OpenDaylight Controller:Config:Examples:Netconf

This page discusses "Southbound" NETCONF ... be aware that the controller itself has an internal NETCONF server for "Northbound" uses.

In the Hydrogen release, the Netconf Client (NCC) must be configured with the address and login credentials (username/password) for each target Netconf Server to which it connects. An NCC instance must be configured for a each target Server.

The yang model for NCC configuration can be found at {{{3}}}. The per target Server configuration includes the target Server's IP address, port, connection type (TCP or SSH), and username/password. The NCC configuration also includes a reference to MD-SAL and the configuration of NCC's thread group for opening and maintaining sockets and its event executor for sourcing events into MD-SAL. Defaults values for global configuration, defined in this page, should be used in most cases.

In text below phrases like NCC, mount-point, netconf-client are used interchangeably.

Contents

Using netopeer netconf server instead of a real device

Netopeer is a open source server that can be used to explore mdsal and restconf.

Docker image

The easiest way, though not supported, is to install docker on a Linux machine and then pull netopeer image:

```
docker run --rm -t -p 1831:830 dockeruser/netopeer
```

Then confirm that you can get a hello message use:

```
ssh root@localhost -p 1831 -s netconf
(password root)
```

You should see server immediately send an xml followed by netconf EOM.

Manual installation

OpenDaylight_Controller:Config:Examples:Netconf:Manual_netopeer_installation

Using ODL's netconf server as mountpoint

It should be possible to connect md-sal to local netconf server that is running on port 8383. It is configured out of box in file /etc/opendaylight/karaf/99-netconf-connector.xml .

Spawning netconf connectors via topology configuration

Netconf connectors are configured through the config datastore.

One of the odl-netconf-topology or odl-netconf-clustered-topology features needs to be installed. To configure or update a netconf-connector via topology you need to send following request to Restconf:
Method: PUT
URI: http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/new-netconf-device
Headers:
Accept: application/xml
Content-Type: application/xml

Payload:

```
<node xmlns="urn:TBD:params:xml:ns:yang:network-topology">
  <node-id>new-netconf-device</node-id>
  <host xmlns="urn:opendaylight:netconf-node-topology">127.0.0.1</host>
  <port xmlns="urn:opendaylight:netconf-node-topology">17830</port>
  <username xmlns="urn:opendaylight:netconf-node-topology">admin</username>
  <password xmlns="urn:opendaylight:netconf-node-topology">admin</password>
  <tcp-only xmlns="urn:opendaylight:netconf-node-topology">false</tcp-only>
  <keepalive-delay xmlns="urn:opendaylight:netconf-node-topology">0</keepalive-delay>
</node>
```

To delete a netconf connector issue a DELETE request to the following url:
URI: http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf/node/new-netconf-device

Connecting to a device not supporting netconf monitoring

The netconf connector in ODL relies on ietf-netconf-monitoring support when connecting to remote netconf device. The ietf-netconf-monitoring support allows netconf connector to list and download all yang schemas that are used by the device. Netconf connector can only communicate with a device if it knows the set of used schemas (or at least a subset). However some devices use yang models internally but do not support netconf monitoring. Netconf connector can also communicate with these devices, but you have to side load the necessary yang models into ODL's yang model cache for netconf connector. In general there are 2 situations you might encounter:

1. Netconf device does not support ietf-netconf-monitoring but it does list all its yang models as capabilities in hello message

This could be a device that internally uses only only ietf-inet-types yang model with revision 2010-09-24. In the hello message that is sent from this device there is this capability reported:

```
urn:ietf:params:xml:ns:yang:ietf-inet-types?module=ietf-inet-types&revision=2010-09-24
```

For such devices you only need to put the schema into folder **cache/schema** inside your karaf distribution.
Important note: The file with yang schema for ietf-inet-types has to be called ietf-inet-types@2010-09-24.yang. Its the required naming format of the cache.

