of the scan -->  
  <!ATTLIST ScanResumeRequest scan-id CDATA #REQUIRED>  
>
```

ScanResumeResponse

```
<!DOCTYPE ScanResumeResponse [  
<!ELEMENT ScanResumeResponse (Failure?)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST ScanResumeResponse success (0|1) #REQUIRED>  
>
```

ScanStop

Stop a running scan.

ScanStopRequest

```
<!DOCTYPE ScanStopRequest [  
<!ELEMENT ScanStopRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST ScanStopRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST ScanStopRequest session-id CDATA #REQUIRED>  
  <!-- the ID of the scan -->  
  <!ATTLIST ScanStopRequest scan-id CDATA #REQUIRED>  
>
```

ScanStopResponse

```
<!DOCTYPE ScanStopResponse [  
<!ELEMENT ScanStopResponse (Failure?)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST ScanStopResponse success (0|1) #REQUIRED>  
>
```

ScanStatus

Check the current status of a scan.

ScanStatusRequest

```
<!DOCTYPE ScanStatusRequest [  
<!ELEMENT ScanStatusRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST ScanStatusRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST ScanStatusRequest session-id CDATA #REQUIRED>  
  <!-- the ID of the scan -->  
  <!ATTLIST ScanStatusRequest scan-id CDATA #REQUIRED>  
>
```

ScanStatusResponse

```
<!DOCTYPE ScanStatusResponse [  
<!ELEMENT ScanStatusResponse (Failure?)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST ScanStatusResponse success (0|1) #REQUIRED>  
  <!-- the ID of the scan -->  
  <!ATTLIST ScanStatusResponse scan-id CDATA #REQUIRED>  
  <!-- the ID of the scan engine -->  
  <!ATTLIST ScanStatusResponse engine-id CDATA #REQUIRED>  
  <!-- the current scan status -->  
  <!ATTLIST ScanStatusResponse status (running|finished|stopped|error|  
                                         dispatched|paused|aborted|unknown) #REQUIRED>  
>
```

ScanStatistics

Get scan statistics, including node and vulnerability breakdowns.

ScanStatisticsRequest

```
<!DOCTYPE ScanStatisticsRequest [  
<!ELEMENT ScanStatisticsRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST ScanStatisticsRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST ScanStatisticsRequest session-id CDATA #REQUIRED>  
  <!-- the ID of the scan -->  
  <!ATTLIST ScanStatisticsRequest scan-id CDATA #REQUIRED>  
>
```

ScanStatisticsResponse

```
<!DOCTYPE ScanStatisticsResponse [  
<!ELEMENT ScanStatisticsResponse (Failure|ScanSummary)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST ScanStatisticsResponse success (0|1) #REQUIRED>  
  <!-- see the ScanSummary DTD for more details -->  
>
```

Vulnerability Assessment

VulnerabilityListing

Provide a list of vulnerabilities checked by Nexpose.

NOTE: Additional features for this API are now available in API Extended v1.2. Refer to the *Nexpose Extended API v1.2 Guide* for more information.

VulnerabilityListingRequest

```
<!DOCTYPE VulnerabilityListingRequest [
<!ELEMENT VulnerabilityListingRequest EMPTY>
  <!-- user defined synchronization token id used to avoid duplicate requests -->
  <!ATTLIST VulnerabilityListingRequest sync-id CDATA #IMPLIED>
  <!-- the current session id -->
  <!ATTLIST VulnerabilityListingRequest session-id CDATA #REQUIRED>
]>
```

VulnerabilityListingResponse

```
<!DOCTYPE VulnerabilityListingResponse [  
<!ELEMENT VulnerabilityListingResponse (Failure|VulnerabilitySummary*)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST VulnerabilityListingResponse success (0|1) #REQUIRED>  
<!ELEMENT VulnerabilitySummary EMPTY>  
  <!ATTLIST VulnerabilitySummary id CDATA #REQUIRED>  
  <!ATTLIST VulnerabilitySummary title CDATA #REQUIRED>  
  <!ATTLIST VulnerabilitySummary severity CDATA #REQUIRED>  
  <!ATTLIST VulnerabilitySummary pciSeverity CDATA #REQUIRED>  
  <!ATTLIST VulnerabilitySummary cvssScore CDATA #IMPLIED >  
  <!ATTLIST VulnerabilitySummary cvssVector CDATA #IMPLIED >  
  <!-- the published date and time is in ISO 8601 format,  
    YYYYMMDDTHHMMSSsss, such as: 19981231T00000000 -->  
  <!ATTLIST VulnerabilitySummary published CDATA #IMPLIED >  
  <!-- the added date and time is in ISO 8601 format,  
    YYYYMMDDTHHMMSSsss, such as: 19981231T00000000 -->  
  <!ATTLIST VulnerabilitySummary added CDATA #REQUIRED>  
  <!-- the modified date and time is in ISO 8601 format,  
    YYYYMMDDTHHMMSSsss, such as: 19981231T00000000 -->  
  <!ATTLIST VulnerabilitySummary modified CDATA #REQUIRED>  
  ]>
```

VulnerabilityDetails

Provide the full details of a vulnerability, including its description, cross-references, and solution.

NOTE: Additional features for this API are now available in API Extended v1.2. Refer to the Nexpose *Extended API v1.2 Guide* for more information.

VulnerabilityDetailsRequest

```
<!DOCTYPE VulnerabilityDetailsRequest [
<!ELEMENT VulnerabilityDetailsRequest EMPTY>
  <!-- user defined synchronization token id used to avoid duplicate requests -->
  <!ATTLIST VulnerabilityDetailsRequest sync-id CDATA #IMPLIED>
  <!-- the current session id -->
  <!ATTLIST VulnerabilityDetailRequest session-id CDATA #REQUIRED>
  <!-- the id of the vulnerability to retrieve -->
  <!ATTLIST VulnerabilityDetailRequest vuln-id CDATA #REQUIRED>
]>
```

VulnerabilityDetailsResponse

```
<!DOCTYPE VulnerabilityDetailsResponse [  
<!ELEMENT VulnerabilityDetailsResponse (Failure|Vulnerability)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST VulnerabilityDetailsResponse success (0|1) #REQUIRED>  
<!ELEMENT Vulnerability (description, references, solution)>  
  <!ATTLIST Vulnerability id CDATA #REQUIRED>  
  <!ATTLIST Vulnerability title CDATA #REQUIRED>  
  <!ATTLIST Vulnerability severity CDATA #REQUIRED>  
  <!ATTLIST Vulnerability pciSeverity CDATA #REQUIRED>  
  <!ATTLIST Vulnerability cvssScore CDATA #IMPLIED >  
  <!ATTLIST Vulnerability cvssVector CDATA #IMPLIED >  
  <!-- the published date and time is in ISO 8601 format,  
        YYYYMMDDTHHMMSSsss, such as: 19981231T00000000 -->  
  <!ATTLIST Vulnerability published CDATA #IMPLIED >  
  <!-- the added date and time is in ISO 8601 format,  
        YYYYMMDDTHHMMSSsss, such as: 19981231T00000000 -->  
  <!ATTLIST Vulnerability added CDATA #REQUIRED>  
  <!-- the modified date and time is in ISO 8601 format,  
        YYYYMMDDTHHMMSSsss, such as: 19981231T00000000 -->  
  <!ATTLIST Vulnerability modified CDATA #REQUIRED>  
  
<!ELEMENT description (#PCDATA)>  
  
<!ELEMENT references (reference*)>  
<!ELEMENT reference (#PCDATA)>  
  <!-- the source of the reference, such as cve, bid, mskb, etc -->  
  <!ATTLIST reference source CDATA #REQUIRED>  
  
<!ELEMENT solution (#PCDATA)>  
  
>
```

Reporting

ReportTemplateListing

Provide a list of all report templates the user can access on the security console.

ReportTemplateListingRequest

```
<!DOCTYPE ReportTemplateListingRequest [  
  <!ELEMENT ReportTemplateListingRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST ReportTemplateListingRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST ReportTemplateListingRequest session-id CDATA #REQUIRED>  
>
```

ReportTemplateListingResponse

```
<!DOCTYPE ReportTemplateListingResponse [  
  <!ELEMENT ReportTemplateListingResponse (Failure|ReportTemplateSummary*)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST ReportTemplateListingResponse success (0|1) #REQUIRED>  
  
  <!-- See the ReportTemplateSummary DTD for more details -->  
>
```

ReportTemplateConfig

Retrieve the configuration for a report template.

ReportTemplateConfigRequest

```
<!DOCTYPE ReportTemplateConfigRequest [
<!ELEMENT ReportTemplateConfigRequest EMPTY>
  <!-- user defined synchronization token id used to avoid duplicate requests -->
  <!ATTLIST ReportTemplateConfigRequest sync-id CDATA #IMPLIED>
  <!-- the current session id -->
  <!ATTLIST ReportTemplateConfigRequest session-id CDATA #REQUIRED>
  <!-- the ID of the report template to retrieve the config for -->
  <!ATTLIST ReportTemplateConfigRequest template-id CDATA #REQUIRED>
]>
```

ReportTemplateConfigResponse

```
<!DOCTYPE ReportTemplateConfigResponse [
<!ELEMENT ReportTemplateConfigResponse (Failure|ReportTemplate)>
  <!-- set to 1 upon success, 0 otherwise -->
  <!ATTLIST ReportTemplateConfigResponse success (0|1) #REQUIRED>

  <!-- See the ReportTemplate DTD for more details -->

]>
```

ReportTemplateSave

Save the configuration for a report template.

ReportTemplateSaveRequest

If the user attempts to save a report with a scope of “silo” without being logged into a silo, an error occurs.

