Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 13, 2017

A. Shaikh
R. Shakir
Google
K. D'Souza
AT&T

March 12, 2017

Catalog and registry for YANG models
draft-openconfig-netmod-model-catalog-02

Abstract

This document presents an approach for a YANG model catalog and registry that allows users to find models relevant to their use cases from the large and growing number of YANG modules being published. The model catalog may also be used to define bundles of YANG modules required to realize a particular service or function.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

As YANG [RFC6020][RFC7950] adoption and usage grows, the number of YANG models (and corresponding module and submodule files) published is increasing rapidly. This growing collection of modules potentially enables a large set of management use cases, but from a user perspective, it is a daunting task to navigate the largely ad-hoc landscape of models to determine their functionality, compatibility, and available implementations. For example, the IETF Routing Area Coordination page [RTG-AD-YANG] currently tracks over 150 YANG models related to layer 2 and layer 3 technologies.

YANG models are also being developed and published beyond the IETF, for example by open source projects, other standards organizations, and industry forums. These efforts are generally independent from each other and sometimes result in overlapping models. While we recognize that models may come from multiple sources, the current approach of having a flat online listing of models is not sufficient to help users find the models they need, along with the information to retrieve and utilize the models in actual operational systems. There is a need for a wider registry and catalog of available models that provides a central reference for model consumers and developers.

The idea of a model catalog is inspired by service catalogs in traditional IT environments. Service catalogs serve as software-based registries of available services with information needed to discover and invoke them.

In earlier proposals [I-D.openconfig-netmod-model-structure] we motivated the need for a common structure that allows a set of models to be used together coherently in order to manage, for example, a complete network device. Other efforts have subsequently proposed further options for modeling the complete device structure [I-D.rtgyangdt-rtgwg-device-model]. We also briefly described the notion of a model catalog to provide a structured view of all of the models available from different organizations. In this document, we further elaborate on some of the details and use cases for a model catalog and registry.

There are recent proposals that address related issues in terms of understanding the set of YANG models available on a device [RFC7895], and how to classify models based on their role in describing a multi-layer service [I-D.ietf-netmod-yang-model-classification]. The latter, in particular, describes a taxonomy for classifying YANG models that could also be used in the model catalog, though it does not address the problem of expressing model functionality, which is a key requirement.

2. Model catalog and registry requirements

At a high level, the model catalog must provide enough information for users to determine which YANG modules or module bundles are available to describe a specific service or technology, and attributes of those modules that would help the user select the best model for their scenario. While this draft does not specifically address selection criteria -- they would be specific to each user -- some examples include:

o model maturity, including availability of server implementations (e.g., native device support)

o availablility of co-requisite models, and complexity of the model dependencies

o identity and reputation of the entity or organization publishing the model

The model catalog should, therefore, include key information about YANG modules, including:

- o organization responsible for publishing and maintaining the module with contact information; organizations may include standards bodies (SDOs), industry forums, open source projects, individuals, etc.

- o classification of the module or model, which could be along several axes, e.g., functional category, service vs. element models, commercial vs. free-to-use, etc.; currently, identities are available for the classifications defined in [I-D.ietf-netmod-yang-model-classification]

- o groupings of modules (and their versions) that are compatible with each other, and together provide a defined set of functionality

- o for open models, the license under which the model is distributed; this is important if there are limitations on how the model may be modified or redistributed

- o module dependencies, e.g., a list of all of the YANG modules that are required

- o pointer to the YANG module, e.g., a machine-readable URI and authentication information to allow users to verify that the model they retrieve is authentic and unaltered

- o implementation information, for example, a list of available server implementations that support the module

Establishing a globally applicable classification scheme for models is not straightforward -- each organization developing models likely has its own taxonomy or organization strategy for YANG modules. This is an area of the catalog that is likely to require extensibility and customization, e.g., by letting each organization augment the schema with its own categories. Similarly, users may want to define their own classifications for use by internal systems.

The proposed catalog schema should be useful as a local database, deployed by a single user, and also as a global registry that can be used to discover available models. For example, the local catalog could be used to define the approved set of models for use within an organization, while the registry serves as a channel for all model developers to make information about their models available. The IETF XML Registry [RFC3688], maintained by IANA serves a similar purpose for XML documents used in IETF protocols, but it is limited to IETF-defined YANG models, is tied to XML encoded data, and has a very limited schema.

The registry implementation could be as simple as a metadata database that reflects the proposed catalog schema, along with means for online access and viewing. A key requirement for the online registry would be a robust query capability that allows users to search for modules meeting a variety of selection criteria, along with an easy way to retrieve modules (where applicable).

3. Organizing YANG modules

We propose a schema for the model catalog defined using YANG (see the modules in Section 9). The YANG modules and groupings in the catalog are organized at the top level by the publishing organization and its associated contact information. The catalog structure is shown below.

```
     +--rw organizations
       +--rw organization* [name]
          +--rw name               string
          +--rw type?              identityref
          +--rw contact?           string
          +--rw modules
          |     ...
          +--rw release-bundles
          |     ...
          +--rw feature-bundles
          |     ...
          +--rw implementations
                ...
```

In this model, each organization publishes a list of available modules, each module having associated data describing its version, dependencies, and other basic metadata. Organizations may also publish release bundles, which are groupings of compatible modules, or feature bundles, which describes specific functionality.