2. Netconf device does not support ietf-netconf-monitoring and it does NOT list its yang models as capabilities in hello message

Compared to device that lists its yang models in hello message, in this case there would be no capability with ietf-inet-types in the hello message. This type of device basically provides no information about the yang schemas it uses so its up to the user of ODL to properly configure netconf connector for this device. Netconf connector has an optional configuration element called **yang-module-capabilities** that contains a list of "yang module based" capabilities and an optional flag indicating whether to override or merge this list of capabilities with the capabilities from device. So by setting this configuration element, it is possible to **override** or augment the "yang-module-based" capabilities reported in hello message of the device. To do this, we need to modify the configuration of netconf connector by adding this element (It needs to be added next to the host, port, etc. config elements):

```
<yang-module-capabilities>
  <capability>
      urn:ietf:params:xml:ns:yang:ietf-inet-types?module=ietf-inet-types&amp;revision=2010-09-24
  </capability>
</yang-module-capabilities>
```

By default the list of capabilities is merged with those of the device. If you want to completely override the device's capabilities, specify the **override** elemen t as true:

```
<override>true</override>
```

Remember to also put the yang schemas into the cache folder.
Note: For putting multiple capabilities, you just need to replicate the capability xml element inside yang-module-capability element. Capability element is modeled as a leaf-list.
With this configuration, we would make the remote device report usage of ietf-inet-types in the eyes of netconf connector.

Aggregation of a separate SharedSchemaCache, SchemaContextFactory and FilesystemSchemaSourceCache

There are certain instances in which you should aggregate a separate Schema cache, independent of the one that other netconf mounts use. For example, in the case of revisionless import, there may be cause to import a specific revision for one netconf mount, and not another. In order to achieve this, when you mount the netconf device, add the following at the same level as the "host" and "port" leaves:

```
  <schema-cache-directory>your_schema_cache_directory</schema-cache-directory>
```

your_schema_cache_directory is a folder relative to $KARAF_HOME/cache. In the case above, the folder used to store yang downloaded from devices is:

```
$KARAF_HOME/cache/your_schema_cache_directory/
```

You can now safely edit yang files in the above folder without side-effecting other devices. In order for the edits to take effect, you **must** restart the controller. You can reuse the schema cache directories (i.e, set up one cache directory for all devices of a certain type).

A postman collection to test out this functionality can be found here [1]

Testing mountpoint via restconf

Issuing

```
curl -v http://localhost:8080/restconf/operational/network-topology:network-topology/topology
/topology-netconf/node/libnetconfd/
```

should output

```
{
    "node": [
        {
            "id": "libnetconfd",
            "netconf-node-inventory:initial-capability": [
                "(http://netconfcentral.org/ns/toaster?revision=2009-11-20)toaster",
                "(urn:ietf:params:xml:ns:yang:ietf-netconf-acm?revision=2012-02-22)ietf-netconf-
acm",
                "(urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults?revision=2010-06-09)ietf-
netconf-with-defaults",
                "(urn:ietf:params:xml:ns:netconf:notification:1.0?revision=2008-07-14)
notifications",
                "(urn:ietf:params:xml:ns:netmod:notification?revision=2008-07-14)nc-
notifications",
                "(urn:ietf:params:xml:ns:yang:ietf-netconf-notifications?revision=2011-08-07)ietf-
netconf-notifications",
                "(urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring?revision=2010-10-04)ietf-
netconf-monitoring",
                "(urn:ietf:params:xml:ns:netconf:base:1.0?revision=2011-03-08)ietf-netconf",
                "(urn:ietf:params:xml:ns:yang:ietf-yang-types?revision=2010-09-24)ietf-yang-
types",
                "(urn:ietf:params:xml:ns:yang:ietf-inet-types?revision=2010-09-24)ietf-inet-types"
            ],
            "netconf-node-inventory:connected": true
        }
    ]
}
```

However if the remote node is not connected, output will look like:

```
{
    "node": [
        {
            "id": "libnetconfd"
        }
    ]
}
```

