Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 5, 2018

J. Clarke
B. Claise
Cisco Systems, Inc.

April 3, 2018

YANG module for yangcatalog.org
draft-clacla-netmod-model-catalog-03

Abstract

This document specifies a YANG module that contains metadata related to YANG modules and vendor implementations of those YANG modules.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on October 5, 2018.

Table of Contents

1. Introduction

YANG [RFC6020] [RFC7950] became the standard data modeling language of choice. Not only is it used by the IETF for specifying models, but also in many Standard Development Organizations (SDOs), consortia, and open-source projects: the IEEE, the Broadband Forum (BFF), DMTF, MEF, ITU, OpenDaylight, Open ROADM, Openconfig, sysrepo, and more.

With the rise of data model-driven management and the success of YANG as a key piece comes a challenge: the entire industry develops YANG models. In order for operators to automate coherent services, the industry must ensure the following:

1. Data models must work together

2. There exists a toolchain to help one search and understand models

3. Metadata is present to further describe model attributes

The site <https://www.yangcatalog.org> (and the YANG catalog that it provides) is an attempt to address these key tenants. From a high level point of view, the goal of this catalog is to become a reference for all YANG modules available in the industry, for both YANG developers (to search on what exists already) and for operators (to discover the more mature YANG models to automate services). This YANG catalog should not only contain pointers to the YANG modules themselves, but also contain metadata related to those YANG modules: What is the module type (service model or not?); what is the maturity level? (e.g., for the IETF: is this an RFC, a working group document or an individual draft?); is this module implemented?; who is the contact?; is there open-source code available? And we expect many more in the future. The industry has begun to understand that the metadata related to YANG models become equally important as the YANG models themselves.

This document defines a YANG [RFC7950] module called yang-catalog.yang that contains the metadata definitions that are complementary to the related YANG modules themselves. The design for this module is based on experience and real code. As such, it's expected that this YANG module will be a living document. Furthermore, new use cases, which require new metadata in this YANG module, are discovered on a regular basis.

The yangcatalog.org instantiation of the catalog provides a means for module authors and vendors implementing modules to upload their metadata, which is then searchable via an API, as well as using a variety of web-based tools. The instructions for contributing and searching for metadata can be found at <https://www.yangcatalog.org/contribe.php>.

1.1. Status of Work and Open Issues

The top open issues are:

1. Obtain feedback from vendors and SDOs

2. Socialize module at the IETF and incorporate feedback

3. Provide module bundle support

2. Learning from Experience

While implementing the catalog and tools at yangcatalog.org, we initially looked at the "Catalog and registry for YANG models" [I-D.openconfig-netmod-model-catalog] as a starting point but we

quickly realized that the objectives are different. As a consequence, even if some of the information is similar, this YANG module started to diverge. Below are the justifications for the divergence, our observations, and our learning experience as we have been developing and getting feedback.

2.1. YANG Module Library

In order for the YANG catalog to become a complete inventory of which models are supported on the different platforms, content such as the support of the YANG module/deviation/feature/etc. should be easy to import and update. An easy way to populate this information is to have a similar structure as the YANG Module Library [RFC7895]. That way, querying the YANG Module Library from a platform provides, directly in the right format, the input for the YANG catalog inventory.

There are some similar entries between the YANG Module Library and the Openconfig catalog. For example, the Openconfig catalog model defines a "uri" leaf which is similar to "schema" from [RFC7895]). And this adds to the overall confusion.

2.2. YANG Catalog Data Model

The structure of the yang-catalog.yang module described in this document is found below. The meaning of the symbols in this and seubsequent tree diagrams in this document is explained in [I-D.ietf-netmod-yang-tree-diagrams]:

```
module: yang-catalog
    +--rw catalog
       +--rw modules
       |  +--rw module* [name revision organization]
       |     +--rw name                        yang:yang-identifier
       |     +--rw revision                    union
       |     +--rw organization                string
       |     +--rw ietf
       |     |  +--rw ietf-wg?   string
       |     +--rw namespace                   inet:uri
       |     +--rw schema?                     inet:uri
       |     +--rw generated-from?             enumeration
       |     +--rw maturity-level?             enumeration
       |     +--rw document-name?              string
       |     +--rw author-email?               yc:email-address
       |     +--rw reference?                  inet:uri
       |     +--rw module-classification       enumeration
       |     +--rw compilation-status?         enumeration
       |     +--rw compilation-result?         inet:uri
```

```
       |     +--rw prefix?                    string
       |     +--rw yang-version?              enumeration
       |     +--rw description?               string
       |     +--rw contact?                   string
       |     +--rw module-type?               enumeration
       |     +--rw belongs-to?                yang:yang-identifier
       |     +--rw tree-type?                 enumeration
       |     +--rw yang-tree?                 inet:uri
       |     +--rw expires?                   yang:date-and-time
       |     +--rw expired?                   union
       |     +--rw submodule* [name revision]
       |     |  +--rw name        yang:yang-identifier
       |     |  +--rw revision    union
       |     |  +--rw schema?     inet:uri
       |     +--rw dependencies* [name]
       |     |  +--rw name        yang:yang-identifier
       |     |  +--rw revision?   union
       |     |  +--rw schema?     inet:uri
       |     +--rw dependents* [name]
       |     |  +--rw name        yang:yang-identifier
       |     |  +--rw revision?   union
       |     |  +--rw schema?     inet:uri
       |     +--rw semantic-version?          yc:semver
       |     +--rw derived-semantic-version?  yc:semver
       |     +--rw implementations
       |        +--rw implementation* [vendor platform software-version
software-flavor]
       |           +--rw vendor             string
       |           +--rw platform           string
       |           +--rw software-version   string
       |           +--rw software-flavor    string
       |           +--rw os-version?        string
       |           +--rw feature-set?       string
       |           +--rw os-type?           string
       |           +--rw feature*           yang:yang-identifier
       |           +--rw deviation* [name revision]
       |           |  +--rw name        yang:yang-identifier
       |           |  +--rw revision    union
       |           +--rw conformance-type?  enumeration
      +--rw vendors
         +--rw vendor* [name]
            +--rw name         string
            +--rw platforms
               +--rw platform* [name]
                  +--rw name                string
                  +--rw software-versions
                     +--rw software-version* [name]
                        +--rw name               string
```

```
                    +--rw software-flavors
```

```
                          +--rw software-flavor* [name]
                             +--rw name         string
                             +--rw protocols
                             |  +--rw protocol* [name]
                             |     +--rw name                identityref
                             |     +--rw protocol-version*   string
                             |     +--rw capabilities*       string
                             +--rw modules
                                +--rw module* [name revision organization]
                                   +--rw name                -> /catalog/
modules/module/name
                                   +--rw revision            -> deref(../
name)/../revision
                                   +--rw organization        -> deref(../
revision)/../organization
                                   +--rw os-version?         string
                                   +--rw feature-set?        string
                                   +--rw os-type?            string
                                   +--rw feature*            yang:yang-
identifier
                                   +--rw deviation* [name revision]
                                   |  +--rw name        yang:yang-identifier
                                   |  +--rw revision    union
                                   +--rw conformance-type?   enumeration
```