3.1. Module information

Each module has several types of information associated with it. These are described below (only node names are shown).

```
           +--rw modules
                +--rw module* [name version]
                   +--rw name
                   +--rw version
                   +--rw namespace?
                   +--rw prefix?
                   +--rw revision?
                   +--rw summary?
                   +--rw classification
                   |  +--rw category?
                   |  +--rw subcategory?
                   |  +--rw deployment-status?
                   +--rw dependencies
                   |  +--rw required-module*
                   +--rw access
                   |  +--rw uri?
                   |  +--rw md5-hash?
                   +--rw submodules
                      +--rw submodule* [name]
                         +--rw name
                         +--rw access
                            +--rw uri?
                            +--rw md5-hash?
```

The basic information includes module metadata, such as its version which may be different from the YANG revision statement as in [OC-SEMVER]. Other common information includes the module’s prefix, namespace, and a summary of its functionality.

The classification data includes some base information but leaves the taxonomy largely to model publishers. The category and subcategory leaves are identities that are expected to be augmented with additional values. The current version includes classification categories defined in [I-D.ietf-netmod-yang-model-classification]. The classification also includes a status to indicate the development or deployment status of the module, e.g., whether it is purely experimental, or mature enough for production use.

Module dependencies are represented as a simple list of references to co-requisite modules indicated by ’import’ statements in the module. Only the first-level dependencies are included in the list. That is, each of the listed dependences can be examined in turn to determine its dependencies.

The access data contains information required to retrieve and validate the module. Specifically, it includes a URI that can be used to download modules directly. It also includes a simple MD5 checksum to allow checking the data integrity of the module. Further data for verifying authenticity and origin of the module may be added in future versions of the catalog.

For YANG modules that are composed of submodules, the submodules container provides their names and access information. Note that the submodules are an integral part of their parent modules, and hence are listed together with their parent and corresponding version.

4. Identifying interoperable models

YANG models for configuration and operational state data are under active development and still maturing, especially with regard to their use in production networks. As models (and their corresponding YANG modules) evolve and are revised, there is a significant challenge for users to identify the set of models that are known, or designed, to work together. This is made more complicated by the fact that models are being sourced by different organizations which may use different modeling conventions. Since there are often cross-dependencies between modules (e.g., interface configuration and various routing protocols), it is critical that users understand which modules can be used together.

The model catalog defines the notion of "release" bundles which provide a grouping of YANG modules that are part of a cohesive release. For example, a release bundle can be defined at a granular level to collect all of the modules related to interface configuration that are known to work together. These bundles can be further grouped into larger releases of models that interoperate, e.g., a release containing interoperable routing, interface, and policy-related modules.

Release bundles are also useful for implementors to know which dependencies must have what versions in order to work with a given module. For example, when an implementor wishes to support a new version of a module, the release bundle provides information about what other modules need to be upgraded in order to be compatible. It is expected that the publisher of the bundle ensures version compatibility of the release, although release bundles and the modules they include do not necessarily need to be from the same organization. We expect, however, that users and publishers of modules would be the primary source of release bundle definitions, and vendors and implementors would be the primary consumers.

4.1. Release bundle information

The schema for release bundles is shown below (only node names are shown).

```
           +--rw release-bundles
              +--rw release-bundle* [name version]
                 +--rw name       string
                 +--rw version    oc-cat-types:module-version-type
                 +--rw members
                    +--rw member* [id]
                       +--rw id
                       +--rw type?
                       +--rw module?
                       +--rw release-bundle?
                       +--rw publisher?
                       +--rw compatible-versions*
```

The release bundle has a name and version assigned to the bundle itself, and a list of members that are part of the bundle. The list may include a reference to a module or another bundle. The compatible-versions list indicate which semantic versions [OC-SEMVER] of the respective module or bundle are known to work together.

5. Specifying functionality with feature bundles

From an operational perspective, the utility of a single module is quite limited. Most, if not all, use cases require multiple modules that work together coherently. Managing a network device typically requires configuration and operational state models for device-wide services, network protocols, virtual instances, etc. Network services, such as those delivered by many service providers, require not only infrastructure-level management models, such as devices and protocols, but also service-level models that describe service parameters.

The model catalog and registry provides a common way to define feature bundles that describe the set of schema paths required to realize a feature or service. The feature bundle paths are specified against a release bundle that ensures the paths are drawn from a set of compatible modules and/or bundles.

Feature bundles are useful for defining specific sets of functionality that can be further composed to build higher level features or services. For example, a Layer 3 VPN bundle could be composed of more specific features such as interfaces, routing, policy, and QoS. Note these bundle definitions complement the configuration models for such services, which may focus on providing an abstracted set of configuration or operational state variables. These variables would then be mapped onto device level variables.

Feature bundle definitions can also be used by organizations to identify a canonical set of modules that should be used to build a particular service. Users within the organization can be assured that the corresponding bundles are known and approved to work together to support the desired service.

Finally, a key use case for feature bundles is to define specific units of compliance for an implementation. Users can define a feature bundle containing only those paths that are required for a given usage, allowing an implementor or vendor to focus their implementation and testing on those paths rather than having to implement the entire contents of a module or bundle. The implementor may also wish to publish a deviation module that indicates which paths are not supported.

5.1. Feature bundle information

The schema for feature bundles in the catalog is shown below (note only node names are shown).

```
          +--rw feature-bundles
             +--rw feature-bundle* [name version]
                +--rw name
                +--rw version
                +--rw path*
                +--rw release-bundle
                |  +--rw name?
                |  +--rw publisher?
                |  +--rw version?
                +--rw feature-bundles
                   +--rw feature-bundle* [name]
                      +--rw name
                      +--rw publisher?
                      +--rw version?
```

Each feature bundle includes basic information such as the name of the feature or service, the bundle version, and the set of specific schema paths. The schema paths are based on the release bundle specified as part of the feature bundle. For simplicity, only one release bundle may be specified. If the schema paths in a feature bundle cross release bundle boundaries, a new release bundle should be created to include all of the paths needed by the feature bundle. The feature bundle may itself be composed of more granular feature bundles. This allows the definition of "base" features that can be reused across feature bundles.

