Hewlett Packard Enterprise

HPE Network Node Manager i Software

Software Version 10.30

Developer’s ToolKit Guide

Contents

Overview

The HPE Network Node Manager i Software (NNMi) Developer's Toolkit highlights the features of NNMi. NNMi is available for developers to customize and enhance the core functionality of NNMi and integrate NNMi with other components and/or products.

This toolkit is comprised of this document, references to other documentation sources provided with the NNMi help system, and sample projects that demonstrate access to NNMi's web services.

Summary

The NNMi Developer's Toolkit encompasses product runtime configuration and customization aspects as well as API level access. While the runtime capabilities are available to any user having an administrator role, to gain access to the NNMi API documentation and samples for development of web-service clients, an NNMi Developer's License is required.

Runtime Configuration/Customization

The Configuration workspace enables you and your team to alter NNMi's out-of-the-box configuration settings. To distribute your enhancements, you can use command-line commands to export your changes. The export files can be imported onto other NNMi management servers. By using the Configuration → User Interface Configuration → Menu Items form, you can extend the Actions menu commands. This includes the ability to launch external tools while passing specific data from the NNMi database.

API – Programmatic Access

The NNMi API is Web Services based and offers read/write capabilities on all basic NNMi object types. WS-I endpoints are available for basic read/write capabilities. Additional custom behaviors are also offered through the WS-I services. For more details on web services, see the references listed at the end of this document.

Runtime Configuration/Customization

Configuration Workspace

From within the NNMi console, any user assigned to the administrator role can tailor various NNMi parameters and behaviors from a set of configuration forms. The details for this type of configuration are found in the NNMi console at Help → Help for Administrators when you log on as an administrator. This includes the ability to customize discovery, monitoring, and incident generation and correlation.

Import/Export Command Line Tools

All configurations available to administrators in the Configuration workspace are persisted in the NNMi database. Use provided export commands to preserve these settings for later recovery or to install onto another NNMi installation.

For details on the use of these tools, see **Help → Documentation Library → Reference Pages → nnmconfigexport.ovpl** and **nnmconfigimport.ovpl**.

This configuration import/export tool is the recommended procedure to use when integrators add custom configuration information to NNMi at component install time. For example, menu integrations (see next section about launch actions) should be imported as a post-install step using the `nnmconfigimport.ovpl` tool.

Exporting Configurations

Configurations that include an “Author” field should use this field to distinguish configuration records from multiple providers. Any configurations exported from NNMi should use the `-a` (author) option to export only those configurations that the integration component is responsible for. Using launch actions as an example, running `nnmconfigexport.ovpl -c` menuitem without the `-a` option exports ALL Actions menu items currently configured. Importing the results of the `nnmconfigexport.ovpl` discussed in this example results in overwriting the out-of-the-box provided configurations of NNMi or other components.

Exporting Incident Configurations

To export Incident configurations that include the addition of new Family or Category definitions REQUIRES the use of the `-a` option to `nnmconfigexport.ovpl` to include these new Family/Category records in the exported file.

Launch Action Menu Items

Use the **Menu Items** form (available from the Configuration workspace) to add menu items within the NNMi console. The menu items can provide access to tools that you provide, to other web based applications, to your web site, or to anything that can be accessed through a URL. You can pass NNMi object attribute values within the URL syntax, incorporating real-time data mining from the NNMi database. You can configure the Actions menu item to appear in the NNMi console only when your criteria are met.

See **Help for Administrators → Extending NNMi Capabilities → Control the Actions** for details, syntax, and limitations.

Additional Note for Developers: The variable substitution from object attributes to construct a URL for a launch action can be used to dynamically provide things such as protocol, host, and port values. While most object types might not have attributes that apply for such a substitution, the incident object type contains child incident attributes (CIAs) that can be useful to developers for this purpose. To accomplish this, the incoming trap from which the NNMi incident is created needs to have one or more varbinds with values that are suitable for substituting a protocol, host, and/or port. For example, assume a trap has varbinds with name/values such as the following:

- `com.my.protocol / https`
- `com.my.host / 199.166.1.1`
- `com.my.port / 8443`

The corresponding CIAs can then be used in a launch action for incidents as follows:

`${cias[name=com.my.protocol].value}:// ${cias[name=com.my.host].value}:`

`${cias[name=port].value}/…`

resulting in https://199.166.1.1:8443/...

Launch actions for Node and Interface objects can reference the `CustomAttributes` and `Capabilities` of these objects in the same way that CIAs are available for Incident objects. For example, assume a Node has the following custom attribute and capability:

- `com.my.report / nodeReport`

- com.my.nodeCapability

A launch action can be defined with a conditional filter whereby only those nodes that have the capability com.my.nodeCapability, will be available in the Actions menu to launch a parameterized launch action as follows:

http://myHost.my.com/myApp?${customAttributes[name=com.my.report]

resulting in http://myHost.my.com/myApp?nodeReport

The combination of NNMi's Extension Properties web services and launch actions enables custom integrations to take advantage of integration defined global properties in parameterized Launch action definitions. See the ExtProperties WS-I service description in the Web Service section of this document.

Launch actions also support enhanced integration context passing using Environment Attributes that allow the context to be saved in a launch into NNMi to be passed back to an application through a launch action. See the next section regarding launching NNMi URLs.

Launch NNMi URLs

Use URLs to provide access to the NNMi console or certain NNMi features from another component or application. For example:

- Embed views, forms, or maps within a Web portal.
- Launch an NNMi map within your application.
- Launch a filtered NNMi view from a browser window to display the information you need.
- Run an NNMi tool without opening the NNMi console.

See **Help for Administrators** → **Integrating NNMi Elsewhere with URLs** for more information. For examples on launching various NNMi views, forms, tools, etc., see **Help** → **NNMi Documentation Library** → **Integrate NNMi Elsewhere with URLs**.

Integrations between custom applications and NNMi can perform enhanced context passing by the use of Environment Attributes. These are special parameters that are added to the URL definition that launches an NNMi view but whose values can be saved and returned to the original application when a launch action is launched from NNMi. For example, assume that NNMi was launched from an application that wants to preserve a session ID (123) and object attribute (node25) as follows:

http://host/nnm?cmd=showView&objtype=Node&envattrs=com.my.sessionId=123;com.my.objectName=node25

Note

The name value pairs following the 'envattrs' above are application defined and might be any arbitrary string that serves to 'remind' the application of the context that was passed to NNMi. If the original application is re-launched from the resulting NNMi view with a launch action such as the following, the original context representing the application's session Id and the application's value for object name will be passed as well.

---

http://myHost/myApplication?com.my.sessionId=${getExtAttr(com.my.sessionId)}&com.my.objectName=$getExtAttr(com.my.objectName)}

Resulting in the following URL:

http://myHost/myApplication?com.my.sessionId=123&com.my.objectName=node25

Trap Generation

Two special varbind OIDs are interpreted by NNMi for use in the NNMi console Actions menu item, Path View. If these varbinds are present, NNMi creates the corresponding CIAs that NNMi uses to substitute the corresponding varbind value when launching a Path View from an Incident.

Varbinds with Corresponding CIAs

OID	CIA Name
.1.3.6.1.4.1.11.2.17.1.0.70008000	pathSrc
.1.3.6.1.4.1.11.2.17.1.0.70008001	pathDest

By default, when these CIAS are not present for an incident, NNMi uses the source node for the source of the Path View and “null” for the destination.

Lifecycle Transition Actions

Lifecycle transition actions provide for custom behaviors to be triggered based on incoming incidents or updates to the lifecycle state of an incident. These custom behaviors can be implemented in Jython scripts to be executed based on incident parameters that you configure (i.e., type of incident or lifecycle state).

See **Help for Administrators** → **Configuring Incidents** → **Configure an Action for an Incident** for details, syntax, and limitations.

API – Web Services

NNMi offers many WS-I compliant web services for reading, and updating the NNMi topology as well as related services for SNMP operations, reading/updating configuration and receiving a variety of notifications.

WS-Eventing is supported by NNMi to allow client web-services to subscribe to incident events from NNMi. By subscribing to the NNMi notification service, your client receives Incidents as they are added to the NNMi Event pipeline. In addition, any time an Incident's `lifecycleState` or `rcaState` changes, you can receive an update indicating the change.

The remainder of this document assumes that the reader has some basic understanding of web services concepts and Java programming, although Java is not a required platform for interoperating with NNMi's web services. For more details on web services, see the references listed at the end of this document.

Sample projects are provided with this document. (See the `nms-sdk-samples` folder of your installation where this document is located.) These samples demonstrate how to access each of the available web services. Additionally, there are sample Java classes available (see `nms-sdk-sources.jar`) to allow developers to quickly build the sample projects provided. Note, however, that these Java classes are simply examples and not directly required by a web service client. Instead, these classes (if used in a Java client) are automatically generated from the service WSDL by a third-party web service client generation tool (for example, Axis or Sun's Java Web Service Developer Pack).

Security

The NNMi Web Service APIs support BASIC authentication and require credentials to be passed that match a permitted NNMi user role. Allowed roles include Web Services Client, Admin, or System. Use the console Configuration: Users Accounts and Role workspace to create a user account having one of these roles.

Each of the client samples provided defaults to pass a username/password of `webservices/nnm`. This can be overridden at runtime.

WS-I APIs

The interface specification for each of the NNMi web-service APIs is governed by the service WSDL that might be found at the URL described in the table below for the <serviceType> of interest. For each service, see the corresponding description in the tables found in Object Attributes section of this document for details about attributes returned or expected on input. Each of these services also allows for filtered reading of objects from NNMi's data store. See the section on WS-I Filter Definition later in this document for more details.

URL Access to NNMi's WS-I Supported Services

URL for NNMi WS-I Services	<serviceType>
http://<host>:<port>/<service>BeanService/<service>Bean?wsdl	Incident
	IncidentConfiguration
	Node
	Interface
	IPAddress
	IPSubnet
	L2Connection
	Snmp
	NodeGroup
	Topology
	ExtProperties
	VLAN
	Mib
	FrameRelayEndpoint
http://<host>:<port>/NmsSdkService/<service>Bean?wsdl	Card
	Port
	Security
	WsRegistry

where:

- ***<host>*** is the server where NNMi is installed

- ***<port>*** is the port where NNMi is installed
- ***<service>*** is one of the supported NNMi services in the table above

For example, the WSDL for the Node service can be viewed on a machine where NNMi is installed (assuming port 80) by pointing a browser to the following URL: http://localhost/NodeBeanService/NodeBean?wsdl

Service Parameters

Object Identification

Unless specifically stated otherwise, most NNMi APIs accept either a 'persistence id' or a 'universally unique object identifier' where an object identifier is accepted as input. This value corresponds to the object's <id> attribute or if available its <uuid> attribute. In other words, they can be used interchangeably in most cases. In some other cases, where noted explicitly, a <uuid> is expected.