Various elements of this module tree will be discussed in the subsequent sections.

2.3. Module Sub-Tree

Each module in the YANG Catalog is enumerated by its metadata and by various vendor implementations. While initially each module used the "module-list" grouping from the YANG Library [RFC7895], it was found that some of the nodes within that grouping such as "conformance-type", "feature", and "deviation" are only valid when a module is implemented by a server. As pure YANG data (which the Catalog is) it is not possible to provide meaningful values for those nodes. As such, common leafs were extracted from the YANG Library's "module-list" for use in the module sub-tree of yang-catalog. Those server-specific nodes are moved under the implementation sub-tree. The yang-catalog module then augments these common nodes to add metadata elements that aid module developers and module consumers alike in understanding the relative maturity, compilation status, and the support contact(s) of each YANG module.

```
  +--rw modules
  |  +--rw module* [name revision organization]
  |     +--rw name                    yang:yang-identifier
```

```
|     +--rw revision                union
|     +--rw organization            string
|     +--rw ietf
|     |  +--rw ietf-wg?   string
```

```
|     +--rw namespace               inet:uri
|     +--rw schema?                 inet:uri
|     +--rw generated-from?         enumeration
|     +--rw maturity-level?         enumeration
|     +--rw document-name?          string
|     +--rw author-email?           yc:email-address
|     +--rw reference?              inet:uri
|     +--rw module-classification   enumeration
|     +--rw compilation-status?     enumeration
|     +--rw compilation-result?     inet:uri
|     +--rw prefix?                 string
|     +--rw yang-version?           enumeration
|     +--rw description?            string
|     +--rw contact?                string
|     +--rw module-type?            enumeration
|     +--rw belongs-to?             yang:yang-identifier
|     +--rw tree-type?              enumeration
|     +--rw yang-tree?              inet:uri
|     +--rw expires?                yang:date-and-time
|     +--rw expired?                union
|     +--rw submodule* [name revision]
|     |  +--rw name        yang:yang-identifier
|     |  +--rw revision    union
|     |  +--rw schema?     inet:uri
|     +--rw dependencies* [name]
|     |  +--rw name        yang:yang-identifier
|     |  +--rw revision?   union
|     |  +--rw schema?     inet:uri
|     +--rw dependents* [name]
|     |  +--rw name        yang:yang-identifier
|     |  +--rw revision?   union
|     |  +--rw schema?     inet:uri
|     +--rw implementations
|        +--rw implementation*
|                [vendor platform software-version software-flavor]
|           +--rw vendor              string
|           +--rw platform            string
|           +--rw software-version    string
|           +--rw software-flavor     string
|           +--rw os-version?         string
|           +--rw feature-set?        string
|           +--rw os-type?            string
|           +--rw feature*            yang:yang-identifier
|           +--rw deviation* [name revision]
|           |  +--rw name        yang:yang-identifier
|           |  +--rw revision    union
|           +--rw conformance-type?   enumeration
```

Many of these additional metadata fields are self-explanatory, especially given their descriptions in the module itself and the fact that many elements translate directly to YANG schema elements. However, those requiring additional explanation or context as to why they are needed are described in the subsequent sections.

2.4. Compilation Information

For the inventory to be complete, YANG modules at different stages of their lifecyle should be taken into account, including YANG modules that are clearly works-in-progress (i.e., that do not validate correctly either because of faulty YANG constructs, because of a faulty imported YANG module, or simply because of warnings). The results of compilation testing are denoted in the "compilation-status" leaf with links to the output of the tests stored in the "compilation-result" leaf. Note that some warnings seen in "compilation-result" are not always show-stoppers from a code generation point of view (see the Generated From section). Nonetheless, the compilation or validation status, along with the compilation output, provide a clear indication of a given YANG module's development phase and stability. The current set of validator is pyang, confdc, yangdump-pro, and yanglint.

```
leaf compilation-status {
type enumeration {
enum passed {
  description
    "All compilers were able to compile this YANG module without
     any errors or warnings.";
}
enum passed-with-warnings {
  description
    "All compilers were able to compile this YANG module without
     any errors, but at least one of them caught a warning.";
}
enum failed {
  description
    "At least one of compilers found an error while
     compiling this YANG module.";
}
enum pending {
  description
    "The module was just added to the catalog and compilation testing is still
     in progress.";
}
enum unknown {
  description
    "There is not sufficient information about compilation status.  This Could
     mean compilation crashed causing it not to complete fully.";
}
}
description
"Status of the module, whether it was possible to compile this YANG module or
 there are still some errors/warnings.";
}
leaf compilation-result {
type string;
description
"Result of the compilation explaining specifically what error or warning
occurred.
 This is not existing if compilation status is PASSED.";
}
```

The current instantiation of the YANG Catalog at <https://www.yangcatalog.org> uses a number of different YANG compilers for testing. The wrapper that handles validation attempts to use metadata from the catalog to determine which tests to perform on a given module. For example, if the module is authored by the IETF, IETF-specific tests will be conducted to provide the most accurate and complete set of tests possible.