6. Module implementations

Model implementors can use the catalog to indicate the data models they support using the implementation container. An implementation is expected to indicate a set of feature bundles it supports. The feature bundles may be defined by a user (i.e., a set of compliance units), or by the implementor or vendor to indicate the full list of what is supported.

6.1. Implementation information

The implementation information in the catalog is shown below (only node names are shown):

```
            +--rw implementations
             +--rw implementation* [id]
                +--rw id
                +--rw description?
                +--rw reference?
                +--rw platform?
                +--rw platform-version?
                +--rw status?
                +--rw feature-bundles
                   +--rw feature-bundle* [name version]
                      +--rw name
                      +--rw publisher?
                      +--rw version
```

The implementation container provides information about the platform and version on which the feature bundles are supported, as well as the status of the implementation. It also includes a URI reference to retrieve artifacts or further information on the implementation. The list of supported feature bundles are references to defined feature bundles in the catalog.

7. Security Considerations

The model catalog and registry described in this document do not define actual configuration and state data, hence are not directly responsible for security risks.

However, since the model catalog is intended to be an authoritative and authenticated database of published modules, there are security considerations in securing the catalog (both contents and access), and also in authenticating organizations that deposit data into the catalog.

8. IANA Considerations

The YANG model catalog is intended to complement the IANA XML Registry. YANG modules defined in this document may be entered in the XML registry if they are placed or redirected for the standards track, with an appropriate namespace URI.