Attribute Values as Input Parameters

Many simple object attribute values are accepted as free form text String input. See sections on "Enumerations and Reserved Values' and 'Codes' in 'Inventory Object Attributes' for cases where the input parameter might be expected to conform to a specific format or value. For cases where the input parameter is a complex object type, the object and attributes will accompany the service description below.

Filtering results

See WS-I Filter Definition later in this document for more details on forming a query filter to customize the results returned from an inventory query using the get<objectType>() operations of most services.

Service Fault Handling

Many NNMi services will return an appropriate service fault for unrecoverable errors or invalid client input. When a fault is returned from an NNMi service, the request transaction is cancelled and no updates or return data should be expected. The fault type will depend on the service request (`Nms<serviceType>Fault`, for example). Below is a list of some common general purpose fault messages shared by multiple services. For service specific faults, see the service descriptions that follow. Additional troubleshooting information might also be found in the nnm-trace log at the time of the client request.

Table 3. Common Service Faults

Service Fault Message	Description
Cannot interrogate model: <exception>	An unrecoverable error has occurred while performing a query within the NNMi persistence layer. The corresponding exception accompanies the fault.
Cannot invoke service: <exception>	The internal NNMi support service implementation has thrown an unrecoverable exception. The corresponding exception accompanies the fault.
Internal server error locating service implementation	The internal NNMi support service or bean is unavailable or unresponsive.
Unable to locate <object>	Indicates that the object input identifier is either invalid OR the object has been removed (possibly by user activity).
Unable to <perform requested activity>	Indicates that the service was unable to perform the requested operation. An example is "Unable to delete incident: <id>". This is most likely due to invalid input or possibly internal error. Where possible, the service will return a more specific fault (e.g., "Unable to locate incident: <id>") if the failure was due to invalid input. Again, the underlying cause of invalid identifier input may be due to user or other processes having removed the desired object.

Incidents

The Incident service permits a client to read, update, and create Incident objects in the NNMi data store.

- `Incident[] getIncidents(Filter filter)`

Note

The filter must include a Constraint having includeCias with a value of true in order to include incident Cias in the result. By default, the incidents do not include Cias.

- Integer getIncidentCount(Filter filter)
  - Returns the count of objects that would be returned by getIncidents() using the same input filter.
- Incident[] getChildIncidents(String id)
  - Returns a complete list of correlated incidents for a given parent incident (corresponding to <id>) if there are any correlated children. Also see getChildCorrelations.
- addIncident(IncidentMgmtEvent)
  - See Incident table in section Object Attributes under Inject Mgmt Event column for IncidentMgmtEvent attribute details.
  - Note, this operation is asynchronous and only submits the supplied IncidentMgmtEvent to the Events subsystem pipeline for processing. There is no guarantee that the event will be persisted for later retrieval. For example, an incident may not be persisted if the indicated source node cannot be resolved.
  - Incidents added via addIncident are also not automatically forwarded to a global manager. An incident added to an NNMi regional manager must also be added to the global manager using addIncident on the global manager if that is the desired behavior.
- deleteIncident(String id)
- deleteIncidentByUuid(String uuid)
- updateNotes(String id, String notes)
  - The supplied 'notes' value will completely replace the current object's notes attribute value.
- updateLifecycleState(String id, String lifecycleState)
- updatePriority(String id, String priority)
- updateCias(String id, Cia[])
  - The supplied array of Cia's will completely replace the current incident's set of cia values. To augment this list, please read the current incident Cias using getIncidents prior to calling updateCias.
- IncidentCorrelation[] getChildCorrelations(String id)
  - Returns a complete list of child correlations, if any, grouped by their correlation type. Also see getChildIncidents.
- String[] getAssignToPrincipals()
  - Returns a list of principals that might be assigned to any given incident. See updateAssignedTo.
- void updateAssignedTo(String id, String assignTo)
  - Assigns an incident to a principal. See getAssignToPrincipals.

Cia object:

String name;

String type;

valid values for 'type' include:

{ "INTEGER", "STRING", "BOOLEAN", "DOUBLE", "UNKNOWN" }

String value;

IncidentCorrelation object:

String name;

CorrelationType type;

Valid values for type include:

{ "APA", "IMPACT", "DEDUP", "RATE", "PAIRWISE", "CUSTOM" }

String parent;

Date originOccurrenceTime;

```
int correlationResent;

String[] children;
```

Injection of Management Events

NNMi offers a web service endpoint that permits injection of management events into the NNMi event pipeline. Any management event that is configured with NNMi's Incident Configuration (see console's Configuration workspace) can be injected with the `addIncident` port on the Incident Service API shown above.

Incident Service Faults

Service Faults

Service Operation	Fault Message
getIncidents	Unable to retrieve incident CIAs: <id>
addIncident	Invalid nature: <nature> Invalid priority: <priority> Cannot inject incident: <name>
updateLifecycleState	The internal NNMi support service or bean is unavailable or unresponsive.
updateLifecycleState	Unable to update incident lifecycleState: <id>
updatePriority	Unable to update incident priority: <id>
updateAssignedTo	Invalid AssignTo principal: <principal>
getAssignToPrincipals	Unable to identify assignTo principals
updateNotes	Unable to update incident notes: <id>
deleteIncident	Unable to locate incident: <id> Unable to delete incident: <id>
deleteIncidentByUuid	Unable to locate incident: <uuid> Unable to delete incident: <uuid>
getChildIncidents	Unable to retrieve child incidents
getChildCorrelations	Unable to retrieve child correlations
updateCias	Unable to update Cias: <id>

Incident Configuration

The Incident Configuration service permits a client to read Incident Configurations in the NNMi data store. In this version, the input Filter objects are ignored and each operation returns the full set of configurations for the type specified. The attributes of each configuration object correspond to the values shown in the NNMi console.

See Help for **Administrators** → **Configuring Incidents** for more information about each configuration type.

- `RemoteNNMEventConfig[] getRemoteEventConfig(Filter filter)`
- `ManagementEventConfig[] getManagementEventConfig(Filter filter)`
- `SnmpTrapConfig[] getSnmpTrapConfig(Filter filter)`
- `PairwiseConfig[] getPairwiseConfig(Filter filter)`

`RemoteNNMEventConfig object:`

```
String id;

String name;

String incidentConfigType;

String enable;

String description;

String messageFormat;
```

String category;

String author;

String severity;

String family;

DedupConfig dedupConfig;

RateConfig rateConfig;

ActionConfig actionConfig;

String oId;

ManagementEventConfig object:

String id;

String name;

String incidentConfigType;

String enable;

String description;

String messageFormat;

String category;

String author;

String severity;

String family;

DedupConfig dedupConfig;

RateConfig rateConfig;

ActionConfig actionConfig;

PairwiseConfig object:

String id;

String name;

String incidentConfigType;

String enable;

String firstIncidentName;

String secondIncidentName;

PairItem[] pairItems;

DedupConfig object:

String enable;

String comparisonCriteria;

String correlationIncidentConfigName;

ComparisonParam[] comparisonParams;

ComparisonParam object:

String paramType;

String paramValue;

RateConfig object:

String enable;

int rateCount;

int hourInterval;

int minuteInterval;

int secondInterval;

String comparisonCriteria;

String correlationIncidentConfigName;

ComparisonParam[] comparisonParams;

ActionConfig

String enable;

LifecycleTransitionAction[] actions;

LifecycleTransitionAction

String commandType;

String command;

String lifeCycleState;

PairItem object;

String firstInPair;

String secondInPair;

PairItem object;

String firstInPair;

String secondIn Pair;

Node

The Node service permits a client to read, update, and delete Nodes from the NNMi data store.

- Node[] getNodes(Filter filter)

Note

The filter must include a Constraint having includeCustomAttributes with a value of true in order to include node CustomAttributes and Capabilities in the result. By default, the nodes do not include CustomAttributes and Capabilities.

- Integer getNodeCount(Filter filter)
  - Returns the count of objects that would be returned by getNodes() using the same input filter.
- NodeConclusion[] getConclusions(String id)
- addConclusion(String iD, String incidentId, Status status, String conclusion, String[] cancelledConclusions)

- Can be used to influence ‘status’ on a node. Status will be recalculated (with the most severe being set on the node).
- Conclusion strings can be any user meaningful description. As such, it is a recommended best practice to always prefix a conclusion string with either a namespace or a category to avoid collisions with other clients that might also be adding conclusion strings. For example, com.my_agent.ServiceUnavailable or OracleMonitorX_ServiceUnavailable are preferred to simply ServiceUnavailable. See “Reserved Node Conclusions” in section 4 for NNMi reserved conclusion values.
- The incidentId is optional (may be empty string) but if the conclusion is associated with an incidentId and another conclusion is later added that cancels the previous conclusion, the associated Incident will be closed. If it is not desired to associate a conclusion to an incident, provide an empty string (“”);
- ‘cancelledConclusions’ are an optional list of conclusions that the added conclusion will cause to be cancelled. If not used and no conclusions are intended to be cancelled, pass an empty list.
- It is strongly suggested that SDK clients do not add or cancel NNMi defined conclusions.

Conclusions might be deleted by supplying a blank ‘conclusion’ value along with the list of conclusions to be deleted as ‘cancelledConclusions’.

- updateManagementMode(String id, String managementMode)
- updateNotes(String id, String notes)
- addSeeds(String[] hostnamesOrIPs)
- addSeedsForTenant(String[] hostnamesOrIPs, String tenantName, String notes)
- addHints(String[] hostnamesOrIPs)
- removeSeeds(String[] hostnamesOrIPs)
- rediscoverHosts(String[] hostnamesOrIPs)
- addHints(String[] hostnamesOrIPs)
- pollHosts(String[] hostnamesOrIPs)
- addCapabilities(String id, Capability[] capabilities)
- String[] addCapabilitiesBulk(String[] ids, Capability[] capabilities)
  - Returns list of node IDs not successfully added.
- removeCapabilities(String id, String[] capabilityKeys)
  - See ‘Capability’ object description below. The list of capabilityKeys is expected to correspond to the ‘key’ attribute of the desired Capabilities to remove.
- String[] removeCapabilitiesBulk(String[] ids, String[] capabilityKeys)
  - Returns list of node IDs not successfully removed.
- updateCustomAttributes(String id, CustomAttribute[] customAttributes)
- addCustomAttributes(String id, CustomAttribute[] customAttributes)
- String[] addCustomAttributesBulk(String[] ids, CustomAttribute[] customAttributes)
  - Returns list of node IDs not successfully added.
- removeCustomAttributes(String id,String[] customAttributes)
- String[] removeCustomAttributesBulk(String[] nodeIds,String[] customAttributes)
  - Returns list of node IDs not successfully removed.
- boolean deleteNode(String id)
  - Returns ‘true’ if node was successfully deleted.
- boolean deleteNodeByUuid(String uuid)
  - Returns ‘true’ if node was successfully deleted.
- getNnmSystemName(final String id)

  - For Global Network Management, to obtain the system name for a given node.
- boolean isLocal(final String id)
  - For Global Network Management, to determine if a node is local (vrs remote) to the current NNMi management station.
- boolean isForwardable(final String id)
  - For Global Network Management, to determine if a node is being forwarded to a central NNMi management station.
- String[] getForwardableNodes(final String[] ids)
  - For Global Network Management, to determine which nodes are being forwarded to a central NNMi management station.