2.5. Maturity Level

Models also have inherent maturity levels from their respective Standards Development Organizations (SDOs). These maturity levels help module consumers understand how complete, tested, etc. a module is.

```
leaf maturity-level {
type enumeration {
enum ratified {
  description
    "Maturity of a module that is fully approved (e.g., a standard).";
}
enum adopted {
  description
    "Maturity of a module that is actively being developed by a organization
towards ratification.";
}
enum initial {
  description
    "Maturity of a module that has been initially created, but has no official
     organization-level status.";
}
enum not-applicable {
  description
    "The maturity level is not used for vendor-supplied models, and thus all
vendor
     modules will have a maturity of not-applicable";
}
}
description
"The current maturity of the module with respect to the body that created it.
 This allows one to understand where the module is in its overall life cycle.";
}
```

This enumeration mapping has been implemented for the YANG modules from IETF and BBF. The "maturity-level" MUST be "not-applicable" for all vendor-authored modules.

In addition to a module's maturity, modules that are part of works-in-progress (e.g., IETF internet drafts) may expire if work ceases on the related document. To track that, the catalog has two module leafs: "expires" and "expired". The "expires" leaf indicates a date and time when the module is expected to expire whereas the "expired" leaf indicates whether or not the module has already expired. For those modules that will never expire, the "expired" leaf MUST be set to "not-applicable".

2.6. Generated From

While many models are written by hand (i.e., authored by humans) others are generated from things such as vendor code or CLI constructs or from SMI-based MIB modules. These "generated" modules do not necessarily require the same stringent validity checking that hand-written modules require. As such, these modules have a generated-from value that is designed to inform validators how much checking to do.

```
     leaf generated-from {
       type enumeration {
         enum "mib" {
           description
             "Module generated from Structure of Management Information
(SMI)
              MIB per RFC6643.";
         }
         enum "not-applicable" {
           description
             "Module was not generated but it was authored manually.";
         }
         enum "native" {
           description
             "Module generated from platform internal,
              proprietary structure, or code.";
         }
       }
       default "not-applicable";
       description
         "This statement defines weather the module was generated or not.
          Default value is set to not-applicable, which means that module
          was created manualy and not generated.";
     }
```

2.7. Implementation

As of version 02 of openconfig-model-catalog.yang [I-D.openconfig-netmod-model-catalog] it is not possible to identify the implementations of one specific module. Instead modules are grouped into feature-bundle, and feature-bundles are implemented by devices. Because of this, we added our own implementation sub-tree under each module to yang-catalog.yang. Our implementation sub-tree is:

```
+--rw implementation* [vendor platform software-version software-flavor]
 +--rw vendor              string
 +--rw platform            string
 +--rw software-version    string
 +--rw software-flavor     string
 +--rw os-version?         string
 +--rw feature-set?        string
 +--rw os-type?            string
 +--rw feature*            yang:yang-identifier
 +--rw deviation* [name revision]
 |  +--rw name        yang:yang-identifier
 |  +--rw revision    union
 +--rw conformance-type?   enumeration
```

The keys in this sub-tree can be used in the "vendor" sub-tree defined below to walk through each vendor, platform, and software release to get a full list of supported YANG modules for that release.

The "software-flavor" key leaf identifies a variation of a specific version where YANG model support may be different. Depending on the vendor, this could be a license, additional software component, or a feature set.

The other non-key leaves in the implementation sub-tree represent optional elements of a software release that some vendors may choose to use for informational purposes. These leafs are duplicated under the vendor sub-tree.

2.8. Vendor Sub-Tree

The vendor sub-tree provides a way, especially for module consumers, to walk through a specific device and software release to find a list of modules supported therein. This sub-tree turns the "implementation" sub-tree on its head to provide an optimized index for one wanting to go from a platform to a full list of modules.

In addition to the module list, the vendor sub-tree lists the YANG-based protocols (e.g., NETCONF or RESTCONF) that the platforms support.

```
   +--rw vendors
      +--rw vendor* [name]
         +--rw name         string
         +--rw platforms
            +--rw platform* [name]
               +--rw name                string
               +--rw software-versions
                  +--rw software-version* [name]
                     +--rw name               string
                     +--rw software-flavors
                        +--rw software-flavor* [name]
                           +--rw name         string
                           +--rw protocols
                           |  +--rw protocol* [name]
                           |     +--rw name                identityref
                           |     +--rw protocol-version*   string
                           |     +--rw capabilities*       string
                           +--rw modules
                              +--rw module*
                                    [name revision organization]
                              +--rw name                leafref
                              +--rw revision            leafref
                              +--rw organization        leafref
                              +--rw os-version?         string
                              +--rw feature-set?        string
                              +--rw os-type?            string
                              +--rw feature*
                              |       yang:yang-identifier
                              +--rw deviation* [name revision]
                              |  +--rw name
                              |  |       yang:yang-identifier
                              |  +--rw revision    union
                              +--rw conformance-type?   enumeration
```

This sub-tree structure also enables one to look for YANG modules for a class of platforms (e.g., list of modules for Cisco, or list of modules for Cisco ASR9K routers) instead of only being able to look for YANG modules for a specific platform and software release.