```
<!DOCTYPE ReportTemplateSaveRequest [
<!ELEMENT ReportTemplateSaveRequest (ReportTemplate)>
  <!-- user defined synchronization token id used to avoid duplicate requests -->
  <!ATTLIST ReportTemplateSaveRequest sync-id CDATA #IMPLIED>
  <!-- the current session id -->
  <!ATTLIST ReportTemplateSaveRequest session-id CDATA #REQUIRED>
  <!-- the visibility (scope) of the report template -->
  <!ATTLIST ReportTemplateSaveRequest scope (global|silo) #IMPLIED>

  <!-- See the ReportTemplate DTD for more details -->

]>
```

ReportTemplateSaveResponse

```
<!DOCTYPE ReportTemplateSaveResponse [
<!ELEMENT ReportTemplateSaveResponse (Failure?)>
  <!-- set to 1 upon success, 0 otherwise -->
  <!ATTLIST ReportTemplateSaveResponse success (0|1) #REQUIRED>
  <!-- the newly assigned report template ID
  (unchanged for existing report templates) -->
  <!ATTLIST ReportTemplateSaveResponse template-id CDATA #REQUIRED>

]>
```

ReportListing

Provide a listing of all report definitions the user can access on the security console.

ReportListingRequest

```
<!DOCTYPE ReportListingRequest [  
<!ELEMENT ReportListingRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST ReportListingRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST ReportListingRequest session-id CDATA #REQUIRED>  
>
```

ReportListingResponse

```
<!DOCTYPE ReportListingResponse [  
<!ELEMENT ReportListingResponse (Failure|ReportConfigSummary*)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST ReportListingResponse success (0|1) #REQUIRED>  
  
  <!-- See the ReportConfigSummary DTD for more details -->  
>
```

ReportHistory

Provide a history of all reports generated with the specified report definition.

ReportHistoryRequest

```
<!DOCTYPE ReportHistoryRequest [  
<!ELEMENT ReportHistoryRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST ReportListingRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST ReportHistoryRequest session-id CDATA #REQUIRED>  
  <!-- the report definition id -->  
  <!ATTLIST ReportHistoryRequest reportcfg-id CDATA #REQUIRED>  
>
```

ReportHistoryResponse

```
<!DOCTYPE ReportHistoryResponse [  
<!ELEMENT ReportHistoryResponse (Failure|ReportSummary*)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST ReportListingResponse success (0|1) #REQUIRED>  
  
<!-- See the ReportSummary DTD for more details -->  
  

```

ReportConfig

Retrieve the configuration for a report definition.

ReportConfigRequest

```
<!DOCTYPE ReportConfigRequest [  
<!ELEMENT ReportConfigRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST ReportConfigRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  

```

ReportConfigResponse

```
<!DOCTYPE ReportConfigResponse [  
<!ELEMENT ReportConfigResponse (Failure|ReportConfig)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST ReportConfigResponse success (0|1) #REQUIRED>  
  

```

ReportSave

Save the configuration for a report definition.

ReportSaveRequest

```
<!DOCTYPE ReportSaveRequest [  
<!ELEMENT ReportSaveRequest (ReportConfig)>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST ReportSaveRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST ReportSaveRequest session-id CDATA #REQUIRED>  
  <!-- Should the report be generated now? This is checked only if the report is NOT  
  scheduled or scan based. -->  
  <!ATTLIST ReportSaveRequest generate-now (0|1) "1">  
  
<!-- See the ReportConfig DTD for more details -->  
  

```

ReportSaveResponse

```
<!DOCTYPE ReportSaveResponse [  
<!ELEMENT ReportSaveResponse (Failure?)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST ReportSaveResponse success (0|1) #REQUIRED>  
  <!-- the newly assigned report config ID (unchanged for existing reports) -->  

```

ReportGenerate

Generate a new report using the specified report definition.

ReportGenerateRequest

```
<!DOCTYPE ReportGenerateRequest [  
  <!ELEMENT ReportGenerateRequest EMPTY>  
    <!-- user defined synchronization token id used to avoid duplicate requests -->  
    <!ATTLIST ReportGenerateRequest sync-id CDATA #IMPLIED>  
    <!-- the current session id -->  
    <!ATTLIST ReportGenerateRequest session-id CDATA #REQUIRED>  
    <!ATTLIST ReportGenerateRequest report-id CDATA #REQUIRED>  
  ]>
```

ReportGenerateResponse

```
<!DOCTYPE ReportGenerateResponse [  
  <!ELEMENT ReportGenerateResponse (Failure|ReportSummary)>  
    <!-- set to 1 upon success, 0 otherwise -->  
    <!ATTLIST ReportGenerateResponse success (0|1) #REQUIRED>  
  
  <!-- See the ReportSummary DTD for more details -->  
  ]>
```

ReportDelete

Delete a previously generated report or report definition.

ReportDeleteRequest

```
<!DOCTYPE ReportDeleteRequest [  
<!ELEMENT ReportDeleteRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST ReportDeleteRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST ReportDeleteRequest session-id CDATA #REQUIRED>  
  <!-- the report definition id to remove the definition and  
    all reports generated with the definition -->  
  <!ATTLIST ReportDeleteRequest reportcfg-id CDATA #IMPLIED>  
  <!-- the id of the generated report to remove -->  
  <!ATTLIST ReportDeleteRequest report-id CDATA #IMPLIED>  
>
```

ReportDeleteResponse

```
<!DOCTYPE ReportDeleteResponse [  
<!ELEMENT ReportDeleteResponse (Failure?)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST ReportDeleteResponse success (0|1) #REQUIRED>  
>
```

ReportAdhocGenerate

Generate a report once using a simple configuration, and send it back in a multipart mime response.

ReportAdhocGenerateRequest

```
<!DOCTYPE ReportAdhocGenerateRequest [
<!ELEMENT ReportAdhocGenerateRequest (AdhocReportConfig)>
  <!-- user defined synchronization token id used to avoid duplicate requests -->
  <!ATTLIST ReportAdhocGenerateRequest sync-id CDATA #IMPLIED>
  <!-- the current session id -->
  <!ATTLIST ReportAdhocGenerateRequest session-id CDATA #REQUIRED>

  <!-- With the site, device, and group filters you determine which assets to include
  in the report. With the vuln-severity and vuln-status filters you control which vul-
  nerabilities to include in the report. -->
  <!ELEMENT AdhocReportConfig (Filters, Baseline?) >
    <!-- the id of the report template used -->
    <!ATTLIST AdhocReportConfig template-id CDATA #REQUIRED>
    <!ATTLIST AdhocReportConfig format (pdf|html|rtf|xml|text|csv|raw-xml|raw-xml-v2|
ns-xml|qualys-xml) #REQUIRED>

    <!-- The configuration must include at least one of device (asset), site, group
    (asset group) or scan filter to define the scope of report. The vuln-status filter
    can be used only with raw report formats: csv or raw_xml. If the vuln-status filter
    is not included in the configuration, all the vulnerability test results (including
    invulnerable instances) are exported by default in csv and raw_xml reports. -->
    <!ELEMENT Filters (filter+)>
    <!ELEMENT filter EMPTY>
      <!ATTLIST filter type (site|group|device|scan|vuln-severity|vuln-status)
#REQUIRED>
```

continued on the following page

ReportAdhocGenerateRequest (continued)

```
<!-- the ID of a specific site, group, device or scan.
For scan, this can also be "last" for the most recently run scan
For vuln-status, the ID can have one of the following values:
1)    vulnerable-exploited (The check was positive. An exploit verified the vulner-
ability.)
2)    vulnerable-version (The check was positive. The version of the scanned ser-
vice or software is associated with known vulnerabilities.)
3)    potential (The check for a potential vulnerability was positive.)
These values are supported for CSV and XML formats.
-->
    <!ATTLIST filter id CDATA #REQUIRED>

<!ELEMENT Baseline EMPTY>
    <!-- the date to use as the baseline scan in ISO 8601 format,
         YYYYMMDDTHHMMSSss, such as: 19981231T00000000. Additionally,
         "first" can be used for the first run scan, or "previous"
         for the most recently run scan prior to the current scan. -->
    <!ATTLIST Baseline compareTo CDATA #REQUIRED>
]>
```

ReportAdhocGenerateResponse

Response to ReportAdhocGenerateRequest is a Multipart Mime message where the first part is 'response_xml' which contains the following xml element:

```
<!DOCTYPE ReportAdhocGenerateResponse [  
  <!ELEMENT ReportAdhocGenerateResponse (Failure?)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST ReportAdhocGenerateResponse success (0|1) #REQUIRED>  
>
```

The rest of the parts of the multipart mime contain the actual report files depending upon how many files are there. All these files are encoded using the base64 format.

User management functions

Only Nexpose global administrators can use these functions.

UserListing

Provide a list of user accounts and information about those accounts.

UserListingRequest

```
<!DOCTYPE UserListingRequest [  
  <!ELEMENT UserListingRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST UserListingRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST UserListingRequest session-id CDATA #REQUIRED>  
>
```

UserListingResponse

```
<!DOCTYPE UserListingResponse [  
  <!ELEMENT UserListingResponse (Failure|UserSummary*)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST UserListingResponse success (0|1) #REQUIRED>  
>
```

UserAuthenticatorListing

Provide a list of user authentication sources.

UserAuthenticatorListingRequest

```
<!DOCTYPE UserAuthenticatorListingRequest [  
  <!ELEMENT UserAuthenticatorListingRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST UserAuthenticatorListingRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST UserAuthenticatorListingRequest session-id CDATA #REQUIRED>  
>
```

UserAuthenticatorListingResponse

```
<!DOCTYPE UserAuthenticatorListingResponse [  
  <!ELEMENT UserAuthenticatorListingResponse (Failure|AuthenticatorSummary*)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST UserAuthenticatorListingResponse success (0|1) #REQUIRED>  
>
```

UserConfig

List information about a given user account.

UserConfigRequest

```
<!DOCTYPE UserConfigRequest [  
  <!ELEMENT UserConfigRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST UserConfigRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST UserConfigRequest session-id CDATA #REQUIRED>  
  <!-- the id of the user to retrieve the config for -->  
  <!ATTLIST UserConfigRequest id CDATA #REQUIRED>  
>
```

UserConfigResponse

```
<!DOCTYPE UserConfigResponse [  
<!ELEMENT UserConfigResponse (Failure|UserConfig)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST UserConfigResponse success (0|1) #REQUIRED>  
  
  <!-- See the UserConfig DTD for more details. Note: a user's  
    password will never be included in the response -->  
]>
```

UserSave

Create a new user account, or update the settings for an existing account. Note that specifying a UserConfig with an id of -1 indicates a *create* request.

It is not possible to create user accounts with custom roles, but it is possible to query these accounts with UserListing or UserConfig.

You cannot change the user name after you create an account. Therefore, the user name that you specify in the update request must be the current user name.