NodeConclusion object:

```
String uuid;
String status;
String conclusion;
Date timestamp;
```

Capability object:

```
String key;
String label;
```

CustomAttribute object:

```
String name;
String value;
```

Node Service Faults

Service Faults

Service Operation	Fault Message
updateManagementMode	Unable to update management mode: <id>
updateNotes	Unable to update notes: <id>
deleteNode	Unable to delete node: <id>
deleteNodeByUuid	Unable to delete node: <id>
addCapabilities	Unable to add capabilities: <id>
addCustomAttributes	Unable to add custom attributes: <id>
removeCustomAttributes	Unable to remove custom attributes: <id>
updateCustomAttributes	Unable to update custom attributes: <id>
addConclusion	Invalid incident id: <id> Unable to add conclusions: <id>
getNnmSystemName	Node not found: <id>
isLocal	Node not found: <id>
getForwardableNodes	Empty node list supplied

Interface

The Interface service permits a client to read and update existing Interface objects in the NNMi data store.

- Interface[] getInterfaces(Filter filter)

Note
The filter must include a Constraint having includeCustomAttributes with a value of true in order to include interface CustomAttributes and Capabilities in the result. By default, the interfaces do not include CustomAttributes and Capabilities.

- Integer getInterfaceCount(Filter filter)
  - Returns the count of objects that would be returned by getInterfaces() using the same input filter.
- InterfaceConclusion[] getConclusions(String id)
- updateManagementMode(String id, String managementMode)
- updateNotes(String id, String notes)
- addCapabilities(String id, Capability[] capabilities)
- String[] addCapabilitiesBulk(String[] ids,Capability[] capabilities)
  - Returns list of interface IDs that were not successfully added.
- removeCapabilities(String id, String[] capabilityKeys)
  - See 'Capability' object description below. The list of capabilityKeys is expected to correspond to the 'key' attribute of the desired Capabilities to remove.
- String[] removeCapabilitiesBulk(String[] ids,String[] capabilityKeys)
  - Returns list of interface IDs that were not successfully removed.
- updateCustomAttributes(String id, CustomAttribute[] customAttributes)
- addCustomAttributes(String id, CustomAttribute[] customAttributes)
- String[] addCustomAttributesBulk(String[] ifaceIds,CustomAttribute[] customAttributes)
  - Returns list of interface IDs that were not successfully added.
- removeCustomAttributes(String id,String[] customAttributes)
- String[] removeCustomAttributesBulk(String[] ifaceIds,String[] customAttributes)
  - Returns list of interface IDs that were not successfully removed
- Long[] getInputSpeedBulk(String[] ifaceIds)
- Long[] getOutputSpeedBulk(String[] ifaceIds)
- Integer addBulkCustomAttributes(String caName, String[] ids, String[] caValues)
  - 'caName' is a single custom attribute name to be added or updated for each of the interfaces specified by 'ifaceIds'.
  - 'caValues' is an array of values that will be added or updated for the 'caName' custom attribute of each of the interfaces of the corresponding 'ifaceIds' array.

InterfaceConclusion object:

```
String uuid;
String status;
String conclusion;
Date timestamp;
```

Capability object:

```
String key;
String label;
```

CustomAttribute object:

```
String name;

String value;
```

Interface Service Fault

Interface Service Faults

Service Operation	Fault Message
updateManagementMode	Unable to update management mode: <id>
updateNotes	Unable to update notes: <id>
addCapabilities	Unable to add capabilities: <id>
removeCapabilities	Unable to remove capabilities: <id>
addCustomAttributes	Unable to add custom attributes: <id>
removeCustomAttributes	Unable to remove custom attributes: <id>
updateCustomAttributes	Unable to update custom attributes: <id>

IP Address

The IP Address service permits a client to read and update existing IPAddress objects in the NNMi data store. The IPAddress object is IP-version neutral, supporting either IPv4 or IPv6 addresses. Though the tenant is not returned as part of the IPAddress object for backwards compatibility reasons, 'tenant' and 'mappedAddress' will be supported as attributes that might be used in the getIPAddresses(Filter filter) operation. When used as conditions in filters, these values are expected to correspond to the tenant name and mapped IP address value as would be returned by the `getTenantsByUuids()` and `getMappedAddressesByUuids()` operations respectively.

- `IPAddress[] getIPAddresses(Filter filter)`
- `IPAddressConclusion[] getConclusions(String id)`
- `updateManagementMode(String id, String managementMode)`
- `updateNotes(String id, String notes)`
- `addCapabilities(String id, Capability[] capabilities)`
- `String[] getTenantsByUuids(String[] uuids)`
- `String[] getMappedAddressesByUuids(String[] uuids)`

`IPAddressConclusion object:`

```
String uuid;

String status;

String conclusion;

Date timestamp;
```

IP Address Service Faults

Service Operation	Fault Message
getConclusions	Address id not found: <id> Unable to retrieve conclusions: <id>
updateManagementMode	Unable to update management mode: <id>
updateNotes	Unable to update notes: <id>
addCapabilities	Unable to add capabilities: <id>
removeCapabilities	Unable to remove capabilities: <id>
getTenantsByUuids	Empty address list supplied
getMappedAddressesByUuids	Empty address list supplied

IP Subnet

The IPSubnet service permits a client to read and update existing IPSubnet objects in the NNMi data store. The IPSubnet object is IP-version neutral, supporting either IPv4 or IPv6 subnets.

Although the tenant is not returned as part of the IPSubnet object for backwards compatibility reasons, 'tenant' will be supported as a filterable attribute in the `getIPSubnets(Filter filter)` operation. When used in a filter condition, this value is expected to correspond to the tenant name as would be returned by the `getTenantsByUuids()` operation.

- `IPSubnet[] getIPSubnets(Filter filter)`
- `updateNotes(String id, String notes)`
- `String[] getTenantsByUuids(String [] uuids)`

IP Subnet Service Faults

Service Operation	Fault Message
updateNotes	Unable to update notes: <id>
getTenantsByUnits	Empty subnet list supplied

L2 Connection

The L2Connection service permits a client to read and update existing L2Connection objects in the NNMi data store.

- `L2Connection[] getL2Connections(Filter filter)`
- `updateNotes(String id, String notes)`
- `String getSource(String id)`
  - Indicates the protocol used to discover the connection (e.g., FDB, CDP, LLDP, etc).

L2 Connection Service Faults

Service Operation	Fault Message
updateNotes	Unable to update notes: <id>
getSource	Connection id not found: <id>

Card

The Card service permits a client to read and update existing Card objects in the NNMi data store.

- `Card[] getCards(final Filter filter)`
- `updateManagementMode(String id, ManagementMode managementMode)`
- `addCapabilities(String id, Capability[] capabilities)`
- `removeCapabilities(String id, String[] capabilityKeys)`
  - See 'Capability' object description below. The list of capabilityKeys is expected to correspond to the 'key' attribute of the desired Capabilities to remove.
- `IncidentConclusion[] getConclusions(String id)`

Capability object:

```
String key;

String label;
```

Port

The Port service permits a client to read and update existing Port objects in the NNMi data store.

- `Port[] getPorts(final Filter filter)`
- `addCapabilities(String id, Capability[] capabilities)`
- `removeCapabilities(String id,String[] capabilityKeys)`

– See ‘Capability’ object description below. The list of capabilityKeys is expected to correspond to the ‘key’ attribute of the desired Capabilities to remove.

Capability object:

String key;

String label;

Topology

The Topology service permits a client to determine the network path between two hosts. The source and/or destinations of the path are not required to be managed by NNM; however, the access router for each must be a managed node within NNMi’s topology.

- NetworkPathResponse getPath(String startHostnameOrIP, endHostnameOrIP)

NetworkPathResponse object:

String pathSource;

String pathDest;

NetworkPathElement[] pathElements;

NetworkPathElement object:

PathElementType elementType;

String persistId;

String label;

String macAddress;

String modelClass;

boolean isDiscovered;

boolean isSource;

boolean isDestination;

PathElementType enumeration:

String: “NODE”, “ROUTER”, “SWITCH”, “INTERFACE”, “CLOUD”, “CONNECTION”

VLAN

The VLAN service permits a client to read existing VLAN and Port objects in the NNMi data store.

- VLAN[] getVLANs(Filter filter)
- Interface[] getInterfacesForVLAN(String vlanId)
  – ‘vlanId’ is the identifier of the vlan and not the object persistence id.
- VLAN[] getVLANsForInterface(String interfaceId)
- Port[] getPortsForVLAN(String vlanId)
  – ‘vlanId’ is the identifier of the vlan and not the object persistence id. See getPortsForVLANbyId below for an alternative based on persistence id.
- Port[] getPortsForVLANbyId(String id)
- VLAN[] getVLANsForDevice(String deviceId)

Port object:

String id;

String uuid;

```
String name;

String hostedOn; //a node id

String associatedInterface; //an interface id
```

SNMP

The SNMP service permits a client to read and update node SNMP configuration settings. All SNMP operations are synchronous and do not return until all results have been accumulated. Care should be exercised when making requests for multiple hosts, especially with many retries or long timeout values. Unspecified (null) input values of SnmpConfigOverrides are defaulted to the NNMi SNMP configuration for the given host if managed by NNMi. For NNMi unmanaged hosts, the defaults are indicated in the table below: The array of responses from this service's operations correspond to the array of hosts supplied as input.