2.9. Regex Expression Differences

Another challenge encountered when trying to using [I-D.openconfig-netmod-model-catalog] as the canonical catalog is the regular expression syntax it uses. The Openconfig module uses a POSIX-compliant regular expression syntax whereas YANG-based protocol implementations like ConfD [1] expect the IETF-chosen W3C syntax. In order to load the Openconfig catalog in such engines, changes to the

regular expression syntax had to be done, and these one-off changes are not supportable.

3. YANG Catalog Use Cases

The YANG Catalog module is currently targeted to address the following use cases.

3.1. YANG Search Metadata

The yangcatalog.org toolchain provides a service for searching [2] for YANG modules based on keywords. The resulting search data currently stores the module and node metadata in a proprietary format along with the search index data. By populating the yang-catalog module, this search service can instead pull the metadata from the implementation of the module. Populating this instance of the yang-catalog module will be using an API that is still under development, but will ultimately allow SDOs and vendors to provide metadata and ensure the search service has the most up-to-date data for all available modules.

3.2. Identify YANG Module Support in Devices

By organizing the yang-catalog module so that one can either find all implementations for a given module, or find all modules supported by a vendor platform and software release, the catalog will provide a straight-forward way for one to understand the extent of YANG module support in participating vendors' software releases. Eventually a web-based graphical interface will be connected to this on yangcatalog.org to make it easier for consumers to leverage the instance of the yang-catalog module for this use case.

3.3. Identify The Backward Compatibility between YANG Module Revisions

The YANG catalog contains not only the most up-to-date YANG module revision of a given module, but keeps all previous revisions as well. With APIs in mind, it's important to understand whether different YANG module revisions are backward compatible (this is specifically imported for native YANG modules, i.e. the ones where generated-from = native). This document uses the following semver.org semantic [semver] to compare the YANG module backwards (in)compatibility:

MAJOR is incremented when the new version of the specification is incompatible with previous versions.

MINOR is incremented when new functionality is added in a manner that is backward-compatible with previous versions.

PATCH is incremented when bug fixes are made in a backward-compatible manner.

Two distinct leaves in the YANG module contains this semver semantic:

the semantic-version leaf contains the value reported as metadata by a specific YANG module.

the derived-semantic-version leaf is established by examining the the YANG module themselves. As such, only the YANG syntax, as opposed to the implementation changes that lead some some semantic changes.

Typically, an Openconfig YANG module would contain an extension, which is mapped to the semantic-version leaf.

```
    // 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.";
    }
```

Note that the absolute numbers in the semantic-version and derived-semantic-version are actually meaningless: the difference between two YANG module semver fields should be looked at.

In addition to the semantic versions, the yang-tree field points to the respective module's simplified graphical representation of its model as described by [I-D.ietf-netmod-yang-tree-diagrams]. This diagram can be compared between two revisions of the same module to visually determine any structural differences when MAJOR or MINOR semantic versions differ.

