

Network Working Group  
Internet-Draft  
Updates: 7950 (if approved)  
Intended status: Standards Track  
Expires: 28 November 2026

P. Andersson, Ed.  
Ionio Systems  
R. Wilton  
Cisco Systems, Inc.  
M. Valisko  
CESNET  
27 May 2026

YANG Schema Comparison  
draft-ietf-netmod-yang-schema-comparison-08

## Abstract

This document specifies an algorithm for comparing two revisions of a YANG schema to determine the scope of changes, and a list of changes, between the revisions. The output of the algorithm can be used to help select an appropriate revision-label or YANG semantic version number for a new revision. Included is also a YANG module describing possible output of this algorithm.

## 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 28 November 2026.

## Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components

extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology and Conventions . . . . .	3
3. Parsed schema tree . . . . .	4
4. Compiled schema tree . . . . .	4
5. YANG schema tree comparison algorithm . . . . .	5
5.1. Module identification . . . . .	6
5.2. Change content . . . . .	6
5.3. Change conformance . . . . .	6
5.3.1. Editorial change . . . . .	7
5.3.2. Backwards-compatible change . . . . .	7
5.3.3. Non-backwards-compatible change . . . . .	7
5.3.4. Extensions . . . . .	8
5.4. Compiled and parsed schema changes . . . . .	9
6. Comparison tooling . . . . .	9
7. Schema Comparison YANG Module . . . . .	9
8. Contributors . . . . .	12
9. Security Considerations . . . . .	12
10. IANA Considerations . . . . .	13
10.1. The "IETF XML" Registry . . . . .	13
10.2. The "YANG Module Names" Registry . . . . .	14
11. References . . . . .	14
11.1. Normative References . . . . .	14
11.2. Informative References . . . . .	16
Appendix A. Schema Comparison Algorithm Output YANG Module . . .	16
Appendix B. Comparison Examples . . . . .	48
B.1. Relaxed Length Module . . . . .	48
Authors' Addresses . . . . .	51

## 1. Introduction

This document defines the algorithm for comparing two revisions of a YANG schema. There are two kinds of YANG schemas: on-the-wire or compiled schema and text or parsed schema.

The compiled schema is what tools use when working with any kinds of YANG data. Generally, in all the YANG modules, all the schema nodes need to be fully resolved and have their final form. For example, every "uses" statement is replaced with the nodes from the corresponding "grouping", every type referencing a "typedef" is replaced with the type of the "typedef" and so on. On the other hand, the parsed schema is comprised of all the YANG modules together as they are. That means no statements are resolved and they remain in exactly the same text form as present in the YANG modules.

For determining whether a new revision of a YANG 1.1 module is or is not backwards-compatible based on all the rules in [RFC7950] and [I-D.ietf-netmod-yang-module-versioning], both of these schemas would be required. Similarly for YANG 1.0, the rules in [RFC6020]. This document defines an algorithm for comparing either only the compiled schema or both the parsed and compiled schema.

The output of this algorithm can be utilized by tools processing YANG instance data to determine what exact changes should they expect and even how to transform instance data valid for an old revision of a YANG module to be valid for a newer revision of the same YANG module. Such tools are typically clients or servers of YANG-driven protocols, e.g. NETCONF, RESTCONF, or gRPC.

But, the generated data can also be used by YANG module authors to check all the changes made between 2 revisions. Additionally, it can assist with deciding what semantic version should the new revision of the module use based on the [I-D.ietf-netmod-yang-semver] document.

## 2. Terminology and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document makes use of the following terminology introduced in the YANG 1.1 Data Modeling Language [RFC7950]:

- \* module
- \* schema node
- \* data node
- \* data definition statement

### 3. Parsed schema tree

When a YANG module is parsed and its contents stored in an internal representation without any statement resolution or modification, the result is a parsed schema tree. Such a schema tree can be printed into a YANG module and it should be exactly the same as the loaded source YANG module.

### 4. Compiled schema tree

The compiled schema node tree resolves all the statements used for syntactic abstraction and simplification to obtain one schema node tree with only data nodes. Textual fields are kept as well because they may affect backwards compatibility of the whole YANG module.