Getting actual configuration data from server:

```
curl -v http://localhost:8080/restconf/config/network-topology:network-topology/topology/topology-
netconf/node/libnetconfd/yang-ext:mount/
```

should output

```
   .
   .
   .
{
    "data": {}
}
```

(in case of netopeer docker image result is empty)

Getting actual operational data from server:

```
curl -v http://localhost:8080/restconf/operational/network-topology:network-topology/topology
/topology-netconf/node/libnetconfd/yang-ext:mount/
```

should output

```
      .
      .
      .
{
    "data": {
        "toaster": {
            "toasterManufacturer": "CESNET, z.s.p.o.",
            "toasterModelNumber": "toaster",
            "toasterStatus": "up"
        },
        "netconf-state": {
            "datastores": {
                "datastore": [
                    {
                        "name": "running"
                    },
                    {
                        "name": "startup"
                    },
                    {
                        "name": "candidate"
                    }
                ]
            },
            "sessions": {
                "session": [
                    {
                        "session-id": 2948,
                        "transport": "netconf-ssh",
                        "username": "jameshall",
                        "source-host": "192.168.1.29",
                        "login-time": "2014-05-02T17:25:06Z",
      .
      .
      .
                        "out-rpc-errors": 6,
                        "out-notifications": 0
                    }
        },
        "netconf": {
            "streams": {
                "stream": [
                    {
                        "name": "NETCONF",
                        "description": "NETCONF Base Notifications",
                        "replaySupport": true,
                        "replayLogCreationTime": "2014-04-22T12:49:00Z"
                    }
                ]
            }
        },
        "nacm": {
            "denied-operations": 0,
            "denied-data-writes": 0,
            "denied-notifications": 0
        }
    }
}
```

Remote RPCs

The pattern is very similar to the gets being performed above.

`http://localhost:8080/restconf/operations/<mountPoint>/yang-ext:mount/<operation>`

Now, since the netconf server adds netconf devices to the network-topology:network-topology url, you would likely be following this pattern to make the RPC call to the remote server:

```
http://localhost:8080/restconf/operations/network-topology:network-topology/topology/topology-netconf/node/<nodeId
>/yang-ext:mount/<operation>
```

For example, to make-toast on the node with id **netopeer** server, you would POST to the following URL:

```
http://localhost:8080/restconf/operations/network-topology:network-topology/topology/topology-netconf/node
/netopeer/yang-ext:mount/toaster:make-toast
```

Note: In the case of the netopeer server, you get an immediate response (204). However, if you immediately reissue the command again you will see that we get an RPC error because the "make-toast" action is still running! Further proof that the request is going out to the remote server.

Remote RPC with loopback netconf mount

The default configuration of ODL controller contains file 99-netconf-connector. This file mounts the netconf server present in ODL (0.0.0.0:1830) and makes it accessible via RESTCONF under mount id: controller-config. To verify that the mount was successful, issue GET request to following URL:

> *http://localhost:8181/restconf/config/network-topology:network-topology/topology/topology-netconf*
> */node/controller-config/yang-ext:mount/config:modules*

You should see the current configuration of the ODL controller in XML.

Now you can invoke rpcs on the netconf server using RESTCONF. In this example we will invoke get-schema rpc from ietf-netconf-monitoring. To do this, issue a POST request to:

> *http://localhost:8181/restconf/operations/network-topology:network-topology/topology/topology-*
> *netconf/node/controller-config/yang-ext:mount/ietf-netconf-monitoring:get-schema*

with:

```
Accept application/xml
Content-Type application/xml
```

and payload:

```
<input xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
  <identifier>toaster</identifier>
</input>
```

This will invoke the get-schema rpc and request the yang schema for toaster module present in the controller. The rpc will be executed immediately and the output should contain the schema of toaster module.

Get-schema rpc can be invoked on any netconf device supporting ietf-netconf-monitoring. The identifier input attribute contains the name of the requested yang module (revision can be specified also in case of multiple revisions of one module under tag *version*).