4. YANG Catalog YANG module

The structure of the model defined in this document is described by the YANG module below.

```
<CODE BEGINS> file "yang-catalog@2018-04-03.yang"
  module yang-catalog {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:yang-catalog";
    prefix yc;

    import ietf-yang-types {
      prefix yang;
    }
    import ietf-yang-library {
      prefix yanglib;
    }
    import ietf-inet-types {
      prefix inet;
    }

    organization
      "yangcatalog.org";
    contact
      "Benoit Claise <bclaise@cisco.com>

       Joe Clarke <jclarke@cisco.com>";
    description
      "This module contains metadata pertinent to each YANG module, as
       well as a list of vendor implementations for each module.  The
       structure is laid out in such a way as to make it possible to
       locate metadata and vendor implementation on a per-module basis
       as well as obtain a list of available modules for a given
       vendor's platform and specific software release.";

    revision 2018-04-03 {
      description
        "Bump the YANG version number to 1.1 for the deref XPath
         function.";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2018-01-23 {
      description
        "* Add leafs to track expire modules
         * Correct a bug with leafref dereferencing";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-09-26 {
```

```
      description
        "* Add leafs for tracking dependencies and dependents
         * Simplify the generated-from enumerated values
         * Refine the type for compilation-result to be an inet:uri
         * Add leafs for semantic versioning";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-08-18 {
      description
        "* Reorder organization to be with the other module keys
         * Add a belongs-to leaf to track a submodule's parent";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-07-28 {
      description
        "* Revert config false nodes as we need to be able to set these via
<edit-config>

         * Make conformance-type optional as not all vendors implement yang-
library

         * Re-add the path typedef";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-07-26 {
      description
        "A number of improvements based on YANG Doctor review:

         * Remove references to 'server' in leafs describing YANG data
         * Fold the augmentation module leafs directly under /catalog/modules/
module
         * Use identities for protocols instead of an emumeration
         * Make some extractable fields 'config false'
         * Fix various types
         * Normalize enums to be lowercase
         * Add a leaf for module-classification
         * Change yang-version to be an enum
         * Add module conformance, deviation and feature leafs under the
implementation branches";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-07-14 {
      description
        "Modularize some of the leafs and create typedefs so they
         can be shared between the API input modules.";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-07-03 {
```

```
      description
        "Initial revision.";
      reference
        "
```

```
         YANG Catalog <https://yangcatalog.org>";
    }

    /*
     * Identities
     */

    identity protocol {
      description
        "Abstract base identity for a YANG-based protocol.";
    }

    identity netconf {
      base protocol;
      description
        "Protocol identity for NETCONF as described in RFC 6241.";
    }

    identity restconf {
      base protocol;
      description
        "Protocol identity for RESTCONF as described in RFC 8040.";
    }

    typedef email-address {
      type string {
        pattern "[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+([.][a-zA-Z0-9-]
+)*";
      }
      description
        "This type represents a string with an email address.";
    }

    /*
     * Typedefs
     */

    typedef path {
      type string {
        pattern '([A-Za-z]:|[\w-]+(\.[\w-]+)*)?(([/\\][\w@.-]+)+)';
      }
      description
        "This type represents a string with path to the file.";
    }

    typedef semver {
      type string {
        pattern '[0-9]+\.[0-9]+\.[0-9]+';
      }
```

```
      description
        "A semantic version in the format of x.y.z, where:

         x = the major version number
         y = the minor version number
         z = the patch version number

         Changes to the major version number denote backwards-incompatible
         changes between two revisions of the same module.

         Changes to the minor version number indicate there have been new
         backwards-compatible features introduced in the later version of
         a module.

         Changes to the patch version indicate bug fixes between two
         versions of a module.";
      reference "Semantic Versioning 2.0.0 <http://semver.org/>";
    }

    container catalog {
      description
        "Root container of yang-catalog holding two main branches -
         modules and vendors. The modules sub-tree contains all the modules in
         the catalog and all of their metadata with their implementations.
         The vendor sub-tree holds modules for specific vendors, platforms,
         software-versions, and software-flavors. It contains reference to a
         name and revision of the module in order to reference the module's
full
         set of metadata.";
      container modules {
        description
          "Container holding the list of modules";
        list module {
          key "name revision organization";
          description
            "Each entry represents one revision of one module
             for one organization.";
          uses yang-lib-common-leafs;
          leaf organization {
            type string;
            description
              "This statement defines the party responsible for this
               module.  The argument is a string that is used to specify a
textual
               description of the organization(s) under whose auspices this
module
               was developed.";
          }
```

```
          uses organization-specific-metadata;
          leaf namespace {
            type inet:uri;
```

```
            mandatory true;
            description
              "The XML namespace identifier for this module.";
          }
          uses yang-lib-schema-leaf;
          uses catalog-module-metadata;
          list submodule {
            key "name revision";
            description
              "Each entry represents one submodule within the
               parent module.";
            uses yang-lib-common-leafs;
            uses yang-lib-schema-leaf;
          }
          list dependencies {
            key "name";
            description
              "Each entry represents one dependency.";
            uses yang-lib-common-leafs;
            uses yang-lib-schema-leaf;
          }
          list dependents {
            key "name";
            description
              "Each entry represents one dependent.";
            uses yang-lib-common-leafs;
            uses yang-lib-schema-leaf;
          }
          leaf semantic-version {
            type yc:semver;
            description
              "The formal semantic version of a module as provided by the
module
               itself.  If the module does not provide a semantic version, this
leaf
               will not be specified.";
          }
          leaf derived-semantic-version {
            type yc:semver;
            description
              "The semantic version of a module as compared to other revisions
of
               the same module.  This value is computed algorithmically by
ordering
               all revisions of a given module and comparing them to look for
backwards
               incompatible changes.";
          }
```

```
          container implementations {
            description
              "Container holding lists of per-module implementation details.";
            list implementation {
              key "vendor platform software-version software-flavor";
```

```
              description
                "List of module implementations.";
              leaf vendor {
                type string;
                description
                  "Organization that implements this module.";
              }
              leaf platform {
                type string;
                description
                  "Platform on which this module is implemented.";
              }
              leaf software-version {
                type string;
                description
                  "Name of the version of software.  With respect to most
network device appliances,
                   this will be the operating system version.  But for other
YANG module
                   implementation, this would be a version of appliance
software.  Ultimately,
                   this should correspond to a version string that will be
recognizable by
                   the consumers of the platform.";
              }
              leaf software-flavor {
                type string;
                description
                  "A variation of a specific version where
                   YANG model support may be different.  Depending on the
vendor, this could
                   be a license, additional software component, or a feature
set.";
              }
              uses shared-implementation-leafs;
              uses yang-lib-implementation-leafs;
            }
          }
        }
      }
      container vendors {
        description
          "Container holding lists of organizations that publish YANG
modules.";
        list vendor {
          key "name";
          description
            "List of organizations publishing YANG modules.";
```

```
          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";
          }
          container platforms {
            description
              "Container holding list of platforms.";
            list platform {
              key "name";
              description
                "List of platforms under specific vendor";
              leaf name {
                type string;
                description
                  "Name of the platform";
              }
              container software-versions {
                description
                  "Container holding list of versions of software versions.";
                list software-version {
                  key "name";
                  description
                    "List of version of software versions under specific
vendor, platform.";
                  leaf name {
                    type string;
                    description
                      "Name of the version of software.  With respect to most
network device appliances,
                       this will be the operating system version.  But for
other YANG module