The steps taken when compiling a parsed schema:

- \* "Import" statements are resolved by finding the referenced YANG modules and making their contents available.
- \* All submodules of a single module are merged to create one large YANG module.
- \* "Uses" statements are replaced by the referenced "groupings" and the YANG nodes they define.
- \* Types referencing any "typedef" statements are replaced with the actual type details.
- \* Augments are resolved by placing the included YANG nodes in their targets.
- \* "Deviation" and "refine" statements are applied by changing the appropriate schema nodes.
- \* All the "if-feature" statements are evaluated based on the enabled "features" of implemented YANG modules and the disabled schema nodes are removed.

After the compilation the result is still a valid YANG module without the following statements:

- \* import
- \* include
- \* revision

- \* typedef
- \* grouping
- \* augment
- \* deviation
- \* feature
- \* if-feature
- \* refine
- \* extension

## 5. YANG schema tree comparison algorithm

This algorithm is defined for separate YANG modules. When comparing the compiled schema, all the non data node statements are first considered and then all the data nodes, recursively. These data definition statements that are not data nodes are skipped:

- \* choice
- \* case

When comparing the parsed schema, the algorithm operates in the same fashion but compares all the YANG statements. However, only in the selected YANG module ignoring imported and included modules.

The non data definition statements are considered modified when their content is changed or any of their substatements are changed. If a statement is present in the old revision of a module and missing in the new revision, it is considered removed. Vice-versa, if missing in the old revision but present in the new revision, it is considered added.

Data definition statements are compared similarly except that a change in a descendant data definition statement is not considered a change of the parent data definition statement but a separate change instead.

Only statements with any changes are reported and included in the algorithm output.

### 5.1. Module identification

When generating the comparison output of two compiled schemas, unique identification of the modules in question is included. For both the old and the new revision of the module, its name, revision, list of submodules, and a leaf-list of enabled features is included. In addition, the same information is reported for all the imported YANG modules, recursively. This ensures that if any other output is produced with a different set of changes for two YANG modules, their identification information must also differ.

### 5.2. Change content

Non data definition statements are uniquely identified with their YANG identifier and data definition statements with their absolute schema node-ids.

For every change, the changed statement and the parent of the changed statement are included. While redundant in many cases, it allows to uniquely report changes in nested statements, e.g. an extension instance change in the type of a leaf statement. Also, all the unchanged substatements with their value or content are included for every changed statement, for both older and newer revision of the YANG module, if applicable. This ensures that the output includes all the possibly required information without relying on learning it from the YANG modules themselves.

### 5.3. Change conformance

Every change is classified as editorial, backwards-compatible or non-backwards-compatible. The classification is based on section 11. of [RFC7950] that mentions most of the possible changes between 2 revisions of a YANG 1.1 module and section 3.1. of [I-D.ietf-netmod-yang-module-versioning] adds more details about the exact conformance of statements with various "status" statement values.

The following text tries to establish general rules of how changes conformance is selected. Unless there are other statement-specific rules in the aforementioned documents, these should always apply.

#### 5.3.1. Editorial change

An editorial change MUST NOT affect valid YANG data of the YANG module in any way. Meaning it MUST NOT make the valid data value space smaller or larger and it must be exactly the same as in the module revision without the change. Also, for a change to be editorial, it MUST NOT affect any YANG module importing the changed YANG module.

Except for some obvious editorial changes such as fixing a typo in a "description", changing the local "prefix" of the module is also categorized as editorial because it does not strictly affect any importing module. Or, replacing a "uses" statement directly with the nodes in the referenced "grouping" without any changes.

#### 5.3.2. Backwards-compatible change

For a change to be backwards-compatible, it MUST expand the value space of valid YANG data. Meaning all the valid YANG data nodes and values before the change must remain valid but additional nodes or values can also become valid after the change. Also, backwards-compatible changes may influence importing modules but MUST NOT cause them to become invalid. In other words, they may exhibit new statements whose removal would then be considered a non-backwards-compatible change but generally must not change the already exposed statements.

For some examples, the majority of possible backwards-compatible changes are specifically mentioned directly in [RFC7950].

#### 5.3.3. Non-backwards-compatible change

If a change is not editorial nor backwards-compatible, it MUST be categorized as non-backwards-compatible. Such changes shrink the value space of valid YANG data causing some previously valid nodes or values to no longer be valid. Also, any change that may cause an importing module to become invalid according to the YANG specification MUST be non-backwards-compatible.