9. YANG modules

The main model catalog and associated types modules are listed below.

```
<CODE BEGINS> file "openconfig-catalog-types.yang"

module openconfig-catalog-types {

  yang-version "1";

  // namespace
  namespace "http://openconfig.net/yang/catalog-types";

  prefix "oc-cat-types";

  import openconfig-extensions { prefix oc-ext; }


  // meta
  organization "OpenConfig working group";

  contact
    "OpenConfig working group
    www.openconfig.net";

  description
    "This module defines types and identities used by the OpenConfig
    YANG module catalog model.";

  oc-ext:openconfig-version "0.2.0";

  revision "2017-03-08" {
    description
      "OpenConfig public release";
    reference "0.2.0";
  }

  revision "2016-02-15" {
    description
      "Initial OpenConfig public release";
    reference "0.1.0";
```

```
     }

     revision "2015-10-18" {
       description
         "Initial revision";
       reference "TBD";
     }

     // extension statements

     // feature statements

     // identity statements

     identity CATALOG_MEMBER_TYPE {
       description
         "Base identity for elements in the catalog";
     }

     identity MODULE {
       base CATALOG_MEMBER_TYPE;
       description
         "Module elements in the catalog";
     }

     identity RELEASE_BUNDLE {
       base CATALOG_MEMBER_TYPE;
       description
         "Release bundle elements in the catalog";
     }

     identity FEATURE_BUNDLE {
       base CATALOG_MEMBER_TYPE;
       description
         "Feature bundle elements in the catalog";
     }


     identity IMPLEMENTATION_STATUS_TYPE {
       description
         "Indications of the status of a module’s implementation on a
         device or server";
     }

     identity IN_PROGRESS {
       base IMPLEMENTATION_STATUS_TYPE;
       description
         "Implementation is in progress";
     }

     identity PLANNED {
       base IMPLEMENTATION_STATUS_TYPE;
       description
         "Implementation is planned";
     }

     identity COMPLETE {
       base IMPLEMENTATION_STATUS_TYPE;
       description
         "Implementation is complete and fully supports the model";
     }

     identity PARTIAL {
       base IMPLEMENTATION_STATUS_TYPE;
       description
         "Implementation is complete, but only supports the model
         partially";
     }

     identity MODULE_STATUS_TYPE {
       description
         "Indicates the deployment status of the module";
     }

     identity EXPERIMENTAL {
       base MODULE_STATUS_TYPE;
       description
         "Module should be considered experimental, not deployed in
         production settings";
     }

     identity PRODUCTION {
       base MODULE_STATUS_TYPE;
       description
         "Module is suitable for use in production, or has been
         deployed in production";
     }

     identity MODULE_CATEGORY_BASE {
       description
         "Base identity for the module category.  It is expected that
         publishing organizations will define additional derived
         identities to describe their categorization scheme.";
     }

     identity MODULE_SUBCATEGORY_BASE {
       description
         "Base identity for the module subcategory.  It is expected that
         publishing organizations will define additional derived
         identities to describe their categorization scheme.";
     }

     identity ORGANIZATION_TYPE {
       description
         "Publishing organization type for the set of modules";
     }

     identity STANDARDS {
       base ORGANIZATION_TYPE;
       description
         "Standards development organization (SDO) publisher type";
     }

     identity INDUSTRY {
       base ORGANIZATION_TYPE;
       description
         "Industry forum or other industry group";
     }

     identity COMMERCIAL {
       base ORGANIZATION_TYPE;
       description
         "Commercial entity, company, etc.";
     }

     identity INDIVIDUAL {
       base ORGANIZATION_TYPE;
       description
         "For modules published by an individual";
     }

     identity IETF_MODEL_LAYER {
       base MODULE_CATEGORY_BASE;
       description
         "Describes layering of models based on their abstraction
          level as defined by IETF model classification proposals";
       reference
         "IETF draft-ietf-netmod-yang-model-classification";
     }

     identity IETF_MODEL_TYPE {
       base MODULE_SUBCATEGORY_BASE;
       description
         "IETF proposed classification dimension of YANG model types as
```

```
          standard YANG models, vendor-specific, or user-specific YANG
          models and extensions";
         reference
           "IETF draft-ietf-netmod-yang-model-classification";
       }

       identity IETF_NETWORK_SERVICE {
         base IETF_MODEL_LAYER;
         description
           "Service-layer model as defined by IETF classification
           proposal";
       }

       identity IETF_NETWORK_ELEMENT {
         base IETF_MODEL_LAYER;
         description
           "Network element-layer model as defined by IETF classification
           proposal";
       }

       identity IETF_TYPE_STANDARD {
         base IETF_MODEL_TYPE;
         description
           "Models published by standards-defining organizations (SDOs)";
       }

       identity IETF_TYPE_VENDOR {
         base IETF_MODEL_TYPE;
         description
           "Developed by organizations (e.g., vendors) with the intent
           to support a specific set of implementations under control of
           that organization";
       }

       identity IETF_TYPE_USER {
         base IETF_MODEL_TYPE;
         description
           "Developed by organizations that operate YANG-based
           infrastructure including devices and orchestrators.
           The intent of these models is to express the specific needs
           for a certain implementation, above and beyond what is provided
           by vendors";
       }

       typedef module-version-type {
         type string;
         description
           "This type defines acceptable formats for the version of a
           module.  The version may be a semantic version, or a YANG
           revision statement date, and may include wildcards when
           included in a bundle compatibility list, e.g.:

           semver format: <major>.<minor>.<patch>
           examples: 0.1.0, 2.1.0, 1.1.*, 2.*.*

           revision format:  YYYY-MM-DD
           example:  2016-11-31";
       }



     }

     <CODE ENDS>


     <CODE BEGINS> file "openconfig-module-catalog.yang"

     module openconfig-module-catalog {

       yang-version "1";

       // namespace
       namespace "http://openconfig.net/yang/module-catalog";

       prefix "oc-cat";

       // import some basic types
       import openconfig-inet-types { prefix oc-inet; }
       import openconfig-catalog-types { prefix oc-cat-types; }
       import openconfig-extensions { prefix oc-ext; }


       // meta
       organization "OpenConfig working group";

       contact
         "OpenConfig working group
         www.openconfig.net";

       description
         "This module provides a schema for cataloging and descrbing
         YANG models published across various organizations.  The catalog
         contains several categories of data:

         * organizations -- entities that publish and/or maintain
```

```
           individual YANG modules or groups of modules

         * modules -- information regarding individual YANG modules,
           including their versions, dependencies, submodules, and how
           to access them

         * release bundles -- groups of modules that are compatible and
           consistent with each other (as determined by the publisher of
           of the bundle).  The release bundle does not necessarily
           correspond to a functional area, e.g., it could the entire
           set of modules published by an organization

         * feature bundles -- sets of schema paths across a
           release bundle that provide a specific set of functionality

         * implementations -- information about available module and/or
           bundle implementations and their status";

       oc-ext:openconfig-version "0.2.0";

       revision "2017-03-08" {
         description
           "OpenConfig public release";
         reference "0.2.0";
       }

       revision "2016-02-15" {
         description
           "Initial OpenConfig public release";
         reference "0.1.0";
       }

       revision "2015-10-18" {
         description
           "Initial revision";
         reference "TBD";
       }


       // grouping statements

       grouping catalog-module-common-config {
         description
           "Data definitions common for both bundles and standalone
           modules";

         leaf name {
           type string;
           description
             "The name of the module or bundle.  For modules, this
             should reflect the ’module’ or ’submodule’
             statement in the YANG module file.

             For bundles, this is the canonical name for the overall
             bundle of modules which is to be released together.
             This name should be consistent over multiple
             releases";
         }

         leaf version {
           type oc-cat-types:module-version-type;
           description
             "For individual modules, this is the version number, e.g.,
             a semantic version.  The version may be the same as the date
             indicated in the module revision statement.

             For bundles, this is a semantic version number for the
             overall bundle. This version is to be defined as per the
             approach specified in the OpenConfig semantic version
             guidance - and is of the form x.y.z, where x is the major
             version, y is the minor version, and z is the patch level";
           reference