                       implementation, this would be a version of appliance
software.  Ultimately,
                       this should correspond to a version string that will be
recognizable by
                       the consumers of the platform.";
                  }
                  container software-flavors {
                    description
                      "Container holding list of software flavors.";
                    list software-flavor {
                      key "name";
                      description
                        "List of software flavors under specific vendor,
platform, software-version.";
                      leaf name {
                        type string;
```

```
                        description
                          "A variation of a specific version where
                           YANG model support may be different.  Depending on
the vendor, this could
                           be a license, additional software component, or a
feature set.";
                      }
                      container protocols {
```

```
                        description
                          "List of the protocols";
                        list protocol {
                          key "name";
                          description
                            "YANG-based protocol that is used on the device.
New identities
                             are expected to be added to address other YANG-
based protocols.";
                          leaf name {
                            type identityref {
                              base yc:protocol;
                            }
                            description
                              "Identity of the YANG-based protocol that is
supported.";
                          }
                          leaf-list protocol-version {
                            type string;
                            description
                              "Version of the specific protocol.";
                          }
                          leaf-list capabilities {
                            type string;
                            description
                              "Listed name of capabilities that are
                               supported by the specific device.";
                          }
                        }
                      }
                      container modules {
                        description
                          "Container holding list of modules.";
                        list module {
                          key "name revision organization";
                          description
                            "List of references to YANG modules under specific
vendor, platform, software-version,
                             software-flavor.  Using these references, the
complete set of metadata can be
                             retrieved for each module.";
                          leaf name {
                            type leafref {
                              path "/catalog/modules/module/name";
                            }
                            description
                              "Reference to a name of the module that is
contained in specific vendor, platform,
```

```
      software-version, software-flavor.";
  }
  leaf revision {
    type leafref {
      path "deref(../name)/../revision";
    }
```

```
                            description
                              "Reference to a revision of the module that is
contained in specific vendor,
                               platform, software-version, software-flavor.";
                          }
                          leaf organization {
                            type leafref {
                              path "deref(../revision)/../organization";
                            }
                            description
                              "Reference to the authoring organization of the
module for the implemented
                                module.";
                          }
                          uses shared-implementation-leafs;
                          uses yang-lib-implementation-leafs;
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }

    grouping catalog-module-metadata {
      uses shared-module-leafs;
      leaf compilation-status {
        type enumeration {
          enum passed {
            description
              "All compilers were able to compile this YANG module without
               any errors or warnings.";
          }
          enum passed-with-warnings {
            description
              "All compilers were able to compile this YANG module without
               any errors, but at least one of them caught a warning.";
          }
          enum failed {
            description
              "At least one of compilers found an error while
               compiling this YANG module.";
          }
          enum pending {
```

```
            description
              "The module was just added to the catalog and compilation testing
is still
```

```
               in progress.";
          }
          enum unknown {
            description
              "There is not sufficient information about compilation status.
This Could
               mean compilation crashed causing it not to complete fully.";
          }
        }
        description
          "Status of the module, whether it was possible to compile this YANG
module or
           there are still some errors/warnings.";
      }
      leaf compilation-result {
        type inet:uri;
        description
          "Link to the result of the compilation explaining specifically what
error or
           warning occurred.  This is not existing if compilation status is
PASSED.";
      }
      leaf prefix {
        type string;
        description
          "Statement of yang that is used to define the prefix associated with
           the module and its namespace. The prefix statement's argument is
           the prefix string that is used as a prefix to access a module. The
           prefix string MAY be used to refer to definitions contained in the
           module, e.g., if:ifName.";
      }
      leaf yang-version {
        type enumeration {
          enum 1.0 {
            description
              "YANG version 1.0 as defined in RFC 6020.";
          }
          enum 1.1 {
            description
              "YANG version 1.1 as defined in RFC 7950.";
          }
        }
        description
          "The optional yang-version statement specifies which version of the
           YANG language was used in developing the module.";
      }
      leaf description {
        type string;
```

```
        description
          "This statement takes as an argument a string that
           contains a human-readable textual description of this definition.
           The text is provided in a language (or languages) chosen by the
```

```
           module developer; for the sake of interoperability, it is
RECOMMENDED
           to choose a language that is widely understood among the community
of
           network administrators who will use the module.";
      }
      leaf contact {
        type string;
        description
          "This statement provides contact information for the module.
           The argument is a string that is used to specify contact information
           for the person or persons to whom technical queries concerning this
           module should be sent, such as their name, postal address, telephone
           number, and electronic mail address.";
      }
      leaf module-type {
        type enumeration {
          enum module {
            description
              "If YANG file contains module.";
          }
          enum submodule {
            description
              "If YANG file contains sub-module.";
          }
        }
        description
          "Whether a file contains a YANG module or sub-module.";
      }
      leaf belongs-to {
        when "../module-type = 'submodule'" {
          description
            "Include the module's parent when it is a submodule.";
        }
        type yang:yang-identifier;
        description
          "Name of the module that includes this submodule.";
      }
      leaf tree-type {
        type enumeration {
          enum split {
            description
              "This module uses a split config/operational state layout.";
          }
          enum nmda-compatible {
            description
              "This module is compatible with the Network Management Datastores
               Architecture (NMDA) and combines config and operational state
```

```
nodes.";
          }
          enum transitional-extra {
```

```
            description
              "This module is derived as a '-state' module to allow for
transitioning
               to a full NMDA-compliant tree structure.";
          }
          enum openconfig {
            description
              "This module uses the Openconfig data element layout.";
          }
          enum unclassified {
            description
              "This module does not belong to any category or can't be
determined.";
          }
          enum not-applicable {
            description
              "This module is not applicable. For example, because the YANG
module only contains typedefs, groupings, or is a submodule";
          }
        }
        description
          "The type of data element tree used by the module as it relates to
the
           Network Management Datastores Architecture.";
        reference "draft-dsdt-nmda-guidelines Guidelines for YANG Module
Authors (NMDA)";
      }
      leaf yang-tree {
        when "../module-type = 'module'";
        type inet:uri;
        description
          "This leaf provides a URI that points to the ASCII tree format of the
module in
           draft-ietf-netmod-yang-tree-diagrams format.";
        reference "See draft-ietf-netmod-yang-tree-diagrams.";
      }
      leaf expires {
        type yang:date-and-time;
        description
          "Date and time of when this module expires (if it expires).  This
will typically be used for
           modules that have not been fully ratified.";
      }
      leaf expired {
        type union {
          type boolean;
          type enumeration {
            enum not-applicable {
```

```
            description
              "This module is not and will not be expired.";
          }
        }
      }
      default "false";
      description
```

```
          "Whether or not this module has expired.  If the current date is
beyond the expires date, then expired
           should be true.";
      }
      description
        "Grouping of YANG module metadata that extends the common list defined
in the YANG
         Module Library (RFC 7895).";
    }

    grouping organization-specific-metadata {