Examples of a less common or obvious compatibility-breaking changes are changing the "yang-version" of a YANG module or expanding the size of a "type". The former change may break importing modules because of specific rules about "import" statement and the target module's YANG version. Changing a "type" of a leaf from "uint32" to "uint64" may seem to fit the definition of a backwards-compatible change, but it may not be so for some data encodings that use a fixed binary data chunk (which would grow in size) or even a text format that may use a different encoding for the two types. Specifically, the JSON YANG data format [RFC7951] encodes a "uint32" value as a JSON number but a "uint64" as a JSON string.

#### 5.3.4. Extensions

There are cases when it may be impossible for tooling to accurately determine the conformance of a change but there is always some conformance assumed for every statement. The affected ambiguous statements with their default conformances are:

- \* pattern - NBC
- \* when - NBC
- \* must - NBC
- \* description - ED
- \* reference - ED
- \* presence - ED
- \* extension instance - BC

Normally, any modification of these statements is reported as the default assumed change conformance. However, if the author of the new revision of a YANG module determines that the change of an aforementioned statement has a different conformance based on the compatibility rules for the specific statement, they can decide to use the special extension instances to signify this fact. These are:

- \* ed-change-at SEMVER
- \* bc-change-at SEMVER
- \* nbc-change-at SEMVER



They determine the conformance of the change of the marked statement to be editorial, backwards-compatible, or non-backwards-compatible, respectively. The argument is a semantic version of the YANG module revision that introduced the specific change and these extension instances are sticky meaning they remain in the module even in its later revisions.

#### 5.4. Compiled and parsed schema changes

Either only compiled data node changes are reported or both compiled and parsed statements are compared and their changes generated depending on the enabled features of the "ietf-yang-schema-comparison" YANG module.

In the latter case, compiled data node changes are generated normally. Then, all the other statements are compared in their original parsed form skipping those that are compiled. Specifically, all the data node statements are found in the compiled schema tree meaning they are not compared the second time as part of the parsed schema tree comparison.

#### 6. Comparison tooling

The "yanglint" tool from the libyang library (in separate Git branch `schema_comparison`) is able to generate the "ietf-yang-schema-comparison" YANG data of two revisions of a YANG module.

#### 7. Schema Comparison YANG Module

YANG module with extensions fine-tuning the classification of differences between two revisions of YANG schemas.

```
<CODE BEGINS> file "ietf-yang-schema-comparison@2026-05-27.yang"
module ietf-yang-schema-comparison {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-schema-comparison";
  prefix schema-cmp;

  organization
    "IETF NETMOD (Network Modeling) Working Group";
  contact
    "WG List: NETCONF WG list <mailto:netmod@ietf.org>
    WG Web:  https://datatracker.ietf.org/wg/netmod

    Author:  Michal Vasko
             <mailto:mvasko@cesnet.cz>

    Author:  Rob Wilton
```

<mailto:rwilton@cisco.com>

Author: Per Andersson  
<mailto:per.ietf@ionio.se>;

description

"This YANG 1.1 module contains extensions to support comparison between different versions of YANG schemas.

Copyright (c) 2026 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Revised BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX (<https://www.rfc-editor.org/info/rfcXXXX>); see the RFC itself for full legal notices.

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document are to be interpreted as described in BCP 14 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.";

// RFC Ed.: update the date below with the date of RFC publication  
// and remove this note.  
// RFC Ed.: replace XXXX (inc above) with actual RFC number and  
// remove this note.

```
revision 2026-05-27 {  
  description  
    "Initial revision.";  
  reference  
    "RFC XXXX: YANG Schema Comparison";  
}
```

```
extension ed-change-at {  
  argument semver;  
  description  
    "Marks statements in a new revision of a module  
    that introduces an editorial change of the statement.
```

