By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on April 2, 2007.

Copyright Notice

Copyright (C) The Internet Society (2006).
Abstract

This memo contains a mechanism by which NETCONF server and client can extended to operate in a master-agent sub-agent scheme. It extends the base NETCONF protocol with additional NETCONF operations, describes the protocol for this interaction and provides error messages exchanged during this interaction.
1. Introduction

This draft document defines a mechanism by which NETCONF server and client can be extended to operate in a master-agent - sub-agent scheme. It defines both the NETCONF operations and the protocol for this interaction. The operations are defined for registration and deregistration of sub-agents for parts of the configuration data, delegation of rpc requests from master-agent to sub-agents. The protocol defines the communication mechanism between the master-agent and the sub-agent.

1.1. Definition of terms

Terms server and agent have identical meaning in the document and are used interchangeably.

Master-agent: A NETCONF agent representing the sub-agents towards the external interface and performing the roles of registrar, dispatcher and consolidator for requests arriving from the external interface.

Sub-agent: A NETCONF agent responsible for one or more entries of the configuration.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

1.2. Overview of master-agent sub-agent interaction

NETCONF uses a simple RPC based server - client communication model for performing various operations like retrieval, modification, and deletion of configuration data. In many device architectures like that of routers, base stations, etc., the device consists of a controller unit and a set of field replaceable units, all the field replaceable units are managed by central controller unit. The configuration data of such device is a collection of configuration blocks from various field replaceable units with in the device. In such cases, master-agent running on the central module interacts with sub-agents running on the modules, to operate on parts of the configuration data for which they are responsible for.

1.3. Capabilities Exchange

1.3.1. NETCONF master-agent

Master-agent is the NETCONF server. It interfaces with one or more sub-agents. Master-agent can handle rpc-requests in itself or delegate them to the appropriate sub-agents. Master-agent sends the
sessionid parameter to the sub agents as part of rpc-request.

A master-agent indicates itself by sending a master-agent capability in the form of a URI in the hello message:

urn:ietf:params:netconf:capability:masteragent:1.0

1.3.2. NETCONF sub-agent

Sub-agent is a NETCONF client. On startup it declares itself by sending a hello message with the sub-agent URI to the server. It receives the corresponding hello message from the master-agent. Sub-agent receives sessionid parameter in the rpc-request and responds back with the same sessionid in the rpc-response.

A sub-agent indicates itself by sending a sub-agent capability in the form of a URI in the hello message:

urn:ietf:params:netconf:capability:subagent:1.0

1.4. Protocol Definition

When a NETCONF peer starts up, it declares its capability as a master-agent or sub-agent or none by using the hello message. NETCONF sub-agent uses a rpc with register-data operation to register with a master-agent for parts of the configuration data. Master-agent acknowledges the registration with a rpc-reply message. When a rpc-request arrives from an external NETCONF based manager, master-agent breaks up the request as per the data model registrations and passes them to the sub agents as smaller rpc-request messages. Sub-agents process the request and send back rpc-reply or rpc-error message to the master-agent. Master-agent collates the rpc-reply messages into a single message and passes it back to the client. The order of collation of rpc-reply SHOULD be same as in the rpc-request sequence.

Scenario 1: Single rpc request is to be executed on multiple sub-agents
Master-agent receives the following request:

```xml
<rpc messageid="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get>
        <filter type="subtree">
            <users/>
            <interfaces/>
        </filter>
    </get>
</rpc>
```

Considering the two different sub-agents are registered for processing interfaces and users. Master-agent converts the request into two rpc-requests and passes them to the sub-agents. Note that master-agent adds a sequenceid attribute to the rpc.

RPC Request to sub-agent1:

```xml
<rpc messageid="101" sequenceid="1"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get>
        <filter type="subtree">
            <users/>
        </filter>
    </get>
</rpc>
```

RPC Request to sub-agent2:

```xml
<rpc messageid="101" sequenceid="2"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get>
        <filter type="subtree">
            <interfaces/>
        </filter>
    </get>
</rpc>
```
Response obtained from sub-agent1:

```xml
<rpc-reply messageid="101" sequenceid="1"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <users>
      <user>
        <name>root</name>
        <type>superuser</type>
        <company-info>
          <dept>IMG</dept>
          <ID>25224</ID>
        </company-info>
      </user>
    </users>
  </data>
</rpc-reply>
```

Response from sub-agent2:

```xml
<rpc-reply message-id="101" sequenceid="1012"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <interfaces>
      <interface ifName="eth0">
        <ifInOctets>45621</ifInOctets>
        <ifOutOctets>774344</ifOutOctets>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```
Response returned by master-agent:

```xml
<rpc-reply messageid="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <users>
      <user>
        <name>root</name>
        <type>superuser</type>
        <company-info>
          <dept>IMG</dept>
          <ID>25224</ID>
        </company-info>
      </user>
    </users>
    <interfaces>
      <interface ifName="eth0">
        <ifInOctets>45621</ifInOctets>
        <ifOutOctets>774344</ifOutOctets>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```

Scenario 2: Single rpc request is to be executed on a single sub-agent

The master-agent simply forwards the request to the sub-agent without adding the sequenceid tag

Scenario 3: Multiple rpc requests are to be executed on multiple sub-agents

Request Processing:

Master-agent receives rpc-request1 with messageid 101:

Master-agent forwards the rpc-request1 to two sub-agents
* sub-agent1 receives messageid 101, sequenceid 1
* sub-agent2 receives messageid 101, sequenceid 2
While rpc-request2 is being processed in the sub-agents, master-agent receives another request with messageid as 102:

Master-agent forwards the rpc-request2 to two sub-agents
* sub-agent1 receives messageid 102, sequenceid 1
* sub-agent2 receives messageid 102, sequenceid 2

Response Processing:
Master agent receives the responses for rpc-request1 from sub-agent1 and sub-agent2 with messageid 101 and sequenceids 1 and 2 respectively, Collates them into a rpc-response1 and returns it.