Default Values if Host is Not Managed

Input Parameter	Default if Host is Not Managed
V2ReadCommunityString	public
V2WriteCommunityString	
snmpVersion	2
snmpPort	161
msTimeout	5000 milliseconds
retries	1
snmpProxyAddress	ignored if null
snmpProxyPort	ignored if snmpProxyAddress is null

- (deprecated) SnmpConfiguration[] getNodeConfiguration(String[] hostnames)
- (deprecated) setNodeConfiguration(String[] hostnames, String[] readCommunityStrings, String managementAddress, Integer snmpPort, Integer retries, Integer timeout)
- SnmpConfigResponse[] getNodeConfigByIds(String[] nodeIds)
- SnmpConfigResponse[] getNodeConfig (String[] hostnamesOrIPs)
- setNodeConfig(String[] hostnames, SnmpConfigOverrides config)
- SnmpResponse[] snmpGet(String[] reqOids, String[] hostnamesOrIPs, SnmpConfigOverrides config)
- SnmpResponse[] snmpGetNext(String[] reqOids, String[] hostnamesOrIPs, SnmpConfigOverrides config)
- SnmpResponse[] snmpGetBulk(String[] reqOids, String[] hostnamesOrIPs, SnmpConfigOverrides config, Integer nonRepeaters, Integer maxRepetitions)
  - nonRepeaters/maxRepetitions defaults to 0/5 respectively
- SnmpResponse[] snmpSet(String setOidStr, String value, Asn1Constant asnType, String[]hostnamesOrIPs, SnmpConfigOverrides config)
- String getOidForName(String name)
- String getNameForOid(String oid)
- String[] listDefaultReadCommunityStrings();
- addDefaultReadCommunityString(String commString);
- removeDefaultReadCommunityString(String commString);

SnmpConfiguration object (deprecated):

```
String nodeID;  //if discovered, null otherwise

String hostname;
```

```
        boolean isSnmpSupported;
        String[] readCommunityStrings;
        String activeReadCommunityString;   //if discovered and polled, null otherwise
        String managementAddress;               //configured address
        String activeManagementAddress; //address being used if discovered and polled
        String snmpVersion;
        int snmpPort;
        int timeoutSeconds;
        int retries;
SnmpConfigResponse object:
        String managementAddress;
        Integer snmpPort;
        Integer msTimeout;
        Integer retries;
        String snmpProxyAddress;
        Integer snmpProxyPort;
        String nodeID;
        String hostname;
        boolean isSnmpSupported;
        SnmpConfigV2 v2Config; //Configured v2 settings represents 'potential' settings
        SnmpConfigV3[] v3Config; //Configured v3 settings represents 'potential' settings
        SnmpConfigActive activeConfig; //Confirmed settings used by NNMi (if known, null        otherwise)
SnmpConfigOverrides object:
//Values supplied in this object are used to override the active configuration settings used by NNMi for SNMP communication. Null or empty values indicate that the NNMi configuration setting (or default setting if host is managed) should not be overridden.
String managementAddress;
Integer snmpPort;
Integer msTimeout;
Integer retries;
String snmpProxyAddress;
Integer snmpProxyPort;
private Integer snmpVersion; //Only used for SNMP queries if version is known by client
private String v2ReadCommunityString; //Ignored if null or if snmpVersion is known to equal 'V3' or if operation requires a writeCommunityString
```

private String v2WriteCommunityString; //Ignored if null or if snmpVersion is known to equal 'V3' or if operation is read-only

private SnmpConfigV3 v3Config; //Ignored if null or if snmpVersion is known to equal 'V1' or 'V2'

SnmpConfigV2 object:

String[] readCommunityStrings;

String writeCommunityString;

SnmpConfigV3 object:

String contextName;

String userName;

String authProtocol;

String authPassphrase;

String privProtocol;

String privPassphrase;

SnmpConfigActive object:

String managementAddress;

String snmpVersion;

String v2ReadCommunityString;

SnmpConfigV3 v3Config;

SnmpResponse object:

SnmpConfigResponse snmpConfig; //actual configuration used for indicated host

int errorStatus = 0; //non-zero indicates an SNMP request failure and errorIndex and varBinds are undefined

String errorStatusTranslation = ""; //Error string if errorStatus non-zero

int errorIndex = 0; //Error index returned from SNMP request

VarBindResponse[] varBinds; //Returned varbinds (valid if errorStatus==0)

VarBindResponse object:

String name;

String type;

String value;

String mnemonic;

NodeGroup

The NodeGroup service enables a client to read node groups from the NNMi data store.

- Node[] getNodeGroups(Filter filter)
- NodeConclusion[] getConclusions(String id)
- String[] getMemberIds(String id)
  - Returns an array of node id's that are currently members of the node group specified by the input node group id.

- NodeGroup[] getNodeGroupsByNode(final String nodeId)
  - Given a node id, returns an array of node group objects that the node is a member of.

ExtProperties

The ExtProperties service permits a client to read and add extended properties to the NNMi data store. This is basically a stand-alone table of custom attributes that a developer can use to persist simple configuration values. The attributes in this table are also available to parameterize launch actions. Each attribute is accessible with a combination of “domain” and “key” attributes. For example, assume that an integration developer would like to save the hostname and port of an application to be addressed by a launch action from the Actions menu. An integration could save the following values:

```
extPropService.setProperty(“myDomain”, “myHost”, “My Hostname”, “STRING”, “app.mycorp.com”);

extPropService.setProperty(“myDomain”, “myPort”, “My Port”, “STRING”, “80”);
```

With these values saved as Extension Properties, you can create a launch action as follows:

http://${getExtAttr(myDomain_myHost)}:${getExtAttr(myDomain_myPort)}/myApp

This permits the configuration of the host and port values of this example to remain separate from the launch action itself, allowing integrators to import a static launch action and handle configuration in a more dynamic fashion. If the application host and port change, the user can invoke a dynamic configuration process that updates the ExtProperties while leaving the launch action untouched.

The Extension Properties service is also convenient for saving credential information such as user names and passwords. Note, however, that such sensitive information should be encrypted before you save it as an Extension Property value. Also note that to access any Extension Property, the client must at least know the domain value under which the properties are saved.

An additional feature of Extension Properties is that there is a companion service ExtPropertiesRead that has a single operation, readPropertyValue. The readPropertyValue operation requires the domain and key value to access a property value.

- setProperty(String domain, String key, String label, String type, String value)
- removeProperty(String domain, String key)
- String getPropertyValue(String domain, String key)
- CustomAttribute[] getProperties(String domain)

Companion Service **ExtPropertiesRead**:

- String readPropertyValue(String domain, String key)

Mib

The Mib service permits a client to obtain a list of MIB variables saved in NNMi configuration.

- MibVariable[] getMibs(String filter)

```
MibVariable object:

        String oid;

        String name;

        String description;

        MibEnumeratedValue[] enumeratedValues;

        String fullName;

        String name;

        MibNotificationVariable[] notifVariables;

        String oid;

        MibTableIndex[] tableIndices;
```

String type;

MibTableIndex object:

String position;

String oid;

MibEnumeratedValue object:

String name;

Integer value;

MibNotificationVariable object:

Integer position;

String oid;

FrameRelayEndpoint

The FrameRelayEndpoint service permits a client to obtain NNMi discovered FrameRelayEndpoint data that might be associated with interfaces if NNMi has been enabled to discover such information.

- FrameRelayEndpoint[] getFrameRelayEndpoints(String filter)

FrameRelayEndpoint object:

String id;

String uuid;

Long dlci;

ManagementMode managementMode;

String interfaceId;

String notes;

Date created;

Date modified;

CustomAttribute[] customAttributes;

Capability[] capabilities;

Registry

The Registry service permits a client to obtain a list of registered web services made available by NNMi extensions (for example, NNM iSPIs). A client that needs to access a SPI provided web service for example, should first bind to http://<server>:<port>/WsRegistryBeanService/WsRegistryBean?wsdl web service endpoint on NNMi server to get the WsRegistry port. Then, the client can call getEndpoint providing the SPI advertised service name (or listEndpoints) to get the needed endpoint URL for the SPI service to be accessed.

- boolean register(final String serviceName, final URL endPoint)
  - to register a URL associated with a service name (unique to identify a web service across the available NNM iSPIs or extensions). If no such entry exists in the registry, a new entry is created; otherwise, the existing entry is updated with the new URL.
- boolean unregister(final String serviceName)
- java.net.URL getEndpoint(final String serviceName)
  - returns the URL associated with a service name used in the register() operation above
- String[] listServiceNames()
- java.net.URL[] listEndpoints()

Security

The Security service permits a client to determine the current security mappings between users and nodes via their associated User Group and node Security Groups. Nodes might be associated with multiple Security Groups and user accounts might be associated with multiple User Groups. These relationships determine which nodes (and their sub-objects) a user might have access to.

- SecurityGroupDTO[] listSecurityGroups()
- TenantDTO[] listTenants()
- NodeSecurityDTO[] getNodeSecurityDataByUuid(String[] nodeUuids)
- NodeSecurityDTO[] getNodeSecurityDataByName(String[] nodeNames)
- UserGroupMembershipDTO[] getUserGroupMembership(String[] userNames)
- SecurityGroupMembershipDTO[] getSecurityGroupMembershipByUserGroupName(String[] userGroupNames)
- SecurityGroupMembershipDTO[] getSecurityGroupMembershipBySecurityGroupUuid(String[] securityGroupUuids)
- TenantOAM[] getTenantOAMs(Filter filter)
- String createTenant(TenantDTO tenant)
  - Returns the uuid of the newly created tenant object.
  - The 'uuid' parameter of the supplied TenantDTO object might be null. In this case, NNMi will create a tenant and return it's uuid as the return value. Alternatively, if a valid 'uuid' is supplied, NNMi will create the tenant with the provided uuid value and also return it as the result.

SecurityGroupDTO object:

```
String name;
String uuid;
```

TenantDTO object:

```
String name;
String uuid;
String group;
```

TenantOAM object: //A mapped address pair for specified tenant

```
String tenantName;
String externalAddress;
String internalAddress;
```

NodeSecurityDTO object:

```
//Specifies a relationship between a node and tenant with a security group.
String nodeName;
String nodeUuid;
String securityGroupName;
String securityGroupUuid;
String tenantName;
String tenantUuid;
```

WS-I Filter Definition

The WS-I services above allow for filtered reading of objects from the NNMi data store. Below is a description of the filter definition used in these APIs.

The value for Date type Condition comparisons can be supplied as formatted Strings (e.g., “July 4, 2007”); however, these are interpreted using the web-service’s server locale. Alternatively, “Long” dates can be passed in as strings (e.g., in java, `Calendar.getInstance().getTimeInMillis().toString()`).