Debugging

A simple way to check both connectivity and whether Netconf Server is up and running is to use SSH as described in Using the NETCONF Protocol over Secure Shell (SSH). For example:

ssh -s repenno@10.0.1.27 -p 830 netconf

Configuring Netconf logging

Edit file logback.xml. If compiling from source it will be:

./opendaylight/distribution/opendaylight/target/distribution.opendaylight-osgipackage/opendaylight/configuration/logback.xml

Add the following two lines:

```
<logger name="org.opendaylight.controller.netconf" level="DEBUG" />
<logger name="org.opendaylight.controller.sal.connect.netconf" level="DEBUG" />
```

Other ways to configure mount point

Configuring Netconf client with yangcli-pro

Requirements

yangcli-pro version 13.04-9.2 or later.

Connecting with yuma

Yuma is commercial software that can be used to access netconf server. Please see User guide for details.

Configuring an NCC Instance

This example shows how to configure the NCC to connect to the collector's own Netconf server. The controller's configuration subsystem can then be accessed through the controller's NB Restconf API.

Once yangcli-pro connects to the controller's Netconf server, issue the following sequence of commands:

- Enter **merge /modules/module**. yangcli-pro will prompt for a string value for leaf <name>. This is the name of the module that we are configuring.
- Enter **controller** for the name of the module, since we are configuring the NCC for the controller (you can enter any ascii name you like). yangcli-pro will prompt for a string value for leaf <type>. This is the type of the module that we are configuring.
- Enter **sal-netconf-connector**. yangcli-pro will print a list of possible choices for the selection of the case statement, as shown below.

```
  1: case netty-threadgroup-fixed:
       leaf thread-count
  2: case binding-broker-impl:
       container data-broker
       container notification-service
  3: case binding-data-broker:
       container dom-broker
       container mapping-service
  4: case logback:
       list file-appenders
       list rolling-appenders
       list console-appenders
       list loggers
  5: case dom-broker-impl:
       container data-store
  6: case threadpool-flexible:
       leaf max-thread-count
       leaf minThreadCount
       leaf keepAliveMillis
       container threadFactory
  7: case remote-zeromq-rpc-server:
       container dom-broker
       leaf port
  8: case shutdown:
       leaf secret
       leaf old-secret
  9: case threadpool-scheduled:
       leaf max-thread-count
       container threadFactory
  10: case netty-hashed-wheel-timer:
       leaf tick-duration
       leaf ticks-per-wheel
       container thread-factory
  11: case async-eventbus:
       container threadpool
  12: case threadfactory-naming:
       leaf name-prefix
  13: case sal-netconf-connector:
       leaf address
       leaf port
       leaf tcp-only
       leaf username
       leaf password
       container dom-registry
       container boss-thread-group
       container worker-thread-group
       container event-executor
  14: case threadpool-fixed:
       leaf max-thread-count
```