```
<!DOCTYPE UserSaveRequest [  
<!ELEMENT UserSaveRequest UserConfig>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST UserSaveRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST UserSaveRequest session-id CDATA #REQUIRED>  
  
  <!-- See the UserConfig DTD for more details -->  
]>
```

UserSaveResponse

```
<!DOCTYPE UserSaveResponse [  
<!ELEMENT UserSaveResponse (Failure?)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST UserSaveResponse success (0|1) #REQUIRED>  
  <!-- the id of the user created or updated -->  
  <!ATTLIST UserSaveResponse user-id CDATA #REQUIRED>  
]>
```

UserDelete

Delete a user account.

Note that you cannot delete a user account that is associated with reports or tickets.

UserDeleteRequest

```
<!DOCTYPE UserDeleteRequest [  
  <!ELEMENT UserDeleteRequest EMPTY>  
  <!-- user defined synchronization token id used to avoid duplicate requests -->  
  <!ATTLIST UserDeleteRequest sync-id CDATA #IMPLIED>  
  <!-- the current session id -->  
  <!ATTLIST UserDeleteRequest session-id CDATA #REQUIRED>  
  <!-- the ID of the user to delete -->  
  <!ATTLIST UserDeleteRequest id CDATA #REQUIRED>  
>
```

UserDeleteResponse

You cannot delete your own user account.

```
<!DOCTYPE UserDeleteResponse [  
  <!ELEMENT UserDeleteResponse (Failure?)>  
  <!-- set to 1 upon success, 0 otherwise -->  
  <!ATTLIST UserDeleteResponse success (0|1) #REQUIRED>  
>
```

General management and diagnostic functions

ConsoleCommand

Execute an arbitrary Nexpose console command that is supplied as text via an API parameter. The Nexpose console commands are documented in the Nexpose *Administrator's Guide*. If you use a command that is not listed in the in Nexpose *Administrator's Guide*, Nexpose will return the XMLResponse.

ConsoleCommandRequest

```
<!DOCTYPE ConsoleCommandRequest [
<!ELEMENT ConsoleCommandRequest (Command)>
  <!-- user defined synchronization token id used to avoid duplicate
  requests -->
  <!ATTLIST ConsoleCommandRequest sync-id CDATA #IMPLIED>
  <!-- the current session id -->
  <!ATTLIST ConsoleCommandRequest session-id CDATA #REQUIRED>
<!ELEMENT Command CDATA #REQUIRED>
]>
```

ConsoleCommandResponse

Warning: Set a higher timeout value for a command that requires a substantial amount of time to execute and finish. Doing so ensures that Nexpose has sufficient time to respond to the command.

```
<!DOCTYPE ConsoleCommandResponse [
<!ELEMENT ConsoleCommandResponse (Command,Output)>
  <!-- set to 1 upon success, 0 otherwise -->
  <!ATTLIST ConsoleCommandResponse success (0|1) #REQUIRED>
<!ELEMENT Command CDATA #REQUIRED>
<!ELEMENT Output CDATA #REQUIRED>
]>
```

SystemInformation

Obtain Nexpose system data, such as total RAM, free RAM, total disk space, free disk space, CPU speed, number of CPU cores, and other vital information.

SystemInformationRequest

```
<!DOCTYPE SystemInformationRequest [
<!ELEMENT SystemInformationRequest EMPTY>
  <!-- user defined synchronization token id used to avoid duplicate
  requests -->
  <!ATTLIST SystemInformationRequest sync-id CDATA #IMPLIED>
  <!-- the current session id -->
  <!ATTLIST SystemInformationRequest session-id CDATA #REQUIRED>
]>
```

SystemInformationResponse

```
<!DOCTYPE SystemInformationResponse [
<!ELEMENT SystemInformationResponse (SystemInformationSummary)>
  <!ATTLIST SystemInformationResponse success (0|1) #REQUIRED>
  <!-- set to 1 upon success, 0 otherwise -->
  <!ELEMENT SystemInformationSummary (Statistic*)>
    <!ELEMENT Statistic CDATA #IMPLIED>
      <!ATTLIST Statistic name (cpu-count|cpu-speed|disk-install|java-name|
      jre-version|last-update-date|last-update-id|disk-tmp|nsc-name|nsc-version|
      nse-version|os|ram-free|ram-total|up-time|db-product|db-version|java-heap-
max|
      java-heap-committed|java-heap-free|java-heap-used|java-total-thread-count|
      java-started-thread-count|java-thread-peak-count|java-daemon-thread-count)
      #IMPLIED>
    ]>
```

StartUpdate

Induce Nexpose to retrieve required updates and restart if necessary.

StartUpdateRequest

```
<!DOCTYPE StartUpdateRequest [
<!ELEMENT StartUpdateRequest EMPTY>
  <!-- user defined synchronization token id used to avoid duplicate
  requests -->
  <!ATTLIST StartUpdateRequest sync-id CDATA #IMPLIED>
  <!-- the current session id -->
  <!ATTLIST StartUpdateRequest session-id CDATA #REQUIRED>
]>
```

StartUpdateResponse

Warning: Set a higher timeout value for a command that requires a substantial amount of time to execute and finish. Doing so ensures that Nexpose has sufficient time to respond to the command.

```
<!DOCTYPE StartUpdateResponse [
<!ELEMENT StartUpdateResponse (Failure?)>
  <!-- set to 1 upon success, 0 otherwise -->
  <!ATTLIST StartUpdateResponse success (0|1) #REQUIRED>
]>
```

Restart

Restart Nexpose.

RestartRequest

```
<!DOCTYPE RestartRequest [
<!ELEMENT RestartRequest EMPTY>
  <!-- user defined synchronization token id used to avoid duplicate
  requests -->
  <!ATTLIST RestartRequest sync-id CDATA #IMPLIED>
  <!-- the current session id -->
  <!ATTLIST RestartRequest session-id CDATA #REQUIRED>
]>
```

RestartResponse

There is no response to RestartRequest. When Nexpose shuts down as part of the restart process, it terminates any active connections. Therefore, Nexpose cannot issue a response when it restarts.

SendLog

Output diagnostic information into log files, zip the files, and encrypt the archive with a PGP public key that is provided as a parameter for the API call. Then, either e-mail this archive to an address that is specified as an API parameter, or upload the archive using HTTP or HTTPS to a URL that is specified as an API parameter.

If you do not specify a key, the SendLogRequest uses a Rapid7 default key.

SendLogRequest

```
<!DOCTYPE SendLogRequest [
<!ELEMENT SendLogRequest (Transport)>
  <!-- user defined synchronization token id used to avoid duplicate
  requests -->
  <!ATTLIST SendLogRequest sync-id CDATA #IMPLIED>
  <!-- the current session id -->
  <!ATTLIST SendLogRequest session-id CDATA #REQUIRED>
  <!ATTLIST SendLogRequest keyid CDATA #IMPLIED>
<!ELEMENT Transport (Email|URL)>
  <!ATTLIST Transport protocol CDATA #REQUIRED (smtp|http|https)>
  <!-- If protocol== "smtp" -->
  <!-- See the Email DTD for more details -->
  <!-- If protocol == "http" || "https" -->
  <!ELEMENT URL CDATA #REQUIRED>
]>
```

SendLogResponse

```
<!DOCTYPE SendLogResponse [
<!ELEMENT SendLogResponse (Failure?)>
  <!-- set to 1 upon success, 0 otherwise -->
  <!ATTLIST SendLogResponse success (0|1) #REQUIRED>
]>
```

Error responses

Examining error messages that the Nexpose API generates can be helpful in understanding why requests fail.

Error messages include stack traces, which can be lengthy. For the examples in this chapter, large portions of stack traces will be represented by ellipses (...).

Example:

```
<stacktrace>org.xml.sax.SAXParseException: XML document structures must start and
end within the same entity.
...
Error parsing XML at line 1, column 54
</stacktrace>
```

This chapter includes descriptions of general types of error responses.

Examples will include valid requests, intentionally invalid requests, and responses for these requests.

Error responses for malformed XML

API requests that include invalid XML structures will generate one type of error message. Examples of malformed XML include misplaced or omitted characters such as closing tags or quotation marks.

Malformed XML error responses will include the `<Failure>` or `<XML response>` tags. See the DTDs for these tags for more information in *Appendix A: DTD Listings* (on page 75).

This is a malformed XML request:

```
<LoginRequest user-id="a" password=".....">
```

The request is missing a closing `</LoginRequest>` tag.

This is the error response for the preceding request:

```
<LoginResponse success="0">
<Failure>
<Exception>
<message>XML document structures must start and end within the same entity.</mes-
sage>
<stacktrace>org.xml.sax.SAXParseException: XML document structures must start and
end within the same entity.
...
Error parsing XML at line 1, column 54
</stacktrace>
</Exception>
</Failure>
</LoginResponse>
```

Error responses for requests for non-existent API functions

Requests for non-existent API functions will generate one type of error message.

These requests often include misspelled functions, such as in the following example:

```
<LorginRequest user-id="a" password="....."/>
```

"Login" is misspelled as "Lorgin".

This is the error response for the preceding request:

```
<XMLResponse success="0">
  <Failure>
    <Exception>
      <message>Failed initializing handler for LorginRequest</message>
      <stacktrace>org.xml.sax.SAXException: Failed initializing handler for LorginRequest
        ...
      </stacktrace>
    </Exception>
  </Failure>
</XMLResponse>
```

Error responses common to all valid requests

Any valid API request will generate an error response if you send it while Nexpose is still starting.

Any valid API request *except for* LoginRequest will generate an error response under the following circumstances:

- Your session is invalid because it expired over time, it was manually closed, or the session ID is invalid.
- You do not enter a `session-id` attribute value.
- You do not include the `session-id` attribute.

Following is an example of an error response. The string [api] represents the API call that was made with a bad session ID.