Filter is defined as one of the following:

- Expression

  `BooleanOperator (AND, OR, NOT, EXISTS, NOT_EXISTS)`

  `Filter[]`
- Condition

  `String: name`

  `Operator (‘=’, ‘!=’, ‘<’, ‘<=’, ‘>’, ‘>=’, ‘like’, ‘NOT IN’)`

  `String: value`
- Constraint

  `String: name`

  `String: value`

Valid Constraint

Name	Default Value
offset	‘0’
maxObjects	‘1000’
includeCias	‘false’
includeCustomAttributes	‘false’

Alternatively:

- <filter> ::= <booleanOperator> <list>
- <list> ::= <subFilter> | <subFilter> <list>
- <subFilter> ::= <constraint> | <condition> | <filter>
- <constraint> ::= <constraintName> <constraintValue>
- <condition> ::= <propertyName> <operator> <propertyValue>
- <booleanOperator> ::= <AND> | <OR> | <NOT> | <EXISTS> | <NOT_EXISTS>
- <operator> ::= <=> | <!=> | <<> | <<=> | <>> | <>=> | <like> | <NOT IN>
- <constraintName> ::= <offset> | <maxObjects>
- <constraintValue> ::= <string representation of a constraint value>
- <propertyName> ::= <name of an exposed NNMi attribute>
- <propertyValue> ::= <string representation of value>

Example Filter

```
Expression {
        AND
        Expression {
                OR
                Condition {id=1}
                Condition {id=2}
```

```
                Condition {id=3}
            }
            Condition {status>Minor}
            Constraint {offset=0}
            Constraint {maxObjects=20}
}
```

Note for Custom Attributes (and CIAs): With 8.1x, NNMi adds filtering on Node or Interface CustomAttributes as well as Incident CIAs. The name portion of the Condition must be prefixed with `customAttribute` for Nodes or Interfaces and with `customIncidentAttribute` for Incidents. For example, the following Java snippet shows how to create a filter to look for nodes or interfaces having a specific custom attribute value:

```
Condition c1=new Condition();
c1.setName( “customAttribute.name” );
c1.setOperator(Operator.EQ);
c1.setValue( “myCustomAttribute” );
Condition c2=new Condition();
c2.setName( “customAttribute.value” );
c2.setOperator(Operator.EQ);
c2.setValue( “someValue” );
Filter[] subFilters=new Filter[]{c1,c2};
Filter filter=new Expression();
Filter.setOperator(BooleanOperator.AND);
Filter.setSubFilters(subFilters);
```

Special Boolean Operators "EXISTS" and "NOT_EXISTS"

Two special Boolean operators that are supported by NNMi to allow for creating clauses that contain multiple custom attribute conditions for filtering out objects that have or do not have multiple custom attribute names and/or values. These special operators include the following:

- “EXISTS” permits you to locate objects with multiple custom attribute names and/or values.
- “NOT_EXISTS” permits you to locate objects that DO NOT contain a particular custom attribute.

The following Java snippet demonstrates how to define a filter using multiple custom attribute conditions to find objects that contain custom attributes having *either* a specific value *or* not having the custom attribute at all.

```
// EXIST clause
Condition condition1 = new Condition();
condition1.setName("customAttribute.name");
condition1.setOperator(Operator.EQ);
condition1.setValue("test");
Condition condition2 = new Condition();
condition2.setName("customAttribute.value");
condition2.setOperator(Operator.EQ);
```

```
condition2.setValue("0");
Filter[] subFilters = new Filter[]{condition1, condition2};
Filter existFilter = new Expression();
existFilter.setOperator(BooleanOperator.AND);
existFilter.setSubFilters(subFilters);

Filter[] existSubFilters = new Filter[]{existFilter};
Expression exist = new Expression();
exist.setOperator(BooleanOperator.EXISTS);
exist.setSubFilters(existSubFilters);

// NOT_EXIST clause
Condition condition3 = new Condition();
condition3.setName("customAttribute.name");
condition3.setOperator(Operator.EQ);
condition3.setValue("test");
Filter[] notExistSubFilters = new Filter[]{condition3};
Expression notExist = new Expression();
notExist.setOperator(BooleanOperator.NOT_EXISTS);
notExist.setSubFilters(notExistSubFilters);
Filter[] allFilters = new Filter[]{exist, notExist};

// If an node has custom attribute test=0 or no test custom attribute at all
Filter filter=new Expression(BooleanOperator.OR, allFilters);
Node[] nodeService.getNodes(filter); c1.setValue( “myCustomAttribute” );
```

Special Operators "like" and "NOT IN"

Two special operators that are supported by NNMi filtering include the following:

- “like” permits you to locate object values that are similar but not exact matches to the value stored in the database.
- “NOT IN” permits you to locate objects that DO NOT contain a particular custom attribute.

“like” Operator

The “like” operator permits the use of wildcards in the value specified for the condition. The syntax of a value specified using the “like” operator is the same as that used in any SQL query and includes the following wildcards:

% permits you to match any string of any length including zero length

_ permits you to match on a single character

Examples:

```
Condition c=new Condition();
```

```
c.setName(" name");

c.setOperator(Operator.LIKE);

c.setValue( "%Down%");

includes all results that contain the word  "Down"  anywhere in the name

Condition c=new Condition();

c.setName(" customAttribute.value");

c.setOperator(Operator.LIKE);

c.setValue( "123_");
```

includes all records containing custom attribute values having 4 character values that start with "123"

"**NOT_IN**" Operator

The "NOT_IN" operator locates records that do not have the specified custom attribute name.

Examples:

```
Condition c=new Condition();

c.setName(" customAttribute.name");

c.setOperator(Operator.NOT_IN);

c.setValue( "myCustomAttribute");
```

returns any records that do not contain a custom attribute having the name "myCustomAttribute" (including records that have no customAttributes)

WS-I Examples

To demonstrate the NNMi web services APIs, there are several example client implementations:

wsi-inventory-client

This client exercises the read/update APIs of the various WS-I Inventory Services (Incident, Node, Interface, IPAddress, IPSubnet, L2Connection, Card, Port).

wsi-incident-generation-client

This client exercises the WS-I Incident Service support for Incident injection (the addIncident operation).

wsi-snmp-client

This client exercises the configuration read/update APIs for NNMi's SNMP configuration for Nodes as well as get, getNext, getBulk and set SNMP operations.

wsi-topology-client

This client exercises the ability to get a network path using the Topology service.

wsi-nodegroup-client

This client exercises NodeGroup service to read node group attributes and determine node membership.

wsi-extproperties-client

This client exercises ExtProperties service to save name/value pair data in the NNMi database.

wsi-incident-config-client

This client exercises the IncidentConfiguration service to read management and trap configuration information from the NNMi database.

wsi-registry-client

This client exercises the web-service registration capabilities of the WsRegister service.

wse-notification-client

This client exercises ws-e notification service for Incident, Node and SNMP Notifications

dom4j-notification-servlet-client

This client exercises the ws-e notification service for IncidentNotifications using a simple servlet.

Building Examples

1. Client examples can be built from the project root folder, `samples`, using Maven (see references at end of this document for more information about Maven). Set JAVA_HOME to ${nnmInstallDir}/nonOV/jdk/nnm to set the proper Java development environment for Maven.
2. Type `mvn clean install` from the sample project root folder, `samples`. In the 'target' folders of the client sub-folders, you should find the corresponding client jar files. These are packaged as a jboss ear file in the `wsi-samples-ear/target` folder as `nms-sdk-samples.ear`. This 'ear' file contains all the sample clients and might be deployed in any jboss installation (including NNMi). Note, the individual client jars are also deployable in the jboss deploy folder, but these require a jboss restart to hot redeploy. The `nms-sdk-samples.ear` might be hot redeployed without a jboss restart.

Running Examples

To run the sample incident injection client, follow these steps:

1. Deploy the `nms-sdk-samples ear` file(s) into a running jboss application server (version 6.0 or greater with jbossWS version 2.0.1).

Note
You can also deploy the ear file and then start up jboss.

---

2. Point a browser at the jboss jmx-console where you deployed the sample ear file(s).

Note:
This might be your NNMi installed jmx-console if this is where you chose to deploy the sample client.

---

3. Locate the management bean for the sample clients under the `com.hp.ov.nms.sdk…` (for example, `com.hp.ov.nms.sdk.inventory`, `com.hp.ov.nms.sdk.incident`, or `com.hp.ov.nms.sdk.snmp` sections for the inventory, incident, and snmp clients respectively.

   `com.hp.ov.nms.sdk.inventory`:

   mbean=InventoryClient

   `com.hp.ov.nms.sdk.incident`:

   mbean=IncidentGeneratorClient

   `com.hp.ov.nms.sdk.snmp`:

   mbean=SnmpClient

   Each client contains an "attributes" section at the top where you can update parameters used by the client. For example, each client needs to know the actual location of the service it invokes as well as the necessary credentials. Other attributes might be optional depending on the service.

   Common attributes for these sample clients include wsURL, username, and password. Under wsURL, update "localhost" and port "80" to reflect the location of your NNMi installation.

4. If you update any attributes, be sure to select Apply Changes to apply your changes to the client management bean's internal values.

Inventory Client

Each of the `listXxx()` operations available from the inventory client has a “canned” example filter that it uses to pass to the corresponding service’s `getXxx()` port. When you invoke one of these operations, it should show the filter being used along with the current set of objects returned by the service using that filter.

Incident Generator Client

To inject an incident into the Incident Service, follow these steps:

1. Enter the name of a configured management event (e.g., NodeDown) under the “ParamValue” edit box for the `addIncident()` operation.
2. Select **Invoke**.

> **Note:**
> While you might enter additional incident attributes, most other incident input values have defaults so only the incident Name (e.g., “NodeDown”) is required.

3. Verify that the Incident Service received the client input by opening the NNMi console’s All Incidents view.

SNMP Client

To obtain the configuration settings for a given hostname or address, enter the hostname or address under the “ParamValue” edit box for the `getNodeConfiguration()` operation and then select Invoke.

To update the read community string or management address for a given hostname/address, follow these steps:

1. Enter the hostname or address in the first “ParamValue” edit box for the `setNodeConfiguration()` operation.
2. Enter the read community string in the second box.
3. Enter the desired management address in the third box.
4. Select **Invoke**.
5. Verify the update by performing the `getNodeConfiguration` for the same hostname.

> **Note**
> While the client only demonstrates updating a single hostname, the SNMP Service permits for reading/updating multiple hostnames since the input parameter for each operation takes a list of hostnames. The same is true for updating the read community strings. The sample is more restrictive than the actual API.

Other Clients

The remaining clients follow a similar pattern to the others described above with invoke methods to test key functionality. To determine the precise input expected by any given MBean operation, you might refer to the corresponding client source code in the sample project for that MBean.

WS-Eventing

NNMi offers a notification service that clients can subscribe to. Notifications come from WS-Eventing. A client subscribes to the NNMi notification service by supplying an “Endpoint Reference” to a web service that they create for receiving notifications.

Incident Subscription

NNMi notifies subscribers when new Incidents are created or when an Incident’s `lifecycleState` or `rcaActive` is updated. Incident Correlations are also sent as notifications starting with NNMi 9.00. New incidents and updates are sent as IncidentNotification messages and correlations appear as IncidentCorrelation messages (see the Incident service above for more details about this object). Some of the Incident attributes are designated with a “trigger”. These are attributes that trigger another notification to be sent for the incident as NNMi changes these attributes values. The changed attribute values are included in the message. Additionally, a time stamp (see `updateTime`) and the previous values for these attributes are included when the update occurs. You can expect these previous values and the `updateTime` to be empty or null when the initial incident notification is received for a brand new incident.