             "Semantic versioning for OpenConfig models";
         }
       }

       grouping feature-bundle-included-reference {
         description
           "References to the included feature bundles";

         leaf name {
           type leafref {
             path "../../../../../../../organizations/" +
               "organization[name=current()/../publisher]/" +
                 "feature-bundles/feature-bundle/name";
           }
           description
             "Name of the referenced feature bundle";
         }

         leaf publisher {
           type leafref {
             path "../../../../../../../organizations/organization/" +
               "name";
           }
           description
```

```
            "Publisher of the referenced feature bundle";
        }

        leaf version {
          type oc-cat-types:module-version-type;
          description
            "Version of the referenced feature bundle";
        }
      }

      grouping catalog-implementation-bundle-config {
        description
          "References to the feature bundles supported by an
          implementation";

        uses feature-bundle-included-reference;
      }

      grouping catalog-implementation-bundle-top {
        description
          "Top-level grouping for the list of feature bundles
          supported by an implementation";

        container feature-bundles {
          description
            "Enclosing container for the list of feature bundles";

          list feature-bundle {
            key "name version";
            description
              "List of feature bundles supported by the implementation";

            uses catalog-implementation-bundle-config;
          }
        }
      }

      grouping catalog-implementation-config {
        description
          "Data describing any available implementations";

        leaf id {
          type string;
          description
            "An identifier for the implementation, provided by the
            implementor.  This id should uniquely identify a specific
            implementation of the module, e.g., based on the vendor,
            platform, and platform version.";
```

```
       }

       leaf description {
         type string;
         description
           "A text summary of important information about the
           implementation";
       }

       leaf reference {
         type union {
           type oc-inet:uri;
           type string;
         }
         description
           "A URI (preferred) or text reference to more detailed
           information about the implementation.";
       }


       leaf platform {
         type string;
         description
           "Name of the platform on which the implementation
           is available -- this could be the model name of a network
           device, a server OS, etc.";
       }

       leaf platform-version {
         type string;
         description
           "Implementor-defined version name or number of the
           module implementation, corresponding to the platform.
           This could be the firmware version of a network device
           such as a router, OS version, or other server platform
           version.";
       }

       leaf status {
         type identityref {
           base oc-cat-types:IMPLEMENTATION_STATUS_TYPE;
         }
         description
           "Indicates the status of the implementation, e.g.,
           complete, partial, in-progress, etc.  Implementors
           may define additional values for the base identity";
       }
     }
     grouping catalog-implementation-top {
       description
         "Top level grouping for information on model implementations";

       container implementations {
         description
           "Container for module implementation information";

         list implementation {
           key "id";
           description
             "List of available implementations, keyed by an identifier
             provided by either the implementor or the module
             maintainer.  Such a key avoids needing a complex composite
             key to uniquely identify an implementation.";

           uses catalog-implementation-config;
           uses catalog-implementation-bundle-top;
         }
       }
     }

     grouping catalog-module-dependency-config {
       description
         "Information about module dependencies";


       leaf-list required-module {
         type leafref {
           path "../../name";
         }
         description
           "List of references to modules that are imported by the
           current module.  This list should reflect all of the 'import'
           statements in the module.";
       }
     }

     grouping catalog-module-dependency-top {
       description
         "Top-level grouping for module dependency data";

       container dependencies {
         description
           "Data about dependencies of the module";

         uses catalog-module-dependency-config;
       }
     }

     grouping catalog-module-classification-config {
       description
         "Data describing the module’s classification(s)";

       leaf category {
         type identityref {
           base oc-cat-types:MODULE_CATEGORY_BASE;
         }
         description
            "Categorization of the module based on identities defined
            or used by the publishing organizations.";
       }

       leaf subcategory {
         type identityref {
           base oc-cat-types:MODULE_SUBCATEGORY_BASE;
         }
         description
            "Sub-categorization of the module based on identities
             defined or used by the publishing organizations.";
       }

       leaf deployment-status {
         type identityref {
           base oc-cat-types:MODULE_STATUS_TYPE;
         }
         description
           "Deployment status of the module -- experimental,
           standards-track, production, etc.";
       }
     }

     grouping catalog-module-classification-top {
       description
         "Data definitions related to module classfications";

       container classification {
         description
           "Container for data describing the module’s classification";

         uses catalog-module-classification-config;
       }
     }

     grouping catalog-module-access-config {
       description
```

```
         "Data pertaining to retrieval and usage of the module";

        leaf uri {
          type oc-inet:uri;
          description
            "URI where module can be downloaded.  Modules may be
            made available from the catalog maintainer, or directly
            from the publisher";
        }

        leaf md5-hash {
          type string;
          description
            "Optional MD5 hash of the module file.  If specified, the
            hash may be used by users to validate data integrity";
        }
      }

      grouping catalog-module-access-top {
        description
          "Top level groupig for data related to accessing a module
          or submodule";

        container access {
          description
            "Container for data pertaining to retrieval and usage of the
            module";

          uses catalog-module-access-config;
        }
      }

      grouping catalog-module-submodule-config {
        description
          "Data definitions for submodules belonging to a
          module";

        leaf name {
          type string;
          description
            "Name of the submodule as indicated by its top-level
            ’submodule’ statement";
        }

      }

      grouping catalog-module-submodule-top {
        description
```

```
         "Top-level grouping for submodule information";

       container submodules {
         description
           "Data for the submodules belonging to a submodule. If the
           module does not have any submodules, this container
           should be empty.";

         list submodule {
           key "name";
           description
             "List of submodules included by a module.  All submodules
             specified by ’include’ statements in the module should be
             included in this list.";

           uses catalog-module-submodule-config;
           uses catalog-module-access-top;
         }
       }
     }

     grouping catalog-module-base-config {
       description
         "Basic information describing the module, e.g., the
         YANG metadata in the module preface.";


       leaf namespace {
         type string;
         description
           "Published namespace of module, i.e., defined by the
           ’namespace’ ";
       }

       leaf prefix {
         type string;
         description
           "Published prefix of the module";
       }

       leaf revision {
         type string;
         description
           "Date in the revision statement of the module";
       }

       leaf summary {
         type string;
         description
           "Summary description of the module";
       }
     }

     grouping release-bundle-member-config {
       description
         "Data for each member of a bundle";

       leaf id {
         type string;
         description
           "Identifier for the bundle member";
       }

       leaf type {
         type identityref {
           base oc-cat-types:CATALOG_MEMBER_TYPE;
         }
         description
           "The type of member that is to be included within the
           release bundle. Release bundles may include modules and
           other release bundles.  Both member modules and member
           bundles should specify the list of compatible versions.";
       }

       leaf module {
         when "../type = 'oc-cat-types:MODULE'" {
           description
             "The module name is specified for bundle membrs that are
             modules";
         }
         type leafref {
           path "../../../../../../../organizations/" +