```
<[api]Response success="0">
  <Failure>
    <Exception>
      <message>Session not found</message>
      <stacktrace>com.rapid7.net.http.HTTPException: Session not found
        ...
      </stacktrace>
    </Exception>
  </Failure>
</[api]Response>
```

Code samples

This section contains sample code for a simple implementation of a Nexpose API client. It is not a complete implementation, but the code samples demonstrate how an API client interacts with Nexpose.

The sample code is written in Ruby, but has been written such that you will not need Ruby expertise to understand how the code works. The code is not meant to illustrate the “best” way of implementing an API client. It is a generic implementation that can be adapted to suit your organization's language choice and coding standards.

A more complete API implementation is available at <http://www.metasploit.com/redmine/projects/framework/repository/entry/lib/rapid7/nexpose.rb>

Fundamental API sequence

The fundamental sequence for interacting with the Nexpose API is the following:

1. Open an HTTPS connection to the Web console, usually on port 3780.
2. Verify that the Content-type HTTP header is set to “text/xml”.
3. Construct a LoginRequest XML request containing valid Nexpose credentials.
4. Send the XML request via the HTTPS connection to `https://ncs:3780/api/1.1/xml` using HTTP POST Method, where “ncs” is the hostname of the Nexpose Security Console.
5. Parse the returned LoginResponse.
6. If the success attribute is set to 1, extract the session-id attribute for use in subsequent requests. If the success attribute is set to 0, extract the Failure information and report it.
7. Construct an XML request containing the session ID.
8. Send the XML request via the HTTPS connection to `https://ncs:3780/api/1.1/xml` if the API command is a version 1.1 command, or to `https://ncs:3780/api/1.2/xml` if the API command is a version 1.2 command, using the HTTP POST Method, where “ncs” is the hostname of the Nexpose Security Console.
9. Parse the returned XML response.
10. If the success attribute is set to 1, extract the requisite information for the XML response. If the success attribute is set to 0, extract the Failure information and report it.
11. Repeat steps 7-10 for the API calls you wish to make. When you have finished, go to Step 13.
12. Construct a LogoutRequest XML request containing the session ID.
13. Send the XML request via the HTTPS connection to `https://ncs:3780/api/1.1/xml` using the HTTP POST Method, where “ncs” is the hostname of the Nexpose Security Console. If the success attribute is set to 1, the session has ended. If the success attribute is set to 0, extract the Failure information and report it.

Preliminaries

The sample API client implementation is structured as a single class, called `APIClient`, that makes the HTTPS connection to the Nexpose Security Console and sends XML requests via HTTP POST. The API commands are methods of the `APIClient` class. In this example, we concentrate on the HTTPS connection initialization. The API command methods are stubs; their content will be documented and explained in subsequent sections..

```
# The three 'require' lines load the libraries that the APIClient
# needs to make an HTTPS connection with the
NexposeSecurity Console,
# and also the standard Ruby XML parser. The libraries that you use
# could be different depending on your environment and requirements.
require 'net/https'
require 'net/http'
require 'rexml/document'

class APIClient
  # The initialize method creates the APIClient object and the HTTPS
  # connection to the specified host and port. The '@' symbol
  # in front of the variable names makes the variable visible to all the
  # methods in the class.
  #
  # Since Nexpose uses a self-signed certificate,
  this client uses
  # @client.verify_mode = OpenSSL::SSL::VERIFY_NONE which configures the
  # SSL connection to forego checking that the host name of the server
  # matches the SSL certificate, even though the encryption itself is
  # functional. This leaves this particular implementation of the
  # APIClient vulnerable to a potential Man-in-the-Middle attack.
  # However, configuring SSL host verification is beyond the scope
  # of this document.
  def initialize(host, port = 3780)
    @client = Net::HTTP.new(host, port)
    @client.use_ssl = true
    @client.verify_mode = OpenSSL::SSL::VERIFY_NONE
    # The URIs for the API. We only use 1.1 APIs in this
    # implementation, but changing the URI is straightforward.
    @uri11 = "/api/1.1/xml"
    @uri12 = "/api/1.2/xml"
    # The HTTP message header must have the content type
    # configured to "text/xml"
    @ext_header = {"Content-type" => "text/xml"}
  end
end
```

```
# This helper method takes messages created by the API client,
# POSTs the messages to the API URI, and parses the response with
# the REXML XML parser. The parsed response is assigned to the
# @response variable, which is visible to all the methods in the

# class.
  def post(body)
    @response = REXML::Document.new(@client.post
(@uri11, body, @ext_header).body).rootend
#The methods below implement the API commands.
  def login
    ...
  end
    def logout
    ...
  end

  def usercreate
    ...
  end
    def sitecreate
    ...
  end
    def sitelisting
    ...
  end
    def scansite
    ...
  end

  def vulndetail
    ...
  end

  def report
    ...
  end
end
#Creates the API client
client = APIClient.new("hostname.com", 3780)
```

Login implementation

The Login command is essential to the operation of a Nexpose API client. The client must send a LoginRequest, along with valid credentials, to the Nexpose API in order to receive a valid session id. The session id must be included with every subsequent interaction with the Nexpose API.

```
def login(username, password)
  # Create the LoginRequest XML message with the provided username
  # and password.
  body = "<LoginRequest user-id=\"#{username}\" password=\"#{password}\"></LoginRe-
  quest>"
  # Sends a POST request containing the XML message created in the
  # previous line, and creates a response.
  post(body)
  # Nexpose returns an XML response.
  If the response has a success
  # attribute of 1, then the session id is extracted and assigned
  # to the @sessionid variable. Otherwise, the login has failed,
  # and the reason is output as a Failure XML message.
  if @response.attributes["success"].to_i == 1
    @sessionid = @response.attributes["session-id"]
    puts "Login successful: #{@sessionid}"
  else
    puts @response
  end
end
```

User creation implementation

This method builds and posts a `UserSaveRequest` to create a new user. This implementation specifically creates users and activates them, so some of the `UserSaveRequest` attributes are given defaults.

```
def usercreate(login, password, name, email, role="user")
  # An id of -1 creates a user.
  id = "-1"
  authsrc = "2"
  enabled = "1"
  allgroups = "true"
  allsites = "true"
  # Build the UserSaveRequest XML message with the session ID
  # and attributes.
  body = "<UserSaveRequest session-id=\"#{@sessionid}\">"
  body << "<UserConfig id=\"#{id}\" authsrcid=\"#{authsrc}\" name=\"#{login}\" "
  body << "password=\"#{password}\" fullname=\"#{name}\" email=\"#{email}\" "
  body << "role-name=\"#{role}\" enabled=\"#{enabled}\" allGroups=\"#{allgroups}\" "
  allSites=\"#{allsites}\">"
  body << "</UserConfig>"
  body << "</UserSaveRequest>"
  # Send the request and receive the response.
  post(body)
  # Process response and return message depending on
  # success or failure.
  if @response.attributes["success"].to_i == 1
    puts "Creation of user #{login} successful."
  else
    puts @response
  end
end

# Use the method with the API client. Create a user named
# John Smith with the login "newguy", the password "secret",
# the e-mail address jsmith@company.com, and assign the "user"
# role.
client.usercreate("newguy", "secret", "John Smith", "jsmith@company.com",
  role="user")
```

Site creation implementation

This method builds and posts a SiteSaveRequest to create a new site. This implementation specifically creates sites, so some of the SiteSaveRequest attributes are given defaults.

```
def sitecreate(host, name, description='', template="full-audit")
  # An id of -1 creates a new site.
  id = "-1"
  # Build the SiteSaveRequest XML message with the session ID
  # and attributes.
  body = "<SiteSaveRequest session-id=\"#{@sessionid}\">"
  body << "<Site id=\"#{id}\" name=\"#{name}\" description=\"#{description}\">"
  body << "<Hosts><host>#{host}</host></Hosts>"
  body << "<Credentials></Credentials>"
  body << "<Alerting></Alerting>"
  body << "<ScanConfig configID=\"#{id}\" name=\"Special Example\" templateID=\"#{template}\"></ScanConfig>"
  body << "</Site>"
  body << "</SiteSaveRequest>"
  # Send the request and receive the response.
  post(body)
  # Process response and return message depending on
  # success or failure.
  if @response.attributes["success"].to_i == 1
    puts "Creation of site #{name} successful."
  else
    puts @response
  end
end

# Use the method with the API client -- create a site with
# IP address 10.0.0.1 called "Primary Site", and assign the "full-
# audit" scan template to the site.
client.sitecreate("10.0.0.1", "Primary Site", "The primary site.", "full-audit")
```

Site listing implementation

This method builds and posts a `ScanListingRequest`. It then extracts a subset of the available information from the `ScanListingResponse` and produces formatted output.