IncidentNotification message:

String id;

String uuid;

String sourceUuid;

String sourceName;

String sourceNodeUuid;

String sourceNodeName;

String sourceNodeLongName;

String name;

String lifecycleState;

Severity severity;

String priority;

String category;

String family;

Origin origin;

Nature nature;

String formattedMessage;

int duplicateCount;

boolean rcaActive;

Date originOccurrenceTime;

Date firstOccurrenceTime;

Date lastOccurrenceTime;

Date created;

Date updateTime;

int incidentResent;

String previousLifecycleState;

String previousRcaActive;

Cia[] cias;

The event source for incident notifications is:

http:// notification.sdk.nms.ov.hp.com/nms-sdk-notify/IncidentNotificationSource

The WSDL describing the notification service and the notification messages can be found at:

http://<host>:<port>/nms-sdk-notify/subscribe?wsdl

where:

<*host*> is the server where NNMi is installed

<*port*> is the port where NNMi is installed

Handling Incident Notification Failures

It is possible that a notification attempt to a given subscriber will fail to be sent. In such cases, the NNMi Developer Toolkit detects these failures and resends the notification. This condition can be detected by examining the `incidentResent` property of a received `IncidentNotification` event. A non-zero value indicates that this event was attempted and failed. The value indicates the number of retries attempted. Retries are discontinued after 3 attempts (i.e., `incidentResent=2` indicates that the message had been attempted 3 times).

Because it is not possible to detect which subscriber did not receive the delivery, the notification service resends the notification to all subscribers. To avoid processing multiple, duplicate events, it might be important to examine the `incidentResent` property to determine if this is a retry attempt and if the event was already received and processed (based on its id and `updateTime`).

Node Subscription

NNMi notifies subscribers when new Nodes are created, updated, or deleted. All notifications are sent as NodeNotification messages. Note that the `nodeRediscovered` event is sent by NNMi when a node is periodically rediscovered by NNMi even if it has not detected any changes to the node. A special flag `discoveryAnalysisUpdated` is also set to `true` by the discovery engine when a node's addresses, interfaces, or router redundancy groups are changed. A read is required on the node or associated sets to determine the actual changes that have occurred for the node. Additional attributes for node notifications include the event type and update time.

With NNMi 10.20, two new event types, nodeRediscovered2 and nodeResynchronized, have been added to accommodate the new ***full resynchronization*** feature of NNMi. When a user passes the *–fullsync* option on the nnmnoderediscover.ovpl command line, NNMi refreshes the node discovery information and a forced resynchronization of the node state and status is done. The feature helps NNMi users eliminate inconsistencies in the data being presented by NNMi. For details on this feature, see the *Synchronize Regional Manager Discovery from a Global Manager* section in the *NNMi 10.20 Deployment Reference*.

The nodeResynchronized event is sent to indicate that NNMi has performed a full resynchronization. For backwards compatibility, this new event will coincide with a nodeRediscovered event. The nodeRediscovered2 event is equivalent to (and sent simultaneously) with nodeRediscovered events with the exception that nodeRediscovered2 does ***not*** accompany a nodeResynchronized event. Notification subscribers that wish to process the new nodeResynchronized event can easily filter out the older nodeRediscovered event with a filter string such as:
"`/sys:onNotification/arg0[event!='nodeRediscovered']`". This permits a client to replace their handling of nodeRediscovered with the event nodeRediscovered2 without receiving both messages.

NodeNotification message:

String event;

String id;

String uuid;

String longName;

Boolean isSnmpSupported;

int nodeResent;

Date updateTime;

Boolean discoveryAnalysisUpdated;

CustomAttribute[] customAttributes;

Date discoveryLastCompleted;

ManagementMode managementMode;

Node Notification Update Events

Event	Description
nodeDiscovered	A new node has been discovered by NNMi
nodeRediscovered	A node has been re-examined by the NNMi discovery engine (see the "`discoveryAnalysisUpdated`" flag to indicate if node discovery

	conditions have changed – addresses, interfaces, or router redundancy group)
nodeRediscovered2	Same as 'nodeRediscovered' except that while 'nodeRediscovered' is sent simultaneously with 'nodeResynchronized', 'nodeRediscovered2' is not.
nodeResynchronized	NNMi discovery engine has performed a full node resynchronization from a regional NNMi node.
nodeUpdated	Management mode was changed by user or by NNMi
nodeDeleted	Node has been deleted

The event source for node notifications is:

http:// notification.sdk.nms.ov.hp.com/nms-sdk-node-notify/NodeNotificationSource

The WSDL describing the node notification service and the notification messages can be found at:

http://<host>:<port>/nms-sdk-node-notify/subscribe?wsdl

where:

<host> is the server where NNMi is installed

<port> is the port where NNMi is installed

Handling Node Notification Failures

It is possible that a notification attempt to a given subscriber will fail to be sent. In such cases, the NNMi Developer's Toolkit detects these failures and resends the notification. This condition can be detected by examining the nodeResent property of a received NodeNotification event. A non-zero value indicates that this event was attempted and failed. The value indicates the number of retries attempted. Retries are discontinued after 3 attempts (i.e., nodeResent=2 indicates that the message had been attempted 3 times).

Because it is not possible to detect which subscriber did not receive the delivery, the notification service resends the notification to all subscribers. To avoid processing multiple, duplicate events, it might be important to examine the nodeResent property to determine if this is a retry attempt and if the node event was already received and processed (based on its id and updateTime).

SNMP Configuration Change Subscription

NNMi notifies subscribers when the SNMP configuration for Node are updated. All notifications are sent as SnmpNotification messages. A read is required on the Snmp service or the node(s) indicated to determine the actual changes that have occurred for the node's snmp configuration.

Snmp Notification Configuration Update Events

Event	Description
nodeIds	A list of node IDs for which SNMP configuration has changed.

The event source for snmp notifications is:

http:// notification.sdk.nms.ov.hp.com/nms-sdk-snmp-notify/SnmpNotificationSource

The WSDL describing the snmp notification service and the notification messages can be found at:

http://<host>:<port>/nms-sdk-snmp-notify/subscribe?wsdl

where:

<host> is the server where NNMi is installed

<port> is the port where NNMi is installed

Security Change Notification Subscription

NNMi notifies subscribers when the security mappings for nodes are updated. All notifications are sent as NodeSecurityChangeNotification messages.

NodeSecurityChangeNotification message:

String nodeUuid;

String currentSecurityGroupName;

String currentSecurityGroupUuid;

String previousSecurityGroupName;

String previousSecurityGroupUuid;

Date occurrenceTime;

The event source for node security change notifications is:

http:// notification.sdk.nms.ov.hp.com/nms-sdk-node-security-change-notify/NodeSecurityChangeNotificationSource

The WSDL describing the node notification service and the notification messages can be found at:

http://<host>:<port>/nms-sdk-node-security-change-notify/subscribe?wsdl

where:

<host> is the server where NNMi is installed

<port> is the port where NNMi is installed

Tenant Change Notification Subscription

NNMi notifies subscribers when the tenancy for nodes is updated. All notifications are sent as NodeTenantChangeNotification messages.

NodeTenantChangeNotification message:

String nodeUuid;

String currentTenantName;

String currentTenantUuid;

String previousTenantName;

String previousTenantUuid;

Date occurrenceTime;

The event source for node tenant change notifications is:

http:// notification.sdk.nms.ov.hp.com/nms-sdk-node-tenant-change-notify/NodeTenantChangeNotificationSource

The WSDL describing the node tenant change notification service and the notification messages can be found at:

http://<host>:<port>/nms-sdk-node-tenant-change-notify/subscribe?wsdl

where:

<*host*> is the server where NNMi is installed

<*port*> is the port where NNMi is installed

WS-E Examples

To demonstrate subscribing to and receiving NNMi notifications, there are two example client implementations: wse-notification-client and wse-notification-ejb-client.

wse-notification-client

This client demonstrates subscribing to the NNMi notification services and receiving NNMi incidents for EJB3 beans exposed as web services. This is packaged as an ear file that might be deployed in any jboss application server (including NNMi's).

Due to a defect in jboss where a system call to determine the host name running NNMi (Windows only) returns the short host name and not the fully qualified domain name, https access to the full wsdl of the NNMi eventing services cannot be resolved when programmatically accessing the wsdl using the its URL located on that NNMi host system. An alternative is to capture the eventing wsdls and their referenced schema documents and package those files as resources into any NNMi subscribing client. The wse-notification-client does just that. The full set of wsdl documents for all NNMi notification services are available with the Java source for this client.

dom4j-notification-servlet-client

This client demonstrates subscribing to the NNMi NodeNotification service and receiving NNMi incidents. This is packaged as a simple servlet that might be deployed in any servlet container (including NNMi's jboss application server).

Building Examples

1. Client examples can be built from the project root folder, `nms-sdk-samples`, using Maven (see references at end of this document for more information about Maven).
2. Type `mvn clean install` from the root folder. In the target folders of wse-notification-client and dom4j-notification-servlet-client, you should find the corresponding client deployable files (the former is a "jar" file and the later a "war" file). Note, these deployables are both contained within the nms-sdk-samples.ear described in the 'WS-I Examples' section.

Running Examples

wse-notification-client

To run this sample client, follow these steps:

1. Deploy the nms-sdk-samples ear file into a running jboss application server (version 4.3.0 or greater with jbossWS version 2.0.1).

Note
You can also deploy the ear file and then start up jboss.

---

2. Point a browser at the jboss jmx-console where you deployed the sample ear file. This might be your NNMi installed jmx-console if this is where you chose to deploy the sample client.
3. Locate the management bean for the sample client in the `com.hp.ov.nms.sdk.notification` section.

   `com.hp.ov.nms.sdk.notification:`

   mbean=NotifyClient

4. Verify that the credential information and host/port values are correct for the Username, Password, and wsURL attributes of this bean. Update as necessary and select **Apply Changes.**
5. Select the **subscribeXxx()** invoke button where 'Xxx' is replaced by either 'Incidents', 'Nodes', or 'Snmp') to subscribe to the desired NNMi notification service. You should see subscription details to indicate that the bean subscribed to successfully.
6. Select the **Back to MBean View** link.
7. Select the **testXxxNotification()** invoke button (again where Xxx is replaced by either 'Incidents', 'Nodes', or 'Snmp') to verify that the "Event Sink" service receives a notification.
8. The Event Sink takes the name of the Incident Event message and passes this value back to the NotifyClient. To verify, select the **Back to MBean View** link again, and make sure that the count and value properties that apply to the type of notification you tested have been updated. For example, if you invoked testNodeNotification, you should see the nodeCount and lastNodeReceived properties updated.

dom4j-notification-servlet-client

To run this sample client and subscribe to NNMi NodeNotification service, follow these steps:

1. Deploy the sample war file into a running servlet container such as Tomcat or jboss.

Note
You can also deploy the war and then start up Tomcat or jboss.