Master agent receives the responses for rpc-request2 from sub-agent1 and sub-agent2 with messageid 102 and sequenceids 1 and 2 respectively, Collates them into a rpc-response2 and returns it.

Scenario 4: Notification from a sub-agent
Notification message is sent to the master-agent with out any sequenceid attribute.

1.5. Modification to rpc, rpc-reply and rpc-error messages
An additional sequenceid attribute is introduced in the rpc, rpc-reply and rpc-error messages. sequenceid attribute is introduced by the master-agent when it forwards the requests to multiple sub-agents and is used by the master-agent to provide sequencing of rpc-reply messages in the same order of the rpc-requests. Any tags following the messageid attribute are to be passed back in the rpc-reply / rpc-error message in accordance with the NETCONF protocol specification. When the rpc-reply messages arrive from the sub-agents they carry the same sequenceid as was in the rpc message. The master-agent collates all the messages by their sequenceid values in the increasing order. However, use of this attribute is OPTIONAL in case when the whole request is passed to a single sub-agent. The sequenceid attribute value is a positive integer. The combination of messageid and sequenceid makes a unique id for processing the messages at the sub-agent.

<rpc messageid="101" sequenceid="1 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
1.6. New Protocol Operations

1.6.1. register-data operation

Description:

Register with master-agent the responsibility to perform operations on the specified data tree element. Sub-agent sends a register-data operation in a rpc-request to the master-agent. The request MUST contain a mandatory parameter filter. When this request arrives at the master-agent, it associates the sessionid on which the request arrived with the associated data tree elements for which sub-agent is responsible.

Parameters:

filter: Filter element is used to specify a filter by applying which the master-agent can select a unique tree node which could be either a leaf node or a tree node. If a tree node is chosen on the filter operation, the entire child nodes of the selected tree node are considered to be of the domain sub-agent. Their SHOULD NOT be more than one sub-agents associated to a data node. A rpc-error message is returned when a register-data operation arrives for which already a sub-agent has registered itself.

There are two ways to specify the filter:

(a) sub tree filter: specifying the subtree filter is as per the NETCONF protocol specification.

Example:

```xml
<rpc messageid="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <register-data>
    <filter type="subtree">
      <top xmlns="http://example.com/schema/1.2/config">
        <users>
          <user/>
          <user/>
        </users>
      </top>
    </filter>
  </register-data>
</rpc>
```
In the above example, top/users/user node is associated with the sub-agent which sent the rpc request.

(b) xpath filter: specifying xpath filter is as per the NETCONF protocol specification.

Example:

```xml
<rpc messageid="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <register-data>
    <filter type="xpath"> top/users/user</filter>
  </register-data>
</rpc>
```

Positive Response:

If the master-agent succeeds with the registration, it returns a rpc-reply with the ok element.

Negative Response:

If the master-agent fails with the registration, it returns a rpc-error with the appropriate error information. If the failure is because of the the registration already held by another sub-agent, then a rpc-error is returned with the error-tag as registration-denied. See appendix A.1.

Example:
<rpc messageid="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <register-data>
    <filter type="subtree">
      <top xmlns="http://example.com/schema/1.2/config">
        <users>
          <user/>
        </users>
      </top>
    </filter>
  </register-data>
</rpc>
<rpc-reply messageid="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>

1.6.2. deregister-data message

Description:

deregister-data operation is used by sub-agent and sent to the master-agent indicating that it wants to deregister from performing operations on the specified node or set of nodes. filter parameter is used to provide the list of nodes for which sub-agent would like to deregister.

Parameters:

filter:

Specifies the filter which selects the data nodes for which the sub-agent would like to deregister. Use of filter parameter is OPTIONAL. In the absence of filter parameter all the nodes associated with the sub-agent are released by the master-agent.

Positive Response:

If the master-agent succeeds with the deregistration, it returns a rpc-reply with the ok element.

Negative Response:

If a deregister-data operation arrives with specific data node(s) for which the sub-agent which sent the message is not registered to, then a rpc-error is returned back with error-tag as invalid-deregistration-request. See appendix A.2.

Example:
<rpc messageid="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <deregister-data>
        <filter type="subtree">
            <top xmlns="http://example.com/schema/1.2/config">
                <users>
                    <user/>
                </users>
            </top>
        </filter>
    </deregister-data>
</rpc>
<rpc-reply messageid="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ok/>
</rpc-reply>

1.7. Security Considerations

None.
2. References

2.1. Normative References


2.2. Informative References


Appendix A. Extensions to NETCONF Error List

A.1. Registration denied

Tag: registration-denied

Error-type: protocol

Severity: error

Error-info:

tagname : tag name for which the registration is denied

sessionid : sessionid of session holding the registration

Description: registration-denied error is returned when the data
nodes are already registered with another sub-agent.

A.2. Invalid deregistration request

Tag: invalid-deregistration-request

Error-type: protocol

Severity: error

Error-info:

tagname: tag name for which the deregistration failed

sessionid: sessionid of session holding the registration

Description: invalid-deregistration-request error is sent back when
the session id of the agent requesting deregistration does not match
the sessionid of the registered agent.
Author’s Address

Jayaprakash G. Kulkarni
Broadband Center of Excellence,
Wipro Technologies,
Doddakanelli, Sarjapur Road,
Bangalore 560025
India.

Email: jayaprakash.kulkarni@wipro.com