```
def sitelisting
  # Build the SiteListingRequest with the session ID. Note
  # that the SiteListingRequest has no attributes or elements.
  body = "<SiteListingRequest session-id=\"#{@sessionid}\"></SiteListingRequest>"
  # Send the request and receive the response.
  post(body)
  # Process response and return message depending on
  # success or failure. If successful, extract data
  # from the response.
  if @response.attributes["success"].to_i == 1
    # Loop through each of the SiteSummary elements in the
    # in the response.
    @response.elements.each('SiteSummary') do |s|
      puts "Site ID: #{s.attributes['id']}"
      puts "Name: #{s.attributes['name']}"
      puts "Description: #{s.attributes['description']}"
      # This is a score calculated from two attributes.
      puts "Risk Factor + Risk Score: #{s.attributes['riskfactor'].to_i + s.attributes['riskscore'].to_i}"
      puts
    end
  else
    puts @response
  end
end

# Use the method with the API client
client.sitelisting
```

Site scan implementation

This method builds and posts a SiteScanRequest.

```
def scansite(id)
  # Build the SiteScanRequest with the session ID and the site ID.
  body = "<SiteScanRequest session-id=\"#{@sessionid}\" site-id=\"#{id}\">
</SiteScanRequest>"
  # Send the request and receive the response.
  post(body)
  # Process response and return message depending on success
  # or failure.
  if @response.attributes["success"].to_i == 1
    puts "Scan started."
  else
    puts @response
  end
end

# Use the method with the API client -- scan the site that
# has site ID 12
client.scansite(12)
```

Vulnerability details implementation

This method builds and posts a `VulnerabilityDetailsRequest`. It then extracts a subset of the available information from the `VulnerabilityDetailsResponse` and produces formatted output.

```
def vulndetail(vulnid)
  # Build the VulnerabilityDetailsRequest with the session ID and the vuln ID.
  body = "<VulnerabilityDetailsRequest session-id=\"#{@sessionid}\" vuln-
id=\"#{@vulnid}\">"
  body << "</VulnerabilityDetailsRequest>"
  # Send the request and receive the response.
  post(body)
  # Process response and return message depending on success
  # or failure. If successful, extract data from the response.
  if @response.attributes["success"].to_i == 1
    puts "Title: #{@response.elements["Vulnerability"].attributes["title"]}"
    puts "Description: #{@response.elements["Vulnerability/description"].text}"
    puts "PCI Severity: #{@response.elements["Vulnerability"].attributes["pciSe-
verity"]}"
    puts "Severity: #{@response.elements["Vulnerability"].attributes["severity"]}"
  end
end

# Use the method with the API client -- request details of the
# vulnerability called "apache-buffer-overflow"
client.vulndetail("apache-buffer-overflow")
```

Ad hoc report generation implementation

The ReportAdhocGeneration API command is unusual. While the responses returned by the other commands are in XML format, a successful response to a ReportAdhocGenerationRequest is composed of two components: an XML message and a base64-encoded file, all wrapped in a multi-part MIME-encoded message:

```
--AxB9s13299asdjvbA
Content-Type: application/xml; charset=UTF-8; name=response_xml

<ReportAdhocGenerateResponse success="1"/>
--AxB9s13299asdjvbA
Content-Type: text/xml; name=report.xml
Content-Transfer-Encoding: base64

PE5leHBvc2VSZXBvcnQgdmVyc2lvcj0iMS4wIj4NCjxzY2Fucz4NCjxzY2FuIGlkPSI2IiBuYW1l
PSJBbm90aGVyIEExvY2FsIEhvc3QiIHN0YXJ0VGltZT0iMjAxMDA3MzBUMTIxNTU3MDQ2IiBlbmRU
aW1lPSIyMDEwMDczMFQxMjIxMDcyOTYiIHN0YXR1cz0iZmluaXNoZWQiLz4NCjwvc2NhbnM+PG5v
... lines deleted ...
PC90ZXhwb3NlUmVwb3J0Pg==DQo=
--AxB9s13299asdjvbA--
```

For this reason, the ReportAdhocGenerationResponse must be split into its components. After being separated, the XML message is parsed, and the base64-encoded file is decoded.

```
def report(templateid, format, siteid, filename="report")
  # Build the ReportAdhocGenerateRequest, including the session
  # ID, the report template ID, the ID of the site for which
  # the report is being run, and a filename for the generated
  # report.
  body = "<ReportAdhocGenerateRequest session-id=\"#{@sessionid}\">"
  body << "<AdhocReportConfig template-id=\"#{templateid}\" format=\"#{format}\">"
  body << "<Filters><filter type=\"site\" id=\"#{siteid}\"></filter></Filters>"
  body << "</AdhocReportConfig>"
  body << "</ReportAdhocGenerateRequest>"
  # POST the ReportAdhocGenerateRequest message with
  # "Content-type: text/xml" header
  @response = @client.post(@uri11, body, @ext_header).body
  # Parse the XML portion of the response.
  xmlresponse = REXML::Document.new(@response).root

  # If the ReportAdhocGenerateRequest was successful,
  # split the entire response into parts using the MIME
  # message boundary string as the delineator. Nexpose
  # uses the string --AxB9s13299asdjvbA as the boundary
  # string. One of the sections contains a content
  # header and the base64-encoded report. The report is
  # split from the header, decoded, and written to a file.
  if xmlresponse.attributes["success"].to_i == 1
    filename = filename + "." + format
    f = File.new(filename, "w")
    f.write @response.split(/--AxB9s13299asdjvbA/)[2].split
(/base64/).last.unpack('m')[0]
    f.close
    puts "Report generation request successful."
  else
    puts @response
  end
end

# Use the method with the API client -- produce a report on
# the site with ID 12 using the "audit-report" template
# in raw XML format. Write the report to a file called
# "myreport.raw-xml".
client.report("audit-report", "raw-xml", 12, "myreport")
```

Logout implementation

This method ends the client session and logs out the user.

```
def logout
  # Build the LogoutRequest XML message, including the session ID.
  body = "<LogoutRequest session-id=\"#{@sessionid}\"></LogoutRequest>"
  # Send the request and receive the response
  post(body)
  # Process response and return message depending on success
  # or failure.
  if @response.attributes["success"].to_i == 1
    puts "Logout of #{@sessionid} successful"
  else
    puts @response
  end
end
```

Appendix A: DTD Listings

This appendix includes DTDs for validating the API calls listed throughout this document.

Device DTD

```
<!DOCTYPE device [  
<!ELEMENT device (description?)>  
  <!-- the ID of the device -->  
  <!ATTLIST device id CDATA #REQUIRED>  
  <!-- the ID of the site the device belongs to -->  
  <!ATTLIST device site-id CDATA #IMPLIED>  
  <!-- the primary address or hostname of the device -->  
  <!ATTLIST device address CDATA #IMPLIED>  
  <!-- the current riskfactor (weighting) for the device -->  
  <!ATTLIST device riskfactor CDATA "1.0">  
  <!-- the current risk score of the device -->  
  <!ATTLIST device riskscore CDATA #IMPLIED>  
>
```

SiteSummary DTD

```
<!DOCTYPE SiteSummary [  
  
<!ELEMENT SiteSummary EMPTY>  
  <!ATTLIST SiteSummary id CDATA #REQUIRED>  
  <!ATTLIST SiteSummary name CDATA #REQUIRED>  
  <!ATTLIST SiteSummary description CDATA #IMPLIED>  
  <!ATTLIST SiteSummary riskfactor CDATA "1.0">  
  <!-- The riskscore stored in Nexpose is a computed value equal  
        to riskscore * riskfactor. The risk scores are only computed  
        after the site is scanned. This presents a problem when the  
        site administrator changes the site riskfactor. To account  
        for changing the riskfactor take the existing computed  
        riskscore divide by the old riskfactor and multiply by the  
        new riskfactor.-->  
  <!ATTLIST SiteSummary riskscore CDATA "0.0">  
  
>
```

Site DTD

```
<!DOCTYPE Site [  
<!ELEMENT Site (Hosts, Credentials, Alerting, ScanConfig)>  
  <!--Use id="-1" to create a new Site -->  
  <!ATTLIST Site id CDATA #REQUIRED>  
  <!ATTLIST Site name CDATA #REQUIRED>  
  <!ATTLIST Site description CDATA #IMPLIED>  
  <!ATTLIST Site riskfactor CDATA "1.0">  
<!ELEMENT Hosts ((range|host)+)>  
<!-- IPv4 address range of the form 10.0.0.1 -->  
<!ELEMENT range EMPTY>  
  <!ATTLIST range from CDATA #REQUIRED>  
  <!ATTLIST range to CDATA #IMPLIED>  
<!-- named host (usually DNS or Netbios name -->  
<!ELEMENT host (#PCDATA)>  
<!ELEMENT Credentials (adminCredentials*)>  
<!ELEMENT adminCredentials (#PCDATA|Headers|HTMLForms|PEMKey)>  
  <!-- cifs Concurrent Versioning System (CVS) -->  
  <!-- ftp File Transfer Protocol (FTP) -->  
  <!-- http HyperText Transfer Protocol (HTTP) -->  
  <!-- htmlform Web form authentication -->  
  <!-- httpheaders HTTP session authentication -->  
  <!-- as400 IBM AS/400 -->  
  <!-- notes Lotus Notes/Domino -->  
  <!-- tds Microsoft SQL Server -->  
  <!-- sybase Sybase SQL Server -->  
  <!-- cifs Microsoft Windows/Samba (SMB/CIFS) -->  
  <!-- oracle Oracle -->  
  <!-- mysql MySQL Server -->  
  <!-- db2 IBM DB2 Server -->  
  <!-- postgresql PostgreSQL Server -->  
  <!-- pop Post Office Protocol (POP) -->  
  <!-- remote execution Remote Execution -->  
  <!-- snmp Simple Network Management Protocol -->  
  <!-- ssh Secure Shell (SSH) -->  
  <!-- ssh-key Secure Shell (SSH) Public Key -->  
  <!-- telnet TELNET -->  
<!ATTLIST adminCredentials service CDATA #REQUIRED  
(cvs|ftp|http|as400|notes|htmlform|httpheaders|tds|sybase|cifs|  
oracle|mysql|db2|pop|postgresql|  
remote execution|snmp|ssh|ssh-key|telnet)>  
<!ATTLIST adminCredentials host CDATA #IMPLIED>  
  <!ATTLIST adminCredentials port CDATA #IMPLIED>  
  <!-- the userid, password and realm attributes should ONLY be used
```