2. Point a browser to the deployed servlet (supplying the appropriate host/port values) as follows:

   http://<host>:<port>/NnmNotificationClient/subscribe

   You should see subscription details to indicate that the bean subscribed successfully.
3. Open the NNMi console and update the lifecycleState of any Incident.
4. Examine the log file, `sdk.0.0.log`, found in the `data/log/nnm` folder to verify that an `IncidentChange` event is received by the client. The `IncidentChange` event message (in xml form) that was logged by the notification client should be near the bottom of the file.

Inventory Objects Attributes

For reference, each object has a UI column to show the attributes displayed in the NNMi console's tables and forms for that object. The Read column of these tables indicates the attributes returned in the WS-I `getXxx()` APIs and WS-Mgmt Get requests.

Incidents

Model	UI	Read	Incident Mgmt Events	Incident Notification
String id		✓		✓
String uuid		✓	<unique GUID>	✓
String sourceUuid		✓	✓	✓
String sourceName	Source Object	✓	✓	✓
String sourceType		✓	✓	
String sourceNodeUuid		✓	✓	✓
String sourceNodeName	Node Name	✓	✓	✓
String sourceNodeLongName	Node Long Name			✓
String name	Name	✓	*	✓
String severity	Severity	✓		✓
String priority	Priority	✓	<NONE>	✓
String lifecycleState	Lifecycle State	✓		trigger
String assignedTo	Assigned To	✓		
String category	Category	✓		✓
String family	Family	✓		✓
String origin	Origin	✓		✓
String nature	Correlation Nature	✓	<ROOTCAUSE>	✓
int duplicateCount	Duplicate Count	✓		✓
String formattedMessage Note: formattedMessage will always be returned in the locale of the NNMi server	Message	✓		✓
String notes	Notes	✓		
Boolean rcaActive	RCA Active	✓		trigger
Date originOccurrenceTime	Origin Occurrence Time	✓	<current time>	✓
Date firstOccurrenceTime	First Occurrence Time	✓		✓
Date lastOccurrenceTime	Last Occurrence Time	✓		✓
Date created	Created	✓		✓

Date modified	Last Modified	✓			
Cia[] cias	Custom Attributes	✓	✓	✓	
name	Name	✓	✓	✓	
type	Type	✓	✓	✓	
value	Value	✓	✓	✓	
Date updateTime				✓	
String previousLifecycleState				✓	
String previousRcaActive				✓	
int incidentResent				✓	

*Required

<default> (default value if not supplied)

trigger (attribute value change triggers a notification)

The sourceUuid and sourceName attributes contain the uuid/name values of the related object which is the source of the incident. The sourceType attribute indicates the type of object responsible for this incident.

Node

Model	UI	Read	Notification WS-E	Update
String id		✓	✓	
String uuid		✓	✓	
String name	Name	✓		
String status	Status	✓		
boolean isEndNode	End Node	✓		
boolean isSnmpSupported	SNMP Supported	✓		
String systemName	System Name	✓		
String systemContact	System Contact	✓		
String systemDescription	System Description	✓		
String systemLocation	System Location	✓		
String systemObjectId	System Object ID	✓		
String longName	Hostname	✓		
String managementMode	Node Management Mode	✓		✓
String discoveryState	Discovery State	✓		
String notes	Notes	✓		✓
Capabilities	Capabilities	✓		✓

CustomAttributes	Custom Attributes	✓	✓	✓
Conclusions	Conclusions	via getConclusions()		
Date created	Created	✓		
Date modified	Last Modified	✓		
String snmpVersion		✓		
DeviceProfile				
String deviceModel	Device Model	✓		
String deviceVendor		✓		
String deviceFamily		✓		
String deviceDescription		✓		
String deviceCategory	Device Category	✓		
String event			✓	
Date updateTime			✓	
Boolean discoveryAnalysisUpdated			✓	
int nodeResent			✓	

Interface

Model	UI	Read	Update
String id		✓	
String uuid		✓	
String name	Name	✓	
String status	Status	✓	
String hostedOnId	Hosted On Node	✓	
String connectionId	Layer 2 Connection	✓	
long ifIndex	IfIndex	✓	
String ifAlias	IfAlias	✓	
String ifDescr	IfDescription	✓	
String ifName	IfName	✓	
String ifType	IfType	✓	
long ifSpeed	IfSpeed	✓	
String physicalAddress	Physical Address	✓	
String managementMode	Direct Management Mode	✓	✓
String computedManagementMode	Management Mode		
String notes	Notes	✓	✓

Capabilities	Capabilities	✓	✓
CustomAttributes	Custom Attributes	✓	✓
Conclusions[]	Conclusions	via getConclusions()	
String administrativeState	Administrative State	✓	
String operationalState	Operational State	✓	
Date created	Created	✓	
Date modified	Last Modified	✓	

Note
The Node attributes isCDP, isLanSwitch and isIPv4Router have been removed. The capabilities com.hp.nnm.capability.node.lanswitching and com.hp.nnm.capability.node.ipforwarding have replaced isLanSwitch and isIPv4Router respectively.

IPAddress

Model	UI	Read	Update
String id		✓	
String uuid		✓	
String ipValue	Address	✓	
String hostedOnId	Hosted On Node	✓	
String ipSubnetId	In Subnet	✓	
String inInterfaceId	In Interface	✓	
int prefixLength	Prefix Length	✓	
String managementMode	Direct Management Mode	✓	✓
String computedManagementMode	Management Mode	✓	
Conclusions[]	Conclusions	via getConclusions()	
String notes	Notes	✓	✓
Date created	Created	✓	
Date modified	Last Modified	✓	

IPSubnet

Model	UI	Read	Update
String id		✓	
String uuid		✓	
String name	Name	✓	
int prefixLength	Prefix Length	✓	
String prefix	Prefix	✓	

String notes	Notes	✓	✓
Date created	Created	✓	
Date modified	Last Modified	✓	

L2Connection

Model	UI	Read	Update
String id		✓	
String uuid		✓	
String name	Name	✓	
String notes	Notes	✓	✓
String status	Status	✓	
String[] interfaces	Interfaces	✓	
Date created	Created	✓	
Date modified	Last Modified	✓	

VLAN

Model	UI	Read	Update
String id		✓	
String uuid		✓	
String name	Name	✓	
String vlanId	VLAN Id	✓	

Card

Model	UI	Read	Update
String id		✓	
String uuid		✓	
String name	Name	✓	
String status	Status	✓	
String index		✓	
Long	entityPhysicalIndex	✓	
String cardDescr		✓	
String hostedOnId		✓	
String type (‘Card’,’Chassis’)		✓	
String modelName		✓	
String serialNumber		✓	

String firmwareVersion	✓	
String hardwareVersion	✓	
String softwareVersion	✓	
String hostingCard	✓	
String redundantGroup	✓	
ManagementMode managementMode	✓	✓
Date created	✓	
Date modified	✓	
Capability[] capabilities	✓	✓

Port

Model	UI	Read	Update
String id		✓	
String uuid		✓	
String name	Name	✓	
Integer index		✓	
String hostedOnId		✓	
String iface		✓	
String card		✓	
String type		✓	
String speed		✓	
DuplexSetting duplexSetting		✓	
Status status		✓	
ManagementMode managementMode		✓	✓
Date created		✓	
Date modified		✓	
Capability[] capabilities		✓	✓

Enumerations and Reserved Values

Status (in order of least to highest severity)

- “NORMAL”
- “WARNING”
- “MINOR”
- “MAJOR”
- “CRITICAL”
- “DISABLED”

- “NOSTATUS”
- “UNKNOWN”

Management Mode
- “INHERITED”
- “MANAGED”
- “NOTMANAGED”
- “OUTOFSERVICE”

Nature
- “INHERITED”

Origin
- “MANAGEMENTSOFTWARE”
- “MANUALLYCREATED”
- “REMOTELYGENERATED”
- “SNMPTRAP”
- “SYSLOG”
- “OTHER”

Severity
- “NORMAL”
- “WARNING”
- “MINOR”
- “MAJOR”
- “CRITICAL”

Reserved Node Conclusions
- CustomPollingOnNodeMajor
- CustomPollingOnNodeMinor
- CustomPollingOnNodeWarning
- CustomPollingOnNodeNormal
- NodeWithBadFan
- NodeWithGoodFans
- NodeWithBadPowerSupply
- NodeWithGoodPowerSupplies
- NodeDown
- NodeUp
- WANEdgeRouterUnresponsive
- NodeUnmanagable
- NonSNMPNodeUnresponsive
- NodeOrConnectionDown
- AllUnresponsiveAddressesInNode
- SomeUnresponsiveAddressesInNode
- SomeResponsiveAddressesInNode
- AllResponsiveAddressesInNode
- InterfacesDownInNode

- InterfacesUpInNode
- CrgMalfunctionInNode
- CrgNormalInNode
- CardsDownInNode
- CardsUpInNode
- CardsUndeterminedStateInNode
- UnresponsiveAgentInNode
- ResponsiveAgentInNode
- SomeInterfacesOutsideThresholdBoundariesInNode
- AllInterfacesWithinThresholdBoundariesInNode

Codes

Some attributes are expressed as coded attributes that accept a restricted set of values. While enumerated attributes and their valid values (such as ManagementMode) are described by the service WSDL, these code values are not. Below is a list of code attributes and their current valid values. Additional values might be permitted in the future.

Incident.LifecycleState

- “com.hp.nms.incident.lifecycle.Dampened”
- “com.hp.nms.incident.lifecycle.Registered”
- “com.hp.nms.incident.lifecycle.InProgress”
- “com.hp.nms.incident.lifecycle.Completed”
- “com.hp.nms.incident.lifecycle.Closed”

Incident.Priority

- “com.hp.nms.incident.priority.Low”
- “com.hp.nms.incident.priority.Medium”
- “com.hp.nms.incident.priority.High”
- “com.hp.nms.incident.priority.Top”
- “com.hp.nms.incident.priority.None”

Incident.sourceType

- "com.hp.ov.nms.model.core.Node"
- "com.hp.ov.nms.model.core.Interface"
- "com.hp.ov.nms.model.layer3.IPAddress"
- "com.hp.ov.nms.model.layer3.IPSubnet"
- "com.hp.ov.nms.model.layer2.L2Connection"

IPAddress replaces IPv4Address and IPSubnet replaces IPv4Subnet. See Deprecated Services (7.2) for details.