- *Enter the number of the line that contains the* sal-netconf-connector *case (**13** in the list above).*
- *yuma then starts asking about each leaf to be filled. Full output should look like this:*

```
admin@localhost> merge /modules/module

Filling list /modules/module:
Filling key leaf /modules/module/name:
Enter string value for leaf <name>
admin@localhost:merge> controller

Filling mandatory leaf /modules/module/type:
Enter identityref value for leaf <type>
admin@localhost:merge> sal-netconf-connector

Filling choice /modules/module/configuration:
Enter the number of the selected case statement:

  1: case netconf-client-dispatcher:
       container boss-thread-group
       container worker-thread-group
       container timer
  2: case netty-threadgroup-fixed:
       leaf thread-count
  3: case kitchen-service-impl:
       container rpc-registry
       container notification-service
  4: case binding-broker-impl:
       container data-broker
       container notification-service
  5: case binding-data-broker:
       container dom-broker
       container mapping-service
  6: case binding-data-compatible-broker:
       container dom-async-broker
```

```
       container binding-mapping-service
  7: case threadpool-flexible:
       leaf max-thread-count
       leaf minThreadCount
       leaf keepAliveMillis
       container threadFactory
  8: case threadpool-scheduled:
       leaf max-thread-count
       container threadFactory
  9: case never-reconnect-strategy-factory:
       leaf timeout
       container executor
  10: case reconnect-immediately-strategy-factory:
       leaf timeout
       container executor
  11: case timed-reconnect-strategy-factory:
       leaf deadline
       leaf max-attempts
       leaf max-sleep
       leaf min-sleep
       leaf sleep-factor
       leaf connect-time
       container executor
  12: case netty-hashed-wheel-timer:
       leaf tick-duration
       leaf ticks-per-wheel
       container thread-factory
  13: case async-eventbus:
       container threadpool
  14: case threadfactory-naming:
       leaf name-prefix
  15: case sal-netconf-connector:
       leaf address
       leaf port
       leaf tcp-only
       leaf username
       leaf password
       container dom-registry
       container binding-registry
       container boss-thread-group
       container worker-thread-group
       container event-executor
       container processing-executor
       container client-dispatcher
       leaf connection-timeout-millis
       leaf max-connection-attempts
       leaf between-attempts-timeout-millis
       leaf sleep-factor
  16: case dom-broker-impl:
       container data-store
       container async-data-broker
  17: case dom-inmemory-data-broker:
       container schema-service
  18: case logback:
       list file-appenders
       list rolling-appenders
       list console-appenders
       list loggers
  19: case remote-zeromq-rpc-server:
       container dom-broker
       leaf port
  20: case toaster-provider-impl:
       container rpc-registry
       container notification-service
       container data-broker
  21: case shutdown:
       leaf secret
  22: case threadpool-fixed:
       leaf max-thread-count
       container threadFactory

Enter case number [1 - 22]:
admin@localhost:merge> 15

Filling mandatory case /modules/module/configuration/sal-netconf-connector:
Filling optional leaf /modules/module/configuration/sal-netconf-connector/address:
Enter string value for leaf <address>
admin@localhost:merge> 127.0.0.1

Filling optional leaf /modules/module/configuration/sal-netconf-connector/port:
Enter uint32 value for leaf <port>
admin@localhost:merge> 1830

Filling optional leaf /modules/module/configuration/sal-netconf-connector/tcp-only:
```

```
Enter boolean value for leaf <tcp-only>
admin@localhost:merge> false

Filling optional leaf /modules/module/configuration/sal-netconf-connector/username:
Enter string value for leaf <username>
admin@localhost:merge> admin

Filling optional leaf /modules/module/configuration/sal-netconf-connector/password:
Enter string value for leaf <password>
admin@localhost:merge> admin

Filling container /modules/module/configuration/sal-netconf-connector/dom-registry:
Filling mandatory leaf /modules/module/configuration/sal-netconf-connector/dom-registry/type:
Enter service-type-ref value for leaf <type>
admin@localhost:merge> dom-broker-osgi-registry

Filling mandatory leaf /modules/module/configuration/sal-netconf-connector/dom-registry/name:
Enter leafref value for leaf <name>
admin@localhost:merge> dom-broker

Filling container /modules/module/configuration/sal-netconf-connector/binding-registry:
Filling optional leaf /modules/module/configuration/sal-netconf-connector/binding-registry/type:
Enter service-type-ref value for leaf <type>
admin@localhost> binding-broker-osgi-registry

Filling mandatory leaf /modules/module/configuration/sal-netconf-connector/binding-registry/name:
Enter leafref value for leaf <name>
admin@localhost> binding-osgi-broker

Filling container /modules/module/configuration/sal-netconf-connector/boss-thread-group:
Filling optional leaf /modules/module/configuration/sal-netconf-connector/boss-thread-group/type:
Enter service-type-ref value for leaf <type>
admin@localhost> netty-threadgroup

Filling mandatory leaf /modules/module/configuration/sal-netconf-connector/boss-thread-group/name:
Enter leafref value for leaf <name>
admin@localhost> global-boss-group

Filling container /modules/module/configuration/sal-netconf-connector/worker-thread-group:
Filling optional leaf /modules/module/configuration/sal-netconf-connector/worker-thread-group
/type:
Enter service-type-ref value for leaf <type>
admin@localhost> netty-threadgroup

Filling mandatory leaf /modules/module/configuration/sal-netconf-connector/worker-thread-group
/name:
Enter leafref value for leaf <name>
admin@localhost> global-boss-group

Filling container /modules/module/configuration/sal-netconf-connector/event-executor:
Filling mandatory leaf /modules/module/configuration/sal-netconf-connector/event-executor/type:
Enter service-type-ref value for leaf <type>
admin@localhost> netty-event-executor

Filling mandatory leaf /modules/module/configuration/sal-netconf-connector/event-executor/name:
Enter leafref value for leaf <name>
admin@localhost> global-event-executor

Filling container /modules/module/configuration/sal-netconf-connector/processing-executor:
Filling optional leaf /modules/module/configuration/sal-netconf-connector/processing-executor
/type:
Enter service-type-ref value for leaf <type>
admin@localhost> threadpool

Filling mandatory leaf /modules/module/configuration/sal-netconf-connector/processing-executor
/name:
Enter leafref value for leaf <name>
admin@localhost> global-netconf-processing-executor

Filling container /modules/module/configuration/sal-netconf-connector/client-dispatcher:
Filling optional leaf /modules/module/configuration/sal-netconf-connector/client-dispatcher/type:
Enter service-type-ref value for leaf <type>
admin@localhost> netconf-client-dispatcher

Filling mandatory leaf /modules/module/configuration/sal-netconf-connector/client-dispatcher/name:
Enter leafref value for leaf <name>
admin@localhost> global-netconf-dispatcher

Filling optional leaf /modules/module/configuration/sal-netconf-connector/connection-timeout-
millis:
Enter uint32 value for leaf <connection-timeout-millis> [20000]
admin@localhost>

Filling optional leaf /modules/module/configuration/sal-netconf-connector/max-connection-attempts:
```

```
Enter uint32 value for leaf <max-connection-attempts> [0]
admin@localhost>

Filling optional leaf /modules/module/configuration/sal-netconf-connector/between-attempts-
timeout-millis:
Enter uint16 value for leaf <between-attempts-timeout-millis> [2000]
admin@localhost>

Filling optional leaf /modules/module/configuration/sal-netconf-connector/sleep-factor:
Enter decimal64 value for leaf <sleep-factor> [1.5]
admin@localhost>

RPC OK Reply 75 for session 2 [default]:

admin@localhost> commit

RPC OK Reply 76 for session 2 [default]:

admin@localhost>
```