```

if a security blob cannot be generated and the data is being
transmitted/stored using external encryption (eg, HTTPS)
SiteSaveRequest doesn't handle the security blob right now
So username/password attributes should be used in that case-->
<!ATTLIST adminCredentials userid CDATA #IMPLIED>
<!ATTLIST adminCredentials password CDATA #IMPLIED>
<!-- when using snmp assign the community name to the password attribute -->
<!ATTLIST adminCredentials realm CDATA #IMPLIED>
<!-- when using httpheaders, this represents the set of headers to pass with the
authentication request -->
<!ELEMENT Headers (Header+)>
<!-- A regular expression used to match against the response to identify authentica-
tion
failures. -->
<!ATTLIST Headers soft403 CDATA #IMPLIED>
<!-- the base URL of the application for which the form authentication applies. -->
<!ATTLIST Headers webapproot CDATA #IMPLIED>
<!ELEMENT Header (#PCDATA)>
<!ATTLIST Header name CDATA #REQUIRED>
<!ATTLIST Header value CDATA #IMPLIED>
<!-- when using htmlform, this represents the login form information -->
<!ELEMENT HTMLForms (HTMLForm+)>
<!-- the URL of the login page containing the login form -->
<!ATTLIST HTMLForms parentpage CDATA #IMPLIED>
<!-- A regular expression used to match against the response to identify
authentication failures. -->
<!ATTLIST HTMLForms soft403 CDATA #IMPLIED>
<!-- the base URL of the application for which the form authentication applies. -
->
<!ATTLIST HTMLForms webapproot CDATA #IMPLIED>
<!ELEMENT HTMLForm (Field*)>
<!-- the name of the form being submitted -->
<!ATTLIST HTMLForm name CDATA #IMPLIED>
<!-- the HTTP action (URL) through which to submit the login form -->
<!ATTLIST HTMLForm action CDATA #REQUIRED>
<!-- the HTTP request method with which to submit the form -->
<!ATTLIST HTMLForm method CDATA #IMPLIED>
<!-- the HTTP encoding type with which to submit the form -->
<!ATTLIST HTMLForm enctype CDATA #IMPLIED>
<!ELEMENT Field (#PCDATA)>
<!-- the name of the HTML field (form parameter) -->
<!ATTLIST Field name CDATA #IMPLIED>
<!-- the value of the HTML field (form parameter) -->
<!ATTLIST Field value CDATA #IMPLIED>
<!-- the type of the HTML field (form parameter) -->

```

```

<!ATTLIST Field type CDATA #IMPLIED>
  <!-- is the HTML field (form parameter) dynamically generated? If so, the login
page is requested and the value of the field is extracted from the response. -->
  <!ATTLIST Field dynamic CDATA #IMPLIED>
  <!-- if the HTML field (form parameter) is a radio button, checkbox or select
field,
this flag determines if the field should be checked (selected) -->
  <!ATTLIST Field checked CDATA #IMPLIED>
  <!-- when using ssh-key, this represents the PEM-format keypair information -->
<!ELEMENT PEMKey (#PCDATA)>
<!ELEMENT Alerting (Alert*)>
<!ELEMENT Alert (scanFilter?, vulnFilter?, (smtpAlert|snmpAlert|syslogAlert))>
  <!ATTLIST Alert name CDATA #REQUIRED>
  <!ATTLIST Alert enabled (0|1) "0">
  <!ATTLIST Alert maxAlerts CDATA>
<!ELEMENT scanFilter (#PCDATA)>
  <!ATTLIST scanFilter scanStart (0|1) "0">
  <!ATTLIST scanFilter scanStop (0|1) "0">
  <!ATTLIST scanFilter scanFailed (0|1) "0">
<!ATTLIST scanFilter scanPaused (0|1) "0">
  <!ATTLIST scanFilter scanResumed (0|1) "0"
<!ELEMENT vulnFilter EMPTY>
  <!-- severityThreshold defaults to 1. Currently Nexpose only supports values of 1
(Any Severity), 4 (Severe and Critical) and 8 (Only Critical). >
  <!ATTLIST vulnFilter severityThreshold (1|2|3|4|5|6|7|8|9|10) #REQUIRED>
  <!ATTLIST vulnFilter confirmed (0|1) "1">
  <!ATTLIST vulnFilter unconfirmed (0|1) "1">
<!ELEMENT smtpAlert (recipient+)>
  <!ATTLIST smtpAlert sender CDATA #IMPLIED>
  <!ATTLIST smtpAlert server CDATA #IMPLIED>
  <!ATTLIST smtpAlert port CDATA "25">
  <!ATTLIST smtpAlert limitText (0|1) "0">
<!ELEMENT recipient (#PCDATA)>
<!ELEMENT snmpAlert EMPTY>
  <!ATTLIST snmpAlert community CDATA>
  <!ATTLIST snmpAlert server CDATA #REQUIRED>
  <!ATTLIST snmpAlert port CDATA "162">
<!ELEMENT syslogAlert EMPTY>
  <!ATTLIST syslogAlert server CDATA #REQUIRED>
  <!ATTLIST syslogAlert port CDATA "514">

```

```
<!ELEMENT Users (user+)>
<!ELEMENT user EMPTY>
  <!-- the ID of a non-admin user who has access to this site -->
  <!ATTLIST user id CDATA #REQUIRED>
<!-- See the ScanConfig DTD for more details -->
]>
```

AssetGroupSummary DTD

```
<!DOCTYPE AssetGroupSummary [
<!ELEMENT AssetGroupSummary EMPTY>
  <!ATTLIST AssetGroupSummary id CDATA #REQUIRED>
  <!ATTLIST AssetGroupSummary name CDATA #REQUIRED>
  <!ATTLIST AssetGroupSummary description CDATA #IMPLIED>
  <!ATTLIST AssetGroupSummary riskscore CDATA #IMPLIED>
]>
```

AssetGroup DTD

```
<!DOCTYPE AssetGroup [  
  
<!ELEMENT AssetGroup (Devices)>  
  <!-- Use id="-1" to create a new Asset Group -->  
  <!ATTLIST AssetGroup id CDATA #REQUIRED>  
  <!ATTLIST AssetGroup name CDATA #REQUIRED>  
  <!ATTLIST AssetGroup description CDATA #IMPLIED>  
  <!ATTLIST AssetGroup riskscore CDATA #IMPLIED>  
  
<!ELEMENT Devices (device+)>  
  <!-- See the device DTD for more details -->  
<!ELEMENT Users (user+)>  
<!ELEMENT user EMPTY>  
  <!-- the ID of a non-admin user who has access to this site -->  
  <!ATTLIST user id CDATA #REQUIRED>  
  

```

EngineSummary DTD

Prior to the Nexpose release dated October 15, 2008, EngineSummaryResponse always returned “unknown” for EngineStatus values. As of October 15, 2008, the EngineSummaryResponse may return a value besides “unknown.”

```
<!DOCTYPE EngineSummary [  
  
<!ELEMENT EngineSummary EMPTY>  
  <!ATTLIST EngineSummary id CDATA #REQUIRED>  
  <!ATTLIST EngineSummary name CDATA #REQUIRED>  

```

ScanConfig DTD

```
<!DOCTYPE ScanConfig [  
  
<!ELEMENT ScanConfig (Schedules?)>  
  <!ATTLIST ScanConfig configID CDATA>  
  <!ATTLIST ScanConfig name CDATA>  
  <!-- Specify the scan template to use when starting a scan job -->  
  <!ATTLIST ScanConfig templateID CDATA #REQUIRED>  
  <!-- the scan engine to use. Omit to use the default engine -->  
  <!ATTLIST ScanConfig engineID CDATA #IMPLIED>  
  <!ATTLIST ScanConfig configVersion (3) "3">  
  
<!ELEMENT Schedules (Schedule*)>  
<!ELEMENT Schedule EMPTY>  
  <!ATTLIST Schedule enabled (0|1) "0">  
  <!ATTLIST Schedule type (daily|hourly|monthly-date|monthly-day|weekly) #IMPLIED>  
  <!ATTLIST Schedule interval CDATA>  
  <!-- the earliest date to run the scan on in ISO 8601 format,  
        YYYYMMDDTHHMMSSsss, such as: 19981231T00000000 -->  
  <!ATTLIST Schedule start CDATA #REQUIRED>  
  <!-- the amount of time, in minutes, to allow execution before stopping -->  
  <!ATTLIST Schedule maxDuration CDATA #IMPLIED>  
  <!-- the date after which the schedule is disabled in ISO 8601 format,  
        YYYYMMDDTHHMMSSsss, such as: 19981231T00000000 -->  
  <!ATTLIST Schedule notValidAfter CDATA #IMPLIED>  
  