      container ietf {
        when "../organization = 'ietf'" {
          description
            "Include this container specific metadata of the IETF.";
        }
        leaf ietf-wg {
          type string;
          description
            "Working group that authored the document containing this module.";
        }
        description
          "Include this container for the IETF-specific organization
metadata.";
      }
      description
        "Any organization that has some specific metadata of the yang module
and want them add to the
         yang-catalog, should augment this grouping. This grouping is for any
metadata that can`t be used for
         every yang module.";
    }

    grouping yang-lib-common-leafs {
      leaf name {
        type yang:yang-identifier;
        description
          "The YANG module or submodule name.";
      }
      leaf revision {
        type union {
          type yanglib:revision-identifier;
          type string {
            length "0";
          }
        }
        description
          "The YANG module or submodule revision date.
```

```
          A zero-length string is used if no revision statement
          is present in the YANG module or submodule.";
      }
      description
        "The YANG module or submodule revision date.
```

```
         A zero-length string is used if no revision statement
         is present in the YANG module or submodule.";
      reference "RFC7895 YANG Module Library : common-leafs grouping";
    }

    grouping yang-lib-schema-leaf {
      leaf schema {
        type inet:uri;
        description
          "Contains a URL that represents the YANG schema
           resource for this module or submodule.
           This leaf will only be present if there is a URL
           available for retrieval of the schema for this entry.";
      }
      description
        "These are a subset of leafs from the yang-library (RFC 7895) that
provide some
         extractable fields for catalog modules.  The module-list grouping
cannot be
         used from yang-library as modules themselves cannot have conformance
without
         a server.";
      reference "RFC7895 YANG Module Library : schema-leaf grouping";
    }

    grouping yang-lib-implementation-leafs {
      leaf-list feature {
        type yang:yang-identifier;
        description
          "List of YANG feature names from this module that are
           supported by the server, regardless of whether they are
           defined in the module or any included submodule.";
      }
      list deviation {
        key "name revision";
        description
          "List of YANG deviation module names and revisions
           used by this server to modify the conformance of
           the module associated with this entry.  Note that
           the same module can be used for deviations for
           multiple modules, so the same entry MAY appear
           within multiple 'module' entries.
           The deviation module MUST be present in the 'module'
           list, with the same name and revision values.
           The 'conformance-type' value will be 'implement' for
           the deviation module.";
        uses yang-lib-common-leafs;
      }
```

```
      leaf conformance-type {
        type enumeration {
          enum implement {
```

```
            description
              "Indicates that the server implements one or more
               protocol-accessible objects defined in the YANG module
               identified in this entry.  This includes deviation
               statements defined in the module.
               For YANG version 1.1 modules, there is at most one
               module entry with conformance type 'implement' for a
               particular module name, since YANG 1.1 requires that,
               at most, one revision of a module is implemented.
               For YANG version 1 modules, there SHOULD NOT be more
               than one module entry for a particular module name.";
          }
          enum import {
            description
              "Indicates that the server imports reusable definitions
               from the specified revision of the module but does
               not implement any protocol-accessible objects from
               this revision.
               Multiple module entries for the same module name MAY
               exist.  This can occur if multiple modules import the
               same module but specify different revision dates in
               the import statements.";
          }
        }
        // Removing the mandatory true for now as not all vendors may have
        // this information if they do not implement yang-library.
        //mandatory true;
        description
          "Indicates the type of conformance the server is claiming
           for the YANG module identified by this entry.";
      }
      description
        "This is a set of leafs extracted from the yang-library that are
         specific to server implementations.";
      reference "RFC7895 YANG Module Library : module-list grouping";
    }

    grouping shared-implementation-leafs {
      leaf os-version {
        type string;
        description
          "Version of the operating system using this module.  This is
primarily useful if
           the software implementing the module is an application that requires
a specific
           operating system.";
      }
      leaf feature-set {
```

```
        type string;
        description
```

```
          "An optional feature of the software that is required in order to
implement this
           module.  Some form of this must be incorporated in software-version
or
           software-flavor, but can be broken out here for additional
clarity.";
      }
      leaf os-type {
        type string;
        description
          "Type of the operating system using this module.  This is primarily
useful if
           the software implementing the module is an application that requires
a
           specific operating system.";
      }
      description
        "Grouping of non-key leafs to be used in the module and vendor sub-
trees.";
    }

    grouping shared-module-leafs {
      leaf generated-from {
        type enumeration {
          enum mib {
            description
              "Module generated from Structure of Management Information (SMI)
               MIB per RFC6643.";
          }
          enum not-applicable {
            description
              "Module was not generated but it was authored manually.";
          }
          enum native {
            description
              "Module generated from platform internal,
               proprietary structure, or code.";
          }
        }
        default "not-applicable";
        description
          "This statement defines weather the module was generated or not.
           Default value is set to not-applicable, which means that module
           was created manualy and not generated.";
      }
      leaf maturity-level {
        type enumeration {
          enum ratified {
```

```
            description
              "Maturity of a module that is fully approved (e.g., a
standard).";
          }
          enum adopted {
            description
              "Maturity of a module that is actively being developed by a
organization towards ratification.";
```

```
          }
          enum initial {
            description
              "Maturity of a module that has been initially created, but has no
official
               organization-level status.";
          }
          enum not-applicable {
            description
              "The maturity level is not used for vendor-supplied models, and
thus all vendor
               modules will have a maturity of not-applicable";
          }
        }
        description
          "The current maturity of the module with respect to the body that
created it.
           This allows one to understand where the module is in its overall
life cycle.";
      }
      leaf document-name {
        type string;
        description
          "The name of the document from which the module was extracted or
taken;
           or that provides additional context about the module.";
      }
      leaf author-email {
        type yc:email-address;
        description
          "Contact email of the author who is responsible for this module.";
      }
      leaf reference {
        type inet:uri;
        description
          "A string that is used to specify a textual cross-reference to an
external document, either
           another module that defines related management information, or a
document that provides
           additional information relevant to this definition.";
      }
      leaf module-classification {
        type enumeration {
          enum network-service {
            description
              "Network Service YANG Module that describes the configuration,
state
               data, operations, and notifications of abstract representations
```

```
of
               services implemented on one or multiple network elements.";
          }
          enum network-element {
            description
              "Network Element YANG Module that describes the configuration,
state
               data, operations, and notifications of specific device-centric
               technologies or features.";
          }
```

```
          enum unknown {
            description
              "In case that there is not sufficient information about how to
classify the module.";
          }
          enum not-applicable {
            description
              "The YANG module abstraction type is neither a Network Service
YANG Module
               nor a Network Element YANG Module.";
          }
        }
        mandatory true;
        description
          "The high-level classification of the given YANG module.";
        reference "RFC8199 YANG Module Classification";
      }
      description
        "These leafs are shared among the yang-catalog and its API.";
    }

    grouping online-source-file {
      leaf owner {
        type string;
        mandatory true;
        description
          "Username or ID of the owner of the version control system
repository.";
      }
      leaf repository {
        type string;
        mandatory true;
        description
          "The name of the repository.";
      }
      leaf path {
        type yc:path;
        mandatory true;
        description
          "Location within the repository of the module file.";
      }
      leaf branch {
        type string;
        description
          "Revision control system branch or tag to use to find the module.  If
this is not
           specified, the head of the repository is used.";
      }
```

```
      description
        "Networked version control system location of the module file.";
    }
  }
```