Editing Netconf Client Configuration in the Current Controller Config File

Configuration for different controller modules can be added into the controller's current configuration located in 'configuration/current/controller.currentconfig.xml'. It it important to notice that initially the file controller.currentconfig.xml does not exist whether pulling and compiling from source code or downloading the zipped distribution, it is created after you run the controller the first time. Create the controller.currentconfig.xml file or if the controller.currentconfig.xml exists and have a configuration such as OpenDaylight Controller:Config:Examples:Netconf:Example Configuration, insert the following stanza into the <modules></modules> section to add the above configuration:

```
 <module>
      <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
prefix:sal-netconf-connector</type>
      <name>controller</name>
      <port xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">8383</p
ort>
      <username xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">foo
</username>
      <worker-thread-group xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:
netconf">
          <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:netty">prefix:netty-th
readgroup</type>
          <name>global-worker-group</name>
      </worker-thread-group>
      <address xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">192.
168.4.1</address>
      <tcp-only xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">true
</tcp-only>
      <event-executor xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:
netconf">
          <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:netty">prefix:netty-ev
ent-executor</type>
          <name>global-event-executor</name>
      </event-executor>
      <password xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">bar
</password>
      <boss-thread-group xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:
netconf">
          <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:netty">prefix:netty-th
readgroup</type>
          <name>global-boss-group</name>
      </boss-thread-group>
      <dom-registry xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">

           <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:dom">prefix:dom
-broker-osgi-registry</type>
           <name>dom-broker</name>
      </dom-registry>
</module>
```