The argument marks the most recent version of the

```
    module where the statement has been changed.

    Can only be a substatement of 'pattern', 'when',
    'must', 'description', 'reference', 'presence', or
    extension instance statements.";
reference
  "RFC 7950: The YANG 1.1 Data Modeling Language, section 11;
  RFC 6020: YANG - A Data Modeling Language for the Network
  Configuration Protocol (NETCONF), section 10";
}

extension bc-change-at {
  argument semver;
  description
    "Marks statements in a new revision of a module
    that introduces a backwards-compatible change of
    the statement.

    The argument marks the most recent version of the
    module where the statement has been changed.

    Can only be a substatement of 'pattern', 'when',
    'must', 'description', 'reference', 'presence', or
    extension instance statements.";
reference
  "RFC 7950: The YANG 1.1 Data Modeling Language, section 11;
  RFC 6020: YANG - A Data Modeling Language for the Network
  Configuration Protocol (NETCONF), section 10";
}

extension nbc-change-at {
  argument semver;
  description
    "Marks statements in a new revision of a module
    that introduces a non-backwards-compatible change of
    the statement.

    The argument marks the most recent version of the
    module where the statement has been changed.

    Can only be a substatement of 'pattern', 'when',
    'must', 'description', 'reference', 'presence', or
    extension instance statements.";
reference
  "RFC 7950: The YANG 1.1 Data Modeling Language, section 11;
  RFC 6020: YANG - A Data Modeling Language for the Network
  Configuration Protocol (NETCONF), section 10";
}
```

```
}  
<CODE ENDS>
```

## 8. Contributors

This document grew out of the YANG module versioning design team that started after IETF 101. The following individuals are (or have been) members of the design team and have worked on the YANG versioning project:

- \* Balazs Lengyel
- \* Benoit Claise
- \* Bo Wu
- \* Ebben Aries
- \* Jason Sterne
- \* Joe Clarke
- \* Juergen Schoenwaelder
- \* Mahesh Jethanandani
- \* Michael Wang
- \* Qin Wu
- \* Reshad Rahman
- \* Rob Wilton
- \* Jan Lindblad
- \* Per Andersson

The ideas for a tooling based comparison of YANG module revisions was first described in [I-D.clacla-netmod-yang-model-update]. This document extends upon those initial ideas.

## 9. Security Considerations

This section follows the template defined in Section 3.7.1 of [RFC8407].

The YANG module specified in this document defines a schema for data that is designed to be used by offline tooling to generate output for differences in supplied YANG modules.

The YANG module specified in this document MAY be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

When the module is used for offline tooling there are no security considerations, since the user has full access to all YANG modules used.

The structure "schema-comparison" contains all the groupings reflecting the changes between YANG modules. If the data of this structure are published in online tooling, care needs to be taken so that knowledge of YANG modules is not leaked.

Since the module does not define any RPCs, actions, or notifications, the security considerations for such statements are not provided here.

## 10. IANA Considerations

### 10.1. The "IETF XML" Registry

This document register one URI in the "ns" subregistry of the IETF XML Registry [RFC3688] maintained at <https://www.iana.org/assignments/xml-registry/xml-registry.xhtml#ns>. Following the format in [RFC3688], the following registration is requested:

URI: urn:ietf:params:xml:ns:yang:ietf-yang-schema-comparison  
Registrant Contact: The IESG.  
XML: N/A, the requested URI is an XML namespace.

## 10.2. The "YANG Module Names" Registry

This document registers two YANG modules in the YANG Module Names registry [RFC6020] maintained at <https://www.iana.org/assignments/yang-parameters/yang-parameters.xhtml>. Following the format defined in [RFC6020], the below registrations are requested:

Name: ietf-yang-schema-comparison

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-schema-comparison

Prefix: schema-cmp

Reference: RFC XXXX

Name: ietf-yang-schema-comparison-output

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-schema-comparison-output

Prefix: schema-cmp-out

Reference: RFC XXXX

## 11. References

### 11.1. Normative References

[I-D.ietf-netmod-yang-module-versioning]

Wilton, R., Rahman, R., Lengyel, B., Clarke, J., and J. Sterne, "Updated YANG Module Revision Handling", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-module-versioning-16, 21 April 2026, <<https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-module-versioning-16>>.

[I-D.ietf-netmod-yang-semver]

Clarke, J., Wilton, R., Rahman, R., Lengyel, B., Sterne, J., and B. Claise, "YANG Semantic Versioning", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-semver-26, 7 May 2026, <<https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-semver-26>>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

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

- [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>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [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>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<https://www.rfc-editor.org/info/rfc7951>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", RFC 8407, DOI 10.17487/RFC8407, October 2018, <<https://www.rfc-editor.org/info/rfc8407>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

- [RFC8525] Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", RFC 8525, DOI 10.17487/RFC8525, March 2019, <<https://www.rfc-editor.org/info/rfc8525>>.
- [RFC8791] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Data Structure Extensions", RFC 8791, DOI 10.17487/RFC8791, June 2020, <<https://www.rfc-editor.org/info/rfc8791>>.