<CODE ENDS>

5. Security Considerations

The goal of the YANG Catalog module and yangcatalog.org is to document a large library of YANG modules and their implementations. Already, we have seen some SDOs hestitant to provide modules that have not reached a "ratified" maturity level because of intellectual property leakage concerns or simply organization process that mandates only fully ratified modules can be published. Care must be paid that through private automated testing and validation of such modules that their metadata does not leak before the publishing organization approves the release of such data.

Similarly, from a vendor implementation standpoint, data that is exposed to the catalog before the vendor has fully vetted it could cause confusion amongst that vendor's customers or reveal product releases to the market before they have been officially announced.

Ultimately, there is a balance to be struck with respect to providing a rich library of YANG module metadata, and doing so at the right time to avoid information leakage.

6. IANA Considerations

No IANA action is requested.

7. References

7.1. Normative References

[I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-06 (work in progress), February 2018.

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

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

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

[RFC8199] Bogdanovic, D., Claise, B., and C. Moberg, "YANG Module Classification", RFC 8199, DOI 10.17487/RFC8199, July 2017, <https://www.rfc-editor.org/info/rfc8199>.

[semver] "Semantic Versioning 2.0.0", <https://www.semver.org>.

7.2. Informative References

[I-D.openconfig-netmod-model-catalog] Shaikh, A., Shakir, R., and K. D'Souza, "Catalog and registry for YANG models", draft-openconfig-netmod-model-catalog-02 (work in progress), March 2017.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

7.3. URIs

[1] https://developer.cisco.com/site/confD/index.gsp

[2] https://www.yangcatalog.org/yang-search

Appendix A. Acknowledgments

The authors would like to thanks Miroslav Kovac for this help on this YANG module and the yangcatalog.org implementation. We would also like to thank Radek Krejci for his extensive review and suggestions for improvement.

The RFC text was produced using Marshall Rose's xml2rfc tool.

Appendix B. Changes From Previous Revisions

RFC Editor to remove this section prior to publication.

Draft -00 to -01:

- Redesign of module sub-tree based on review.
- Modularize some leafs and create typedefs to share with API YANG modules.
- Add module conformance-type, deviation and feature leafs under the implementation branch.
- Change yang-version to be an enum.
- Add a leaf for module-classification based on [RFC8199].
- Normalize enums to be lowercase.
- Use identities for protocols instead of an enumeration.
- Make conformance-type optional as not all vendors implement [RFC7895].
- Add a leaf for tree-type based on [RFC8342].
- Add a reference to contributing to the YANG Catalog at yangcatalog.org.
- Various wording and style changes to the document text.

Draft -01 to -02:

- Add a belongs-to leaf to track parent modules.
- Add leafs to track dependents and dependencies for a given module.
- Simplify the generated-from enumerated values.
- Refine the type for compilation-result to be an inet:uri.
- Add leafs for semantic versioning.
- Reorder the organization leaf to be with other module keys.
- Add text to describe generated-from and semantic versioning.

Draft -02 to -03:

- Change YANG ref to RFC7950 as the catalog module now needs YANG 1.1.
- Add a reference to I-D.ietf-netmod-yang-tree-diagrams.
- Document the new yang-tree node in the catalog.
- Document the new expires and expired leafs and their relation to maturity.
- Updtae NMDA reference to point to new RFC number.

Authors' Addresses

Joe Clarke
Cisco Systems, Inc.
7200-12 Kit Creek Rd
Research Triangle Park, North Carolina
United States of America

Phone: +1-919-392-2867
Email: jclarke@cisco.com

Benoit Claise
Cisco Systems, Inc.
De Kleetlaan 6a b1
1831 Diegem
Belgium

Phone: +32 2 704 5622
Email: bclaise@cisco.com