             "organization[name=current()/../publisher]/modules/" +
             "module/name";
         }
         description
           "Name of the module set which is included in this bundle -
           for example, 'openconfig-bgp'";
       }

       leaf release-bundle {
         when "../type = 'oc-cat-types:RELEASE_BUNDLE'" {
           description
             "The release bundle is specified for bundle members that
             are release bundles";
         }
         type leafref {
           path "../../../../../../../organizations/" +
             "organization[name=current()/../publisher]/" +
               "release-bundles/release-bundle/name";
         }
         description
           "Name of the module set which is included in this bundle -
           for example, ’openconfig-bgp’";
       }

       leaf publisher {
         type leafref {
           path "../../../../../../../organizations/organization/" +
             "name";
         }
         description
           "Reference to the name of the publishing organization";
       }

       leaf-list compatible-versions {
         type oc-cat-types:module-version-type;
         description
           "A list of semantic version specification of the versions
           of the specified module or release bundle which are
           compatible when building this version of the bundle.

           Version specifications may be added when changes are made
           to a module within a bundle, and this does not affect the
           interaction between it and other modules. It is expected
           that backwards compatible changes to an individual module or
           member bundle do not affect the compatibility of that
           with other members, and hence wildcard matches are allowed
           within this list.";
       }
     }

     grouping release-bundle-member-top {

       description
         "Parameters relating to models within release bundles";

       container members {
         description
           "List of bundle members which make up this release bundle. A
           member is defined as an individual YANG module specified
           in the YANG catalogue, or another release
           bundle which can be used to group multiple YANG
           models together.";
         list member {
           key "id";
           description
             "A set of modules or bundles which are part of the bundle
             of models. For example, if 'ietf-yang-types' were to be
             specified within the bundle, then this would refer to the
             individual entry within the module catalogue. If the type
             of the entry is set to bundle, then for example,
             openconfig-bgp could be referenced - which itself consists
             of separate modules.";

           uses release-bundle-member-config;

         }
       }
     }

     grouping release-bundle-top {
       description
         "Top-level container for a release bundle";

       container release-bundles {
         description
           "List of release bundles";

         list release-bundle {
           key "name version";

           description
             "List of release bundles - sets of modules and/or
             bundles which are interoperable";

           uses catalog-module-common-config;
           uses release-bundle-member-top;
         }
       }
     }

     grouping feature-bundle-release-config {
       description
         "Data definitions to identify the release bundle that the
         feature bundle is based on.";

       leaf name {
         type leafref {
           path "../../../../release-bundles/release-bundle/name";
         }
         description
```

```
           "Reference to the name of the release bundle used for the
           feature paths.";
       }

       leaf version {
         type leafref {
           path "../../../../release-bundles/" +
             "release-bundle[name=current()/../name]/version";
         }
         description
           "Reference to the release bundle version used for the
           feature paths";
       }

       leaf publisher {
         type leafref {
           path "../../../../release-bundles/" +
             "release-bundle[name=current()/../name]/publisher";
         }
         description
           "Reference to the publisher of the release bundle used for
           the feature paths";
       }
     }

     grouping feature-bundle-release-top {
       description
         "Top-level grouping for data about the release bundle used
         to specify the feature bundle";

       container release-bundle {
         description
           "Data to identify the release bundle from which the feature
           paths should be specified.  If the feature crosses
           release bundles, a new release bundle should be
           created to support the feature bundle.";

         leaf name {
           type leafref {
             path "../../../../../../organizations/" +
               "organization[name=current()/../publisher]/" +
                 "release-bundles/release-bundle/name";
           }
           description
             "Name of the module set which is included in this bundle -
             for example, ’openconfig-bgp’";
         }
         leaf publisher {
           type leafref {
             path "../../../../../../organizations/organization/" +
               "name";
           }
           description
             "Reference to the name of the publishing organization";
         }

         leaf version {
           type oc-cat-types:module-version-type;
           description
             "Version of the referenced release bundle";
         }
       }
     }


     grouping feature-bundle-config {
       description
         "Data definitions for the feature bundle";

       uses catalog-module-common-config;

       leaf-list path {
         type string;
         description
           "The list of schema paths included in the feature.  The
           paths specify subtrees, i.e., all data underneath the
           specified path are included in the feature.";
       }
     }

     grouping feature-bundle-feature-config {
       description
         "Data definitions for included feature bundles";

       uses feature-bundle-included-reference;
     }

     grouping feature-bundle-feature-top {
       description
         "Top level grouping for the list of included feature
         bundles";

       container feature-bundles {
         description
           "Enclosing container for the list of included feature
```

```
          bundles.  Feature bundles may be composed from other
          smaller feature units";

        list feature-bundle {
          key "name";
          description
            "The list of feature bundles included in the current
            feature bundle.";

          uses feature-bundle-feature-config;
        }
      }

    }


    grouping feature-bundle-top {
      description
        "Top-level grouping for OpenConfig feature bundles";

      container feature-bundles {
        description
          "Enclosing container for the list of feature bundles";

        list feature-bundle {
          key "name version";
          description
            "List of feature bundles";

          uses feature-bundle-config;
          uses feature-bundle-release-top;
          uses feature-bundle-feature-top;
        }
      }
    }

    grouping catalog-module-top {
      description
        "Top level structure of the module catalog";

      container modules {
        description
          "Modules published by this organization";

        list module {
          key "name version";
          description
            "List of published modules from the organization";
```

```
           uses catalog-module-common-config;
           uses catalog-module-base-config;
           uses catalog-module-classification-top;
           uses catalog-module-dependency-top;
           uses catalog-module-access-top;
           uses catalog-module-submodule-top;
         }
       }
     }

     grouping catalog-organization-config {
       description
         "Top level grouping for data related to an organization that
         publishes module, bundles, etc.";

       leaf name {
         type string;
         description
           "Name of the maintaining organization -- the name should be
           supplied in the official format used by the organization.
           Standards Body examples:
             IETF, IEEE, MEF, ONF, etc.
           Commercial entity examples:
             AT&T, Facebook, <Vendor>
           Name of industry forum examples:
             OpenConfig, OpenDaylight, ON.Lab";
       }

       leaf type {
         type identityref {
           base oc-cat-types:ORGANIZATION_TYPE;
         }
         description
           "Type of the publishing organization";
       }

       leaf contact {
         type string;
         description
           "Contact information for the publishing organization (web
           site, email address, etc.)";
       }
     }

     grouping catalog-organization-top {
       description
         "Top level grouping for list of maintaining organizations";
       container organizations {
         description
           "List of organizations owning modules";

         list organization {
           key "name";

           description
             "List of organizations publishing YANG modules or
             module bundles";

           uses catalog-organization-config;
           uses catalog-module-top;
           uses release-bundle-top;
           uses feature-bundle-top;
           uses catalog-implementation-top;
         }
       }
     }


     grouping catalog-top {
       description
         "Top-level grouping for the YANG model catalog";

       uses catalog-organization-top;
     }

     // data definition statements

     uses catalog-top;

   }

   <CODE ENDS>
```