Incident.Category

- "com.hp.nms.incident.category.Fault"
- "com.hp.nms.incident.category.Status"
- "com.hp.nms.incident.category.Config"
- "com.hp.nms.incident.category.Accounting"
- "com.hp.nms.incident.category.Performance"
- "com.hp.nms.incident.category.Security"
- "com.hp.nms.incident.category.Alert"

Incident.Family

- "com.hp.nms.incident.family.Address"

- "com.hp.nms.incident.family.Interface"
- "com.hp.nms.incident.family.Node"
- "com.hp.nms.incident.family.OSPF"
- "com.hp.nms.incident.family.HSRP"
- "com.hp.nms.incident.family.AggregatePort"
- "com.hp.nms.incident.family.Board"
- "com.hp.nms.incident.family.Connection"
- "com.hp.nms.incident.family.Correlation"

API Quick Reference

For each object type there are both WS-Mgmt (Enumerate, Get, and Put) APIs as well as overlapping WS-I capabilities. The URLs listed below assume NNM is installed on localhost port 80.

Incident

http://localhost/IncidentBeanService/IncidentBean?wsdl

- Incident[] getIncidents(Filter filter)
- Integer getIncidentCount(Filter filter)
- Incident[] getChildIncidents(String id)
- addIncident(IncidentMgmtEvent)
- deleteIncident(String id)
- deleteIncidentByUuid(String uuid)
- updateNotes(String id, String notes)
- updateLifecycleState(String id, String lifecycleState)
- updatePriority(String id, String priority)
- updateCias(String id, Cia[])
- IncidentCorrelation[] getChildCorrelations(String id)
- String[] getAssignToPrincipals()
- void updateAssignedTo(String id,String assignTo)

IncidentConfiguration

http://localhost/IncidentConfigurationBeanService/IncidentConfigurationBean?wsdl

- RemoteNMEventConfig[] getRemoteEventConfig(Filter filter)
- ManagementEventConfig[] getManagementEventConfig(Filter filter)
- SnmpTrapConfig[] getSnmpTrapConfig(Filter filter)
- PairwiseConfig[] getPairwiseConfig(Filter filter)

Node

http://localhost/NodeBeanService/NodeBean?wsdl

- Node[] getNodes(Filter filter)
- Integer getNodeCount(Filter filter)
- NodeConclusion[] getConclusions(String id)
- addConclusion(String id, String incidentId, Status status, String conclusion, String[] cancelledConclusions)
- updateManagementMode(String id, String managementMode)
- updateNotes(String id, String notes)
- addSeeds(String[] hostnamesOrIPs)
- addSeedsForTenant(String[] hostnamesOrIPs, String tenantName, String notes)
- removeSeeds(String[] hostnamesOrIPs)
- addHints(String[] hostnamesOrIPs)
- rediscoverHosts(String[] hostnamesOrIPs)
- pollHosts(String[] hostnamesOrIPs)
- addCapabilities(String id, Capability[] capabilities)

- String[] addCapabilitiesBulk(String[] ids, Capability[] capabilities)
- removeCapabilities(String id, String[] capabilityKeys)
- String[] removeCapabilitiesBulk(String[] ids, String[] capabilityKeys)
- updateCustomAttributes(String id, CustomAttribute[] customAttributes)
- addCustomAttributes(String id, CustomAttribute[] customAttributes)
- String[] addCustomAttributesBulk(String[] ids, CustomAttribute[] customAttributes)
- removeCustomAttributes(String id,String[] customAttributes)
- String[] removeCustomAttributesBulk(String[] ids, String[] customAttributes)
- boolean deleteNode(String id)
- boolean deleteNodeByUuid(String uuid)
- getNnmSystemName(final String id)
- boolean isLocal(final String id)
- boolean isForwardable(final String id)
- String[] getForwardableNodes(final String[] ids)

Interface

http://localhost/InterfaceBeanService/InterfaceBean?wsdl

- Interface[] getInterfaces(Filter filter)
- Integer getInterfaceCount(Filter filter)
- InterfaceConclusion[] getConclusions(String id)
- updateManagementMode(String id, String managementMode)
- updateNotes(String id, String notes)
- addCapabilities(String id, Capability[] capabilities)
- String[] addCapabilitiesBulk(String[] ids,Capability[] capabilities)
- removeCapabilities(String id, String[] capabilityKeys)
- String[] removeCapabilitiesBulk(String[] ids,String[] capabilityKeys)
- updateCustomAttributes(String id, CustomAttribute[] customAttributes)
- addCustomAttributes(String id, CustomAttribute[] customAttributes)
- String[] addCustomAttributesBulk(String[] ids,CustomAttribute[] customAttributes)
- Integer addBulkCustomAttributes(String caName, String[] ids, String[] caValues)
- removeCustomAttributes(String id,String[] customAttributes)
- String[] removeCustomAttributesBulk(String[] ifaceIds,String[] customAttributes)
- Long[] getInputSpeedBulk(String[] ifaceIds)
- Long[] getOutputSpeedBulk(String[] ifaceIds)

IPAddress

http://localhost/IPAddressBeanService/IPAddressBean?wsdl

- IPAddress[] getIPAddresses(Filter filter)
- IPAddressConclusion[] getConclusions(String id)
- updateManagementMode(String id, String managementMode)
- updateNotes(String id, String notes)
- addCapabilities(String id, Capability[] capabilities)

- String[] getTenantsByUuids(String[] uuids)
- String[] getMappedAddressesByUuids(String[] uuids)

IPSubnet

http://localhost/IPSubnetBeanService/IPSubnetBean?wsdl

- IPSubnet[] getIPSubnets(Filter filter)
- updateNotes(String id, String notes)
- String[] getTenantsByUuids(String[] uuids)

L2Connection

http://localhost/L2ConnectionBeanService/L2ConnectionBean?wsdl

- L2Connection[] getL2Connections(Filter filter)
- updateNotes(String id, String notes)
- String getSource(String id)

Card

http://localhost/NmsSdkService/CardBean?wsdl

- Card[] getCards(String filter)
- updateManagementMode(String id, ManagementMode managementMode)
- addCapabilities(String id, Capability[] capabilities)
- removeCapabilities(String id,String[] capabilities)
- IncidentConclusion[] getConclusions(String id)

Port

http://localhost/NmsSdkService/PortBean?wsdl

- Port[] getPorts(String filter)
- addCapabilities(String id, Capability[] capabilities)
- removeCapabilities(String id, String[] capabilities)

Topology

http://localhost/TopologyBeanService/TopologyBean?wsdl

- NetworkPathResponse getPath(String startHostnameOrIP, endHostnameOrIP)

VLAN

http://localhost/VLANBeanService/VLANBean?wsdl

- VLAN[] getVLANs(Filter filter)
- Interface[] getInterfacesForVLAN(String vlanId)
- VLAN[] getVLANsForInterface(String interfaceId)
- Port[] getPortsForVLAN(String vlanId)
- Port[] getPortsForVLANbyId(String id)
- VLAN[] getVLANsForDevice(String deviceId)

Snmp

http://localhost/SnmpBeanService/SnmpBean?wsdl

- (deprecated) SnmpConfiguration[] getNodeConfiguration(String[] hostnames)
- (deprecated) setNodeConfiguration(String[] hostnames[], String[] readCommunityStrings, String writeCommunityString, String managementAddress, Integer snmpPort, Integer retries, Integer timeout)

- SnmpConfigResponse[] getNodeConfigByIds(String[] nodeIds)
- SnmpConfigResponse[] getNodeConfig (String[] hostnames)
- setNodeConfig(String[] hostnames[], SnmpConfigOverrides config)
- SnmpResponse[] snmpGet(String[] reqOids, String[] hostnames, SnmpConfigOverrides config)
- SnmpResponse[] snmpGetNext(String[] reqOids, String[] hostnames, SnmpConfigOverrides config)
- SnmpResponse[] snmpGetBulk(String[] reqOids, String[] hostnames, SnmpConfigOverrides config, Integer nonRepeaters, Integer maxRepetitions)
- SnmpResponse[] snmpSet(String setOidStr, String value, Asn1Constant asnType, String[] hostnames, SnmpConfigOverrides config)
- String getOidForName(String name);
- String getNameForOid(String oid);
- String[] listDefaultReadCommunityStrings();
- addDefaultReadCommunityString(String commString);
- removeDefaultReadCommunityString(String commString);

NodeGroup

http://localhost/NodeGroupBeanService/NodeGroupBean?wsdl

- Node[] getNodeGroups(Filter filter)
- NodeConclusion[] getConclusions(String id)
- String[] getMemberIds(String id)
- NodeGroup[] getNodeGroupsByNode(final String nodeId)

ExtProperties

http://localhost/ExtPropertiesBeanService/ExtPropertiesBean?wsdl

- setProperty(String domain, String key, String label, String type, String value)
- removeProperty(String domain, String key)
- String getPropertyValue(String domain, String key)
- CustomAttribute[] getProperties(String domain)

ExtPropertiesRead

http://localhost/ExtPropertiesReadBeanService/ExtPropertiesReadBean?wsdl

- String readPropertyValue(String domain, String key)

Mib

http://localhost/MibBeanService/MibBean?wsdl

- MibVariable[] getMibs(String filter)

FrameRelayEndpoint

http://localhost/FrameRelayEndpointBeanService/FrameRelayEndpointBean?wsdl

- FrameRelayEndpoint[] getFrameRelayEndpoints(String filter)

Registry

http://localhost/NmsSdkService/WsRegistryBean?wsdl

- boolean register(final String serviceName, final URL endPoint)
- boolean unregister(final String serviceName)
- URL getEndpoint(final String serviceName)

- String[] listServiceNames()
- URL[] listEndpoints()

Security

http://localhost/NmsSdkService/SecurityServiceBean?wsdl

- SecurityGroupDTO[] listSecurityGroups()
- TenantDTO[] listTenants()
- NodeSecurityDTO[] getNodeSecurityDataByUuid(nodeUuids)
- NodeSecurityDTO[] getNodeSecurityDataByName(nodeNames)
- UserGroupMembershipDTO[] getUserGroupMembership(userNames)
- SecurityGroupMembershipDTO[] getSecurityGroupMembershipByUserGroupName(userGroupNames)
- SecurityGroupMembershipDTO[] getSecurityGroupMembershipBySecurityGroupUuid(securityGroupUuids)
- TenantOAM[] getTenantOAMs(Filter filter)
- String createTenant(TenantDTO tenant)

References

- jboss:

  http://www.jboss.com

- jbossWS 1.2.1 User's Guide:

  http://jbws.dyndns.org/mediawiki/index.php/JAX-WS_User_Guide

- Apache Axis:

  http://ws.apache.org/axis/

- Sun's Java Web Service Developer Pack:

  http://java.sun.com/webservices/downloads/previous/index.jsp

- Maven 3.0:

  http://maven.apache.org/

We appreciate your feedback!

If an email client is configured on this system, by default an email window opens when you click here.

If no email client is available, copy the information below to a new message in a web mail client, and then send this message to **network-management-doc-feedback@hpe.com**.

Product name and version: NNMi 10.20

Document title: Developer's Toolkit Guide

Feedback:

Hewlett Packard
Enterprise