>]
```

ScanSummary DTD

```
<!DOCTYPE ScanSummary [  
<!ELEMENT ScanSummary (message?, tasks?, nodes?, vulnerabilities*)>  
  <!ATTLIST ScanSummary scan-id CDATA #REQUIRED>  
  <!-- the site that was scanned -->  
  <!ATTLIST ScanSummary site-id CDATA #REQUIRED>  
  <!-- the engine the scan was dispatched to -->  
  <!ATTLIST ScanSummary engine-id CDATA #REQUIRED>  
  <!ATTLIST ScanSummary name CDATA #REQUIRED>  
  <!-- the scan start date and time in ISO 8601 format,  
        YYYYMMDDTHHMMSSsss, such as: 19981231T00000000 -->  
  <!ATTLIST ScanSummary startTime CDATA #REQUIRED>  
  <!-- the scan completion date and time in ISO 8601 format,  
        YYYYMMDDTHHMMSSsss, such as: 19981231T00000000 -->  
  <!ATTLIST ScanSummary endTime CDATA #IMPLIED>  
  <!ATTLIST ScanSummary status (running|finished|stopped|error|  
        dispatched|paused|aborted|unknown) #REQUIRED>  
<!ELEMENT message (#PCDATA)>  
<!ELEMENT tasks EMPTY>  
  <!ATTLIST tasks pending CDATA #REQUIRED>  
  <!ATTLIST tasks active CDATA #REQUIRED>  
  <!ATTLIST tasks completed CDATA #REQUIRED>  
<!ELEMENT nodes EMPTY>  
  <!ATTLIST nodes live CDATA #REQUIRED>  
  <!ATTLIST nodes dead CDATA #REQUIRED>  
  <!ATTLIST nodes filtered CDATA #REQUIRED>  
  <!ATTLIST nodes unresolved CDATA #REQUIRED>  
  <!ATTLIST nodes other CDATA #REQUIRED>  
<!ELEMENT vulnerabilities EMPTY>  
  <!ATTLIST vulnerabilities status (vuln-exploit|vuln-version|  
        vuln-potential|  
        not-vuln-exploit|  
        not-vuln-version|  
        error|disabled|other)  
        #REQUIRED>  
  <!-- the vulnerability severity (1-10, only provided with  
        vuln-exploit and vuln-version status) -->  
  <!ATTLIST vulnerabilities severity CDATA #IMPLIED>  
  <!-- the number of vulnerabilities with the specified status and severity -->  
  <!ATTLIST vulnerabilities count CDATA #REQUIRED>  

```

ReportTemplateSummary DTD

```
<!DOCTYPE ReportTemplateSummary [  
<!ELEMENT ReportTemplateSummary (description?)>  
  <!-- the id of the report template -->  
  <!ATTLIST ReportTemplateSummary id CDATA #REQUIRED>  
  <!-- the name of the report template -->  
  <!ATTLIST ReportTemplateSummary name CDATA #REQUIRED>  
  <!-- the visibility (scope) of the report template -->  
  <!ATTLIST ReportTemplateSummary scope (global|silo) #IMPLIED>  
  <!-- With a data template, you can export comma-separated value (CSV) files with  
  vulnerability-based data. With a document template, you can create PDF, RTF, HTML,  
  or XML reports with asset-based information. -->  
  <!ATTLIST ReportTemplateSummary type (data|document) #REQUIRED>  
  <!-- whether the report template is built-in, and therefore cannot be modified  
  (0=false, 1=true) -->  
  <!ATTLIST ReportTemplateSummary builtin (0|1) #REQUIRED  
  
<!ELEMENT description (#PCDATA)>  
>
```

ReportTemplate DTD

```
<!DOCTYPE ReportTemplate [  
  
<!ELEMENT ReportTemplate (description?,ReportSections?,Settings)>  
  <!-- the id of the report template -->  
  <!ATTLIST ReportTemplate id CDATA #REQUIRED>  
  <!-- the name of the report template -->  
  <!ATTLIST ReportTemplate name CDATA #REQUIRED>  
  <!-- the visibility (scope) of the report template -->  
  <!ATTLIST ReportTemplate scope (global|silo) #IMPLIED>  
  <!-- With a data template, you can export comma-separated value (CSV) files with  
  vulnerability-based data. With a document template, you can create PDF, RTF, HTML,  
  or XML reports with asset-based information. When you retrieve a report template,  
  the type will always be visible even though type is implied. When ReportTemplate  
  is sent as a request, and the type attribute is not provided, the type attribute  
  defaults to document, allowing for backward compatibility with existing API  
  clients. -->  
  <!ATTLIST ReportTemplateSummary type (data|document) #IMPLIED>  
  <!-- the report template is built-in, and cannot be modified  
  (0=false, 1=true) -->  
  <!ATTLIST ReportTemplate builtin (0|1) #REQUIRED  
  
<!ELEMENT description (#PCDATA)>  
  
<!ELEMENT ReportSections (ReportSection+,property*)>  
<!ELEMENT property (#PCDATA)>  
  <!-- the name of the property -->  
  <!ATTLIST property name CDATA #REQUIRED>  
  
<!ELEMENT ReportSection (property*)>  
  <!ATTLIST ReportSection name CDATA #REQUIRED>  
<!-- section specific content to include -->  
<!ELEMENT property (#PCDATA)>  
  <!-- the name of the property -->  
  <!ATTLIST property name CDATA #REQUIRED>  
  
<!ELEMENT Settings(showDeviceNames)>  
<!ELEMENT showDeviceNames EMPTY>  
  <!ATTLIST showDeviceNames enabled (0|1) "0">
```

ReportConfigSummary DTD

```
<!DOCTYPE ReportConfigSummary [  
  
<!ELEMENT ReportConfigSummary EMPTY>  
  <!-- the id of the report template -->  
  <!ATTLIST ReportConfigSummary template-id CDATA #REQUIRED>  
  <!-- the report definition (config) id -->  
  <!ATTLIST ReportConfigSummary cfg-id CDATA #REQUIRED>  
  <!-- the current status of the report -->  
  <!ATTLIST ReportConfigSummary status (Started|Generated|Failed|Aborted|Unknown)  
#REQUIRED>  
  <!-- the date and time the report was generated, in ISO 8601 format,  
       YYYYMMDDTHHMMSSsss, such as: 19981231T00000000 -->  
  <!ATTLIST ReportConfigSummary generated-on CDATA #REQUIRED>  
  <!-- the URL to use to access the report (not set for database exports) -->  
  <!ATTLIST ReportConfigSummary report-URI CDATA #IMPLIED>  
  <!ATTLIST ReportConfigSummary scope (global|silo) #IMPLIED>  
  
]>
```

ReportConfig DTD

```
<!DOCTYPE ReportConfig [  
  
<!ELEMENT ReportConfig (description?, Filters, Users, Baseline?, Generate, Delivery,  
DBExport?)>  
  
  <!-- the id of the report definition (config) -->  
  <!ATTLIST ReportConfig id CDATA #REQUIRED>  
  
  <!-- the unique name assigned to the report definition -->  
  <!ATTLIST ReportConfig name CDATA #REQUIRED>  
  
  <!-- With the site, device, and group filters you determine which assets to include  
  in the report. With the vuln-severity and vuln-status filters you control which vul-  
  nerabilities to include in the report. -->  
  <!ELEMENT AdhocReportConfig (Filters, Baseline?) >  
  
    <!-- the id of the report template used -->  
    <!ATTLIST ReportConfig template-id CDATA #REQUIRED>  
  
    <!ATTLIST ReportConfig format (pdf|html|rtf|xml|text|  
                                csv|db|raw-xml|raw-xml-v2|ns-xml|qualys-xml)  
    #REQUIRED>  
  
    <!ATTLIST ReportConfig owner CDATA #REQUIRED>  
  
    <!ATTLIST ReportConfig timezone CDATA #REQUIRED>  
  
  <!ELEMENT description (#PCDATA)>  
  
    <!-- The configuration must include at least one of device (asset), site, group  
    (asset group) or scan filter to define the scope of report. The vuln-status filter  
    can be used only with raw report formats: csv or raw_xml. If the vuln-status filter  
    is not included in the configuration, all the vulnerability test results (including  
    invulnerable instances) are exported by default in csv and raw_xml reports. -->  
  <!ELEMENT Filters (filter+)>  
  
  <!ELEMENT filter EMPTY> <!ATTLIST filter type (site|group|device|scan|vuln-sever-  
ity|vuln-status) #REQUIRED>  
  
  <!-- the ID of a specific site, group, device or scan.  
  For scan, this can also be "last" for the most recently run scan  
  For vuln-status, the ID can have one of the following values:  
  1)    vulnerable-exploited (The check was positive. An exploit verified the vulner-  
  ability.)  
  2)    vulnerable-version (The check was positive. The version of the scanned ser-  
  vice or software is associated with known vulnerabilities.)  
  3)    potential (The check for a potential vulnerability was positive.)  
  These values are supported for CSV and XML formats.  
  -->  
  <!ATTLIST filter id CDATA #REQUIRED>
```

continued on next page

ReportConfig DTD (continued)

