DAEXIM

Release master

Sep 18, 2019

Contents

CHAPTER 1

Data Export/Import Developer Guide

1.1 Overview

This feature is used to export the system data tree state, or part of, to the system's file system. It may also be used to import the system data tree state, or part of, from the system's file system.

1.2 Data Export/Import Architecture

The daexim feature consists of a single feature, which interacts with MD-SAL to export and import the system data.

1.3 Key APIs and Interfaces

The APIs are available via REST. The details are provided are in user-guide.

1.4 API Reference Documentation

The details of the API are also available in the YANG model for this feature. This model is accessiable via the APIDOC explorer interface.

CHAPTER 2

Data Export/Import User Guide

2.1 Overview

Data Export/Import is known as "daexim" (pronounced 'deck-sim') for short. The intended audience are administrators responsible for operations of OpenDaylight.

Data Export/Import provides an API (via RPCs) to request the bulk transfer of OpenDaylight system data between its internal data stores and the local file system. This can be used for scheduling data exports, checking the status of data being exported, canceling data export jobs, importing data from files in the file systems, and checking the import status.

Such export and import of data can be used during system upgrade, enabling the development of administrative procedures that make reconfigurations of the base system without concern of internal data loss.

Data Export produces a models declaration file and one or more data files. The models declaration file records exactly which YANG models were loaded (by module name, revision date and namespace). The data file(s) contain data store data as per the draft-ietf-netmod-yang-json RFC.

Data Import takes a models declaration file and zero or more data files. The models declaration file is used to check that the listed models are loaded before importing any data. Data is imported into each data store in turn with one transaction executed for each data store, irrespective of the number of files for that data store, as inter-module data dependencies may exist. Existing data store data may be cleared before importing.

2.2 Data Export Import Architecture

The daexim feature is a single feature. This feature leverages the existing infrastructure provided by MD-SAL and yangtools.

2.3 Installing the Feature

To install the feature perform the following steps.

```
karaf > feature:install odl-daexim-all
```

The interactions with this feature are via Restconf RPCs. The details are provided in the *Tutorials*.

2.4 Tutorials

The following tutorials provide examples of REST API that are supported by the Data Export/Import feature. As for all ODL RESTCONF calls, the following are the common setting for REST calls:

- **Headers:**
  - **Content-Type:** `application/json`
  - **Accept:** `application/json`
  - **Authentication:** `admin:admin`
- **Method:** `HTTP POST`
- <controller-ip>: Host (or IP) where OpenDaylight controller is running, e.g. localhost
- <restconf-port>: TCP port where RESTCONF has been configured to listen, e.g. 8181 by default

The files created by export are placed in a subdirectory called `daexim/` in the installation directory of OpenDaylight. Similarly files to import must be placed in this `daexim/` subdirectory.

2.4.1 Scheduling Export

The **schedule-export** RPC exports data from data store at a specific time in the future. Each exported file has a JSON-encoded object that contains module data from the corresponding data store. Each file contains at least one empty JSON object.

URL: `http://<controller-ip>:<restconf-port>/restconf/operations/data-export-import:schedule-export`

Payload:

```
{
   "input": {
     "data-export-import:run-at": 500
   }
}
```

The following export options are available:

- run-at
- local-node-only
- strict-data-consistency
- split-by-module
- included-modules
- excluded-modules

The `run-at` time may be specified as an absolute UTC time or a relative offset from the server clock. Attempts to schedule an export in the past times are rejected.

In a clustered environment, if `local-node-only` is provided with a value of `true`, the export operation is performed only on the node on which the RPC was received; otherwise, the export operation is performed on all nodes in the cluster.

```
{
    "input": {
        "data-export-import:run-at": 500,
        "data-export-import:local-node-only": true
    }
}
```

The `strict-data-consistency` flag may be used to specify if strict data consistency needs to be maintained while exporting data. This value determines how data is read from the datastore during export - in one shot (`true` - default) or in smaller chunks (`false`).

```
{
    "input": {
        "data-export-import:run-at": 500,
        "data-export-import:strict-data-consistency": false
    }
}
```

The `split-by-module` flag may be used to request exported data to be split by module name. If value of this flag is `true`, then export process will create separate json files for every top-level container present in the data store.

```
{
    "input": {
        "data-export-import:run-at": 500,
        "data-export-import:split-by-module": true
    }
}
```

Options `included-modules` and `excluded-modules` can be used to include and/or exclude specific modules while exporting the data. Modules are specified according to each data store. If both options are specified, then a module is only exported if it is included (whilelisted) but not also excluded (blacklisted).