Required extensions and types modules included below.

```
   <CODE BEGINS> file "openconfig-extensions.yang"

   module openconfig-extensions {

     yang-version "1";

     // namespace
     namespace "http://openconfig.net/yang/openconfig-ext";
     prefix "oc-ext";

     // meta
     organization "OpenConfig working group";

     contact
       "OpenConfig working group
       www.openconfig.net";

     description
       "This module provides extensions to the YANG language to allow
       OpenConfig specific functionality and meta-data to be defined.";

     revision "2017-01-29" {
       description
         "Added extension for annotating encrypted values.";
       reference "TBD";
     }

     revision "2015-10-09" {
       description
         "Initial OpenConfig public release";
       reference "TBD";
     }

     revision "2015-10-05" {
       description
         "Initial revision";
       reference "TBD";
     }

     // extension statements
     extension openconfig-version {
       argument "semver" {
         yin-element false;
       }
       description
         "The OpenConfig version number for the module. This is
         expressed as a semantic version number of the form:
           x.y.z
         where:
           * x corresponds to the major version,
           * y corresponds to a minor version,
           * z corresponds to a patch version.
         This version corresponds to the model file within which it is
         defined, and does not cover the whole set of OpenConfig models.
         Where several modules are used to build up a single block of
         functionality, the same module version is specified across each
         file that makes up the module.

         A major version number of 0 indicates that this model is still
         in development (whether within OpenConfig or with industry
         partners), and is potentially subject to change.

         Following a release of major version 1, all modules will
         increment major revision number where backwards incompatible
         changes to the model are made.

         The minor version is changed when features are added to the
         model that do not impact current clients use of the model.

         The patch-level version is incremented when non-feature changes
         (such as bugfixes or clarifications to human-readable
         descriptions that do not impact model functionality) are made
         that maintain backwards compatibility.

         The version number is stored in the module meta-data.";
     }

     extension openconfig-encrypted-value {
       description
         "This extension provides an annotation on schema nodes to
         indicate that the corresponding value should be stored and
         reported in encrypted form.

         Clients reading the configuration or applied configuration
         for the node should expect to receive only the encrypted value.
         This annotation may be used on nodes such as secure passwords
         in which the device never reports a cleartext value, even
         if the input is provided as cleartext.";
     }
   }
   <CODE ENDS>


  <CODE BEGINS> file "openconfig-inet-types.yang"

  module openconfig-inet-types {

    yang-version "1";
    namespace "http://openconfig.net/yang/types/inet";
    prefix "oc-inet";

    import openconfig-extensions { prefix "oc-ext"; }

    organization
```

```
      "OpenConfig working group";

    contact
      "OpenConfig working group
      www.openconfig.net";

    description
      "This module contains a set of Internet address related
      types for use in OpenConfig modules.";

    oc-ext:openconfig-version "0.1.0";

    revision 2017-01-26 {
      description
        "Initial module for inet types";
      reference "0.1.0";
    }

    // IPv4 and IPv6 types.

    typedef ipv4-address {
      type string {
        pattern '^(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|'        +
                '25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4]'  +
                '[0-9]|25[0-5])$';
      }
      description
        "An IPv4 address in dotted quad notation.";
    }

    typedef ipv6-address {
      type string {
          pattern
            // Must support compression through different lengths
            // therefore this regexp is complex.
            '^(([0-9a-fA-F]{1,4}:){7}[0-9a-fA-F]{1,4}|'         +
            '([0-9a-fA-F]{1,4}:){1,7}:|'                        +
            '([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}'         +
            '([0-9a-fA-F]{1,4}:){1,5}(:[0-9a-fA-F]{1,4}){1,2}|' +
            '([0-9a-fA-F]{1,4}:){1,4}(:[0-9a-fA-F]{1,4}){1,3}|' +
            '([0-9a-fA-F]{1,4}:){1,3}(:[0-9a-fA-F]{1,4}){1,4}|' +
            '([0-9a-fA-F]{1,4}:){1,2}(:[0-9a-fA-F]{1,4}){1,5}|' +
            '[0-9a-fA-F]{1,4}:((:[0-9a-fA-F]{1,4}){1,6})|'      +
            ':((:[0-9a-fA-F]{1,4}){1,7}|:)'                     +
            ')$';
      }
      description
        "An IPv6 address represented as either a full address; shortened
        or mixed-shortened formats.";
    }

    typedef ipv4-prefix {
      type string {
        pattern '^(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|'       +
                '25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4]' +
                '[0-9]|25[0-5])/(([0-9])|([1-2][0-9])|(3[0-2]))$';
      }
      description
        "An IPv4 prefix represented in dotted quad notation followed by
        a slash and a CIDR mask (0 <= mask <= 32).";
    }

    typedef ipv6-prefix {
      type string {
          pattern
            '^(([0-9a-fA-F]{1,4}:){7}[0-9a-fA-F]{1,4}|'         +
            '([0-9a-fA-F]{1,4}:){1,7}:|'                        +
            '([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}'         +
            '([0-9a-fA-F]{1,4}:){1,5}(:[0-9a-fA-F]{1,4}){1,2}|' +
            '([0-9a-fA-F]{1,4}:){1,4}(:[0-9a-fA-F]{1,4}){1,3}|' +
            '([0-9a-fA-F]{1,4}:){1,3}(:[0-9a-fA-F]{1,4}){1,4}|' +
            '([0-9a-fA-F]{1,4}:){1,2}(:[0-9a-fA-F]{1,4}){1,5}|' +
            '[0-9a-fA-F]{1,4}:((:[0-9a-fA-F]{1,4}){1,6})|'      +
            ':((:[0-9a-fA-F]{1,4}){1,7}|:)'                     +
            ')/(12[0-8]|1[0-1][0-9]|[1-9][0-9]|[0-9])$';
      }
      description
        "An IPv6 prefix represented in full, shortened, or mixed
        shortened format followed by a slash and CIDR mask (0 <= mask <=
        128).";
    }

    typedef ip-address {
      type union {
        type ipv4-address;
        type ipv6-address;
      }
      description
        "An IPv4 or IPv6 address with no prefix specified.";
    }

    typedef ip-prefix {
      type union {
        type ipv4-prefix;
        type ipv6-prefix;
      }
      description
        "An IPv4 or IPv6 prefix.";
    }

    typedef as-number {
      type uint32;
      description
        "A numeric identifier for an autonomous system (AS). An AS is a
        single domain, under common administrative control, which forms
        a unit of routing policy. Autonomous systems can be assigned a
        2-byte identifier, or a 4-byte identifier which may have public
        or private scope. Private ASNs are assigned from dedicated
        ranges. Public ASNs are assigned from ranges allocated by IANA
        to the regional internet registries (RIRs).";
      reference
        "RFC 1930 Guidelines for creation, selection, and registration
                  of an Autonomous System (AS)
         RFC 4271 A Border Gateway Protocol 4 (BGP-4)";
    }

    typedef dscp {
      type uint8 {
        range "0..63";
      }
      description
        "A differentiated services code point (DSCP) marking within the
        IP header.";
      reference
        "RFC 2474 Definition of the Differentiated Services Field
                   (DS Field) in the IPv4 and IPv6 Headers";
    }

    typedef ipv6-flow-label {
      type uint32 {
        range "0..1048575";
      }
      description
        "The IPv6 flow-label is a 20-bit value within the IPv6 header
        which is optionally used by the source of the IPv6 packet to
        label sets of packets for which special handling may be
        required.";
      reference
        "RFC 2460 Internet Protocol, Version 6 (IPv6) Specification";
    }

    typedef port-number {
      type uint16;
      description
```

```
        "A 16-bit port number used by a transport protocol such as TCP
        or UDP.";
      reference
        "RFC 768 User Datagram Protocol
         RFC 793 Transmission Control Protocol";
    }

    typedef uri {
      type string;
      description
        "An ASCII-encoded Uniform Resource Identifier (URI) as defined
        in RFC 3986.";
      reference
        "RFC 3986 Uniform Resource Identifier (URI): Generic Syntax";
    }
  }

  <CODE ENDS>
```