```
<!ELEMENT Users (user+)>
<!ELEMENT user EMPTY>
  <!-- the ID of a non-admin user who has access to this site -->
  <!ATTLIST user id CDATA #REQUIRED>
<!ELEMENT Baseline EMPTY>
  <!-- the date to use as the baseline scan in ISO 8601 format,
  YYYYMMDDTHHMMSSsss, such as: 19981231T00000000. Additionally,
  "first" can be used for the first run scan, or "previous"
  for the most recently run scan prior to the current scan. -->
  <!ATTLIST Baseline compareTo CDATA #IMPLIED>
<!ELEMENT Generate (Schedule?)>
  <!-- will the report be generated after a scan completes (1),
  or is it ad-hoc/scheduled (0) -->
  <!ATTLIST Generate after-scan (0|1) "0">
  <!ATTLIST Generate schedule CDATA #IMPLIED>
<!ELEMENT Schedule EMPTY>
  <!ATTLIST Schedule enabled (0|1) "1">
<!ATTLIST Schedule type (daily|hourly|monthly-date|monthly-day|weekly) #REQUIRED>
<!ATTLIST Schedule interval CDATA #REQUIRED>
  <!-- the earliest date to generate the report on in ISO 8601 format,
  YYYYMMDDTHHMMSSsss, such as: 19981231T00000000 -->
  <!ATTLIST Schedule start CDATA #REQUIRED>
  <!-- the date after which the schedule is disabled in ISO 8601 format,
  YYYYMMDDTHHMMSSsss, such as: 19981231T00000000 -->
<!ATTLIST Schedule notValidAfter CDATA #IMPLIED>
<!ELEMENT Delivery (Storage, Email?)>
  <!-- See the Email DTD for more details -->
<!ELEMENT Storage (location?)>
  <!-- whether to store report on server -->
  <!ATTLIST Storage storeOnServer (0|1) "1">
  <!-- Directory location to store report in (for non-default storage) -->
<!ELEMENT location (#PCDATA)>
<!ELEMENT DBExport (credentials?, param*)>
  <!-- the db type to export to -->
  <!ATTLIST DBExport type CDATA #REQUIRED>
```

continued on next page

ReportConfig DTD (continued)

```
<!ATTLIST DBExport type CDATA #REQUIRED>
<!ELEMENT credentials (#PCDATA)>
  <!-- the userid, password and realm attributes should ONLY be used
        if a security blob cannot be generated and the data is being
        transmitted/stored using external encryption (eg, HTTPS) -->
  <!ATTLIST credentials userid CDATA #IMPLIED>
  <!ATTLIST credentials password CDATA #IMPLIED>
  <!-- DB specific, usually the database name -->
  <!ATTLIST credentials realm CDATA #IMPLIED>
<!ELEMENT param (#PCDATA)>
  <!-- the name of the parameter -->
  <!ATTLIST param name CDATA #REQUIRED>

]>
```

Email DTD

The sendAs and sendToAclAs attributes are optional, but one of them is required for sending reports via e-mail. The sendAs attribute is required for sending e-mails to users who are not on the report access list. The sendToAcl attribute is required for sending e-mails to report access list members.

If you do not set a valid value for either attribute, Nexpose will save the report but not send it via e-mail.

If you set a valid value for the sendAs attribute but not for the sendToAclAs attribute, Nexpose will send the report via e-mail to non-access-list members only.

If you set a valid value for the sendToAclAs attribute, Nexpose will send the report via e-mail to access-list members only.

If you set a valid value for both attributes, Nexpose will send reports via e-mail to access-list members and non-members.

NOTE: E-mails and attachments are sent via the Internet in cleartext and are not encrypted.

```
<!DOCTYPE Email [
<!ELEMENT Email (Recipients?, SmtRelayServer?, Sender?)
  <!-- send as file attachment or zipped file to individuals who are not members of
the report access list -->
  <!ATTLIST Email sendAs (file|zip) #OPTIONAL>
  <!-- send to all the authorized users of sites, groups and devices -->
  <!ATTLIST Email toAllAuthorized (0|1) "0">
  <!-- send to users on the report access listd file or the url-->
  <!ATTLIST Email sendToAclAs (file|zip|url) #OPTIONAL>
<!ELEMENT Recipients (Recipient*)>
<!ELEMENT Recipient (#PCDATA)>
<!ELEMENT SmtRelayServer (#PCDATA)>
<!ELEMENT Sender (#PCDATA)>
]>
```

ReportSummary DTD

```
<!DOCTYPE ReportSummary [  
  
<!ELEMENT ReportSummary EMPTY>  
  <!-- the id of the generated report -->  
  <!ATTLIST ReportSummary id CDATA #IMPLIED>  
  <!-- the report definition (config) id -->  
  <!ATTLIST ReportSummary cfg-id CDATA #REQUIRED>  
  <!-- the current status of the report -->  
  <!ATTLIST ReportSummary status (Started|Generated|Failed|Aborted|Unknown)  
#REQUIRED>  
  <!-- the date and time the report was generated, in ISO 8601 format,  
        YYYYMMDDTHHMMSSsss, such as: 19981231T00000000 -->  
  <!ATTLIST ReportSummary generated-on CDATA #IMPLIED>  
  <!-- the URL to use to access the report (not set for database exports) -->  
  <!ATTLIST ReportSummary report-URI CDATA #IMPLIED>  
  

```

UserConfig DTD

The current version of the API does not support creating user accounts with custom roles. You can only create user accounts with preset roles.

If values for allSites and allGroups are false or not specified, you can specify sites and groups using nested site and group elements.

You cannot change the user name after you create an account.

```
<!DOCTYPE UserConfig [  
  <!ELEMENT UserConfig (UserSite|UserGroup)*>  
  <!-- the id of the user, set to -1 to create a new user -->  
  <!ATTLIST UserConfig id CDATA #REQUIRED>  
  <!-- the role of the user -->  
  <!ATTLIST UserConfig role-name (global-admin|security-manager|site-admin|  
system-admin|user|custom) #REQUIRED>  
  <!-- the id of the authentication source for the user -->  
  <!ATTLIST UserConfig authsrcid CDATA #REQUIRED>  
  <!-- the login name of the user -->  
  <!ATTLIST UserConfig name CDATA #REQUIRED>  
  <!-- the full name of the user -->  
  <!ATTLIST UserConfig fullname CDATA #REQUIRED>  
  <!-- the email address of the user -->  
  <!ATTLIST UserConfig email CDATA #IMPLIED>  
  <!-- new password -->  
  <!ATTLIST UserConfig password CDATA #IMPLIED>  
  <!-- 1 to enable this user, 0 to disable -->  
  <!ATTLIST UserConfig enabled (0|1) #IMPLIED>  
  <!-- true if the user has access to all sites, false otherwise -->  
  <!ATTLIST UserConfig allSites (true|false) #IMPLIED>  
  <!-- true if the user has access to all groups, false otherwise -->  
  <!ATTLIST UserConfig allGroups (true|false) #IMPLIED>  
  
  <!-- See the UserSite DTD for more details -->  
  <!-- See the UserGroup DTD for more details -->  
>
```

User Site DTD

```
<!DOCTYPE Site [  
  <!-- the id of the site the user is associated with -->  
  <!ATTLIST Site id CDATA #REQUIRED>  
>
```

User Group DTD

```
<!DOCTYPE Group [  
  <!-- the id of the group the user is associated with -->  
  <!ATTLIST Group id CDATA #REQUIRED>  
>
```

UserSummary DTD

```
<!DOCTYPE UserSummary [  
  <!-- the id of the user -->  
  <!ATTLIST UserSummary id CDATA #REQUIRED>  
  <!-- the source used to authenticate this user -->  
  <!ATTLIST UserSummary authSource CDATA #REQUIRED>  
  <!-- the module used to authenticated this user -->  
  <!ATTLIST UserSummary authModule CDATA #REQUIRED>  
  <!-- the login name of the user -->  
  <!ATTLIST UserSummary userName CDATA #REQUIRED>  
  <!-- the actual name of the user -->  
  <!ATTLIST UserSummary fullname CDATA #REQUIRED>  
  <!-- the email address of the user (may be empty) -->  
  <!ATTLIST UserSummary email CDATA #REQUIRED>  
  <!-- true if this user is an administrator, false otherwise -->  
  <!ATTLIST UserSummary administrator (1|0) #REQUIRED>  
  <!-- true if this user is disabled, false otherwise -->  
  <!ATTLIST UserSummary disabled (1|0) #REQUIRED>  
  <!-- true if this user is locked, false otherwise -->  
  <!ATTLIST UserSummary locked (1|0) #REQUIRED>  
  <!-- the number of sites this user is allowed to access -->  
  <!ATTLIST UserSummary siteCount CDATA #REQUIRED>  
  <!-- the number of groups this user belongs to -->  
  <!ATTLIST UserSummary groupCount CDATA #REQUIRED>  
>
```

AuthenticatorSummary DTD

```
<!DOCTYPE AuthenticatorSummary [  
<!ELEMENT AuthenticatorSummary EMPTY>  
  <!-- the id of the authenticator -->  
  <!ATTLIST AuthenticatorSummary id CDATA #REQUIRED>  
  <!-- true if this authenticator authenticates using an external source,  
    false otherwise -->  
  <!ATTLIST AuthenticatorSummary external (0|1) #REQUIRED>  
  <!-- the name of the authenticator source -->  
  <!ATTLIST AuthenticatorSummary authSource CDATA #REQUIRED>  
  <!-- the name of the authenticator module -->  
  <!ATTLIST AuthenticatorSummary authModule CDATA #REQUIRED>  

```

XMLResponse DTD

This DTD provides the structure for the API response to a call a non-existent API function.

```
<!DOCTYPE XMLResponse [  
  
  <!-- This element makes sure that valid XML is returned when an error occurs.  
  <!ELEMENT XMLResponse (Failure)>  
    <!-- This attribute will always return 0 since it represents some kind of failure  

```

Failure DTD

```
<!DOCTYPE Failure [  
  
  <!-- The failure description, consisting of one or more message and/or exception -->  
  <!ELEMENT Failure ((message|Exception)*)>  
  
  <!-- the message describing the failure -->  
  <!ELEMENT message (#PCDATA)>  
    <!-- the source of the message, such as the module that caused the error -->  
    <!ATTLIST message source CDATA #IMPLIED>  
    <!-- the source specific message code -->  
    <!ATTLIST message code CDATA #IMPLIED>  
  
  <!-- the exception causing the failure -->  
  <!ELEMENT Exception (message, stacktrace?)>  
    <!-- the name of the Exception class (for Java or C++ exceptions) -->  
    <!ATTLIST Exception name CDATA #IMPLIED>  
  <!ELEMENT stacktrace (#PCDATA)>  
  ]>
```