Guidelines for including/excluding data are:

- The data store name can be `config` or `operational`.
- To select a module, you can use a specific module name or a wildcard (*). Note that wildcard input is currently supported for excluded-modules only. If you use a wild card for a module, all modules from that data store are excluded.
- To include/exclude all the data of a specific module, specify a list of each data store and each item by using the same module name.

```
{
    "input": {
        "data-export-import:run-at": 500,
        "data-export-import:included-modules" : [
            {
                "module-name": "bgp-rib",
                "data-store": "config"
            }
        ],
        "data-export-import:excluded-modules" : [
            {
```

(continues on next page)

(continued from previous page)

```
                "module-name": "*",
                "data-store": "config"
            }
        ]
    }
}
```

2.4.2 Checking Export Status

The **status-export** RPC checks the status of the exported data. If the status has the value of `initial`, an export has not been scheduled. If the status has the value of `scheduled`, `run-at` indicates the time at which the next export runs. If the status has the value of `in-progress`, `run-at` indicates the time at which the running export was scheduled to start. A status of `tasks` indicates activities that are scheduled and currently being performed. The `tasks` status serves as an indicator of progress and success of the activity. If the status has any other value, `run-at` indicates the time at which the last export was scheduled to start; and `tasks` indicates the activities that were undertaken. If the status for any node has failed, the corresponding reason for failure is listed.

URL: `http://<controller-ip>:<restconf-port>/restconf/operations/data-export-import:status-export`

Payload: No payload

2.4.3 Canceling Scheduled Export

The **cancel-export** RPC cancels an already scheduled data export job. To cancel the export, the server stops the tasks that are running (where possible, immediately), clears any scheduled export time value, and releases the associated resources. This RPC may be called at any time, whether an export is in progress, scheduled or not yet scheduled. The returned result is `True` when the server has successfully cleared tasks, the state, and resources. The status is `False` on failure to do so. Note that if no export is scheduled or running, there is no tasks for the server to clear. Therefore, the return result is `True` because the server cannot fail.

URL: `http://<controller-ip>:<restconf-port>/restconf/operations/data-export-import:cancel-export`

Payload: No payload

2.4.4 Importing from a file

The **immediate-import** RPC imports data from files already present in the file system.

URL: `http://<controller-ip>:<restconf-port>/restconf/operations/data-export-import:immediate-import`

Payload:

```
{
   "input" : {
     "check-models" : true,
     "clear-stores" : "all"
   }
}
```

The following import options are available:

- check-models
- clear-stores
- file-name-filter
- strict-data-consistency

The following table lists the options for `check-models`.

Boolean flag	Controller action
`true`	If the boolean flag is `true` then the import process reads the models declaration file and checks that all declared models are loaded before performing any data modifications. If the application cannot verify the models declaration file, the file has bad content, or any declared model is not loaded, then no data modifications are performed and a results of false is returned. This is the default value.
`false`	The check is skipped. If a models declaration file is present, it is ignored.

The following table lists the options for clear-stores.

Enum value	Controller action
`all`	Data in all of the data stores is deleted and the new data is imported. This is the default value.
`data`	All data in the data stores for which data files are supplied is deleted. For example, if only the configuration data is provided (even for a single module), the entire configuration data store is cleared before importing data and the operational data store is untouched. A similar behavior occurs with the any operational data too, where the operational data store is purged and configuration data store is untouched. If the input files contain both configuration and operational data (even for a single module) and this flag is used, both data stores are cleared completely before importing any data, essentially making the option equivalent to `all`.
`none`	Data is not deleted explicitly from any store. The data provided in the data files is imported into the data stores by using the `PUT` operation. Note that this done at the highest container level. So, depending on the data in the JSON files, you may lose some data in the target controller.

The following table lists the options for file-name-filter.

Value	Controller action
`.*` or empty	If property value is `.*` or is omitted (empty), then all data files are considered for import.
regular expression	Each data file's filename is matched against provided regular expression. Only those that match are considered for import.

The `strict-data-consistency` flag may be used to specify if strict data consistency needs to be maintained while importing data. This value determines how data is written to the datastore during import - in one shot (*true* - default) or in smaller chunks (*false*).

```
{
    "input" : {
        "check-models" : true,
        "clear-stores" : "none",
        "file-name-filter": ".*topology.*",
        "strict-data-consistency": false
    }
}
```

2.4.5 Status of Import

The **status-import** RPC checks the last import status. If the status has the value of `initial`, an import has not taken place. For all other values of status, `imported-at` indicates the time at which the restoration has taken place. List nodes hold status about the restoration for each node.

URL: `http://<controller-ip>:<restconf-port>/restconf/operations/data-export-import:status-import`

Payload: No payload

2.4.6 Importing from a file automatically on boot

Any files placed inside the `daexim/boot` subdirectory are automatically imported on start-up. The import performed is the exact same as the one by explicit **immediate-import** RPC, which imports from files `daexim/`, except it happens automatically.

The import on boot happens after all other ODL OSGi bundles have successfully started. The INFO log and **status import** automatically reflect when the boot import is planned (via `boot-import-scheduled`), when the boot import is ongoing (via `boot-import-in-progress`), and when the boot import fails (via `boot-import-failed`).

Upon completion or failure of this boot import, the files inside the `daexim/boot` directory are renamed to `.imported` in order to avoid another import on the next start.