10. References

10.1. Normative references

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <http://www.rfc-editor.org/info/rfc7950>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <http://www.rfc-editor.org/info/rfc7895>.

10.2. Informative references

[RTG-AD-YANG]
Wu, Q. and D. Sinicrope, "Routing Area Yang Coordinator’s Summary Page", October 2015, <http://trac.tools.ietf.org/area/rtg/trac/wiki/RtgYangCoordSummary>.

[OC-SEMVER]
OpenConfig operator working group, "Semantic Versioning for OpenConfig models", September 2015, <http://www.openconfig.net/documentation/semantic-versioning/>.

[I-D.openconfig-netmod-model-structure]
Shaikh, A., Shakir, R., D’Souza, K., and L. Fang, "Operational Structure and Organization of YANG Models", draft-openconfig-netmod-model-structure-00 (work in progress), March 2015.

[I-D.rtgyangdt-rtgwg-device-model]
Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps, "Network Device YANG Organizational Models", draft-rtgyangdt-rtgwg-device-model-05 (work in progress), August 2016.

[I-D.ietf-netmod-yang-model-classification]
Bogdanovic, D., Claise, B., and C. Moberg, "YANG Module Classification", draft-ietf-netmod-yang-model-classification-04 (work in progress), October 2016.

Appendix A. Change summary

A.1. Changes between revisions -01 and -02

- Included additional explanation for release and feature bundles
- Changed feature bundles to be based on schema paths
- Included version 0.2.0 of catalog modules.

A.2. Changes between revisions -00 and -01

- Added release bundle definitions.
- Added IETF module classification identities based on draft-ietf-netmod-yang-model-classification.

Authors’ Addresses

Anees Shaikh
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: aashaikh@google.com

Rob Shakir
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: rjs@rob.sh

Kevin D’Souza
AT&T
200 S. Laurel Ave
Middletown, NJ
US

Email: kd6913@att.com