To change the configuration for a different target Netconf server, edit the <name/>, <address/>, <port/>, <tcp-only/>, <username/> and <password/> entities.

An example configuration file can be found in OpenDaylight Controller:Config:Examples:Netconf:Example Configuration

Upgrade Considerations

ODL has no explicit upgrade process for applications; most of the work for upgrades is done manually. In order to preserve netconf modules configured through the configuration subsystem (or using loopback), the ODL operator must make appropriate efforts to glean restoration information from the existing controller configuration. The controller configuration file is located here:

```
${KARAF_HOME}/etc/opendaylight/current/controller.currentconfig.xml
```

Before starting this process, a backup of the current controller configuration should be made:

```
cp ${KARAF_HOME}/etc/opendaylight/current/controller.currentconfig.xml SOME_BACKUP_LOCATION
```

The operator should then copy out the netconf modules. The netconf modules look like this:

```
  <module>
    <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf"
>prefix:sal-netconf-connector</type>
    <name>netconf-testtool-custom-directory</name>
    <port xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">17830<
/port>
    <connection-timeout-millis xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:
connector:netconf">20000</connection-timeout-millis>
    <schema-cache-directory xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:
connector:netconf">custom_directory_1</schema-cache-directory>
    <between-attempts-timeout-millis xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:
connector:netconf">2000</between-attempts-timeout-millis>
    <reconnect-on-changed-schema xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:
connector:netconf">false</reconnect-on-changed-schema>
    <sleep-factor xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf"
>1.5</sleep-factor>
    <password xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf"
>admin</password>
    <dom-registry xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf">
        <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:dom">prefix:dom-
broker-osgi-registry</type>
        <name>dom-broker</name>
      </dom-registry>
      <default-request-timeout-millis xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:
sal:connector:netconf">60000</default-request-timeout-millis>
      <client-dispatcher xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:
netconf">
        <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:config:netconf">prefix:
netconf-client-dispatcher</type>
        <name>global-netconf-dispatcher</name>
      </client-dispatcher>
      <username xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf"
>admin</username>
      <keepalive-delay xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:
netconf">120</keepalive-delay>
      <address xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf"
>127.0.0.1</address>
      <processing-executor xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:
netconf">
        <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:threadpool">prefix:
threadpool</type>
        <name>global-netconf-processing-executor</name>
      </processing-executor>
      <tcp-only xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:netconf"
>false</tcp-only>
      <keepalive-executor xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:
netconf">
        <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:threadpool">prefix:
scheduled-threadpool</type>
         <name>global-netconf-ssh-scheduled-executor</name>
       </keepalive-executor>
       <binding-registry xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:
netconf">
         <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding"
>prefix:binding-broker-osgi-registry</type>
         <name>binding-osgi-broker</name>
       </binding-registry>
       <max-connection-attempts xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:
connector:netconf">0</max-connection-attempts>
       <event-executor xmlns="urn:opendaylight:params:xml:ns:yang:controller:md:sal:connector:
netconf">
       <type xmlns:prefix="urn:opendaylight:params:xml:ns:yang:controller:netty">prefix:netty-
event-executor</type>
         <name>global-event-executor</name>
       </event-executor>
     </module>
```

Once this information is extracted and saved, then you can upgrade the controller software. After the controller has been upgraded, follow [2] to add the netconf modules again.
Additionally, please consider switching to the new netconf mounting mechanism [3]. This will require you to transform the model, which must be done manually but is fairly straightforward.