## 11.2. Informative References

- [I-D.clacla-netmod-yang-model-update]  
Claise, B., Clarke, J., Lengyel, B., and K. D'Souza, "New YANG Module Update Procedure", Work in Progress, Internet-Draft, draft-clacla-netmod-yang-model-update-06, 2 July 2018, <<https://datatracker.ietf.org/doc/html/draft-clacla-netmod-yang-model-update-06>>.

## Appendix A. Schema Comparison Algorithm Output YANG Module

YANG module with specific output of the schema comparison algorithm supporting comparison of compiled or parsed and compiled schema tree of a YANG module. The "ietf-yang-schema-comparison-output" YANG module imports definitions from the "ietf-yang-types" module defined in [RFC6991] and "ietf-yang-library" module defined in [RFC8525], and "ietf-yang-structure-ext" module defined in [RFC8791].

```
<CODE BEGINS>
file "ietf-yang-schema-comparison-output@2026-05-27.yang"
module ietf-yang-schema-comparison-output {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-schema-comparison-output";
  prefix schema-cmp-out;

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data types";
  }
  import ietf-yang-library {
    prefix yanglib;
    reference
      "RFC 8525: YANG Library";
  }
  import ietf-yang-structure-ext {
    prefix sx;
    reference
      "RFC 8791: YANG Data Structure Extensions";
  }
}
```



```
}

organization
  "IETF NETMOD (Network Modeling) Working Group";
contact
  "WG List: NETCONF WG list <mailto:netmod@ietf.org>
  WG Web:  https://datatracker.ietf.org/wg/netmod

  Author:  Michal Vasko
           <mailto:mvasko@cesnet.cz>

  Author:  Rob Wilton
           <mailto:rwilton@cisco.com>

  Author:  Per Andersson
           <mailto:per.ietf@ionio.se>";
description
  "This YANG 1.1 module contains definitions of output of comparison
  between different versions of YANG schemas. The base schema nodes
  describe changes in a compiled YANG module with all of its
  included submodules and imported modules. The rest of the schema
  nodes are optional and describe changes in all the statements of a
  single YANG module.

  Copyright (c) 2026 IETF Trust and the persons identified as
  authors of the code.  All rights reserved.

  Redistribution and use in source and binary forms, with
  or without modification, is permitted pursuant to, and
  subject to the license terms contained in, the Revised
  BSD License set forth in Section 4.c of the IETF Trust's
  Legal Provisions Relating to IETF Documents
  (https://trustee.ietf.org/license-info).

  This version of this YANG module is part of RFC XXXX
  (https://www.rfc-editor.org/info/rfcXXXX); see the RFC
  itself for full legal notices.

  The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
  'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
  'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
  are to be interpreted as described in BCP 14 (RFC 2119)
  (RFC 8174) when, and only when, they appear in all
  capitals, as shown here.";

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
// RFC Ed.: replace XXXX (inc above) with actual RFC number and
```

```
// remove this note.

revision 2026-05-27 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: YANG Schema Comparison";
}

feature parsed-schema {
  description
    "Compare also the statements removed from the compiled schema to
    get all the backwards-compatible and non-backwards-compatible
    changes possible for a YANG module.";
  reference
    "RFC 7950: The YANG 1.1 Data Modeling Language, section 11;
    RFC 6020: YANG - A Data Modeling Language for the Network
    Configuration Protocol (NETCONF), section 10";
}

typedef schema-node-path {
  type string;
  description
    "Node schema path (node-id). A generic type is used to not enforce
    their evaluation or validation in this context.";
}

typedef schema-stmt-path {
  type string;
  description
    "Path to a statement. Any non data node parents are encoded
    specially.";
}

typedef stmt-type {
  type enumeration {
    enum base {
      description
        "YANG statement 'base'.";
    }
    enum bit {
      description
        "YANG statement 'bit'.";
    }
    enum config {
      description
        "YANG statement 'config'.";
    }
  }
}
```

```
enum contact {
  description
    "YANG statement 'contact'.";
}
enum default {
  description
    "YANG statement 'default'.";
}
enum description {
  description
    "YANG statement 'description'.";
}
enum deviate {
  description
    "YANG statement 'deviate'.";
}
enum deviation {
  description
    "YANG statement 'deviation'.";
}
enum enum {
  description
    "YANG statement 'enum'.";
}
enum error-app-tag {
  description
    "YANG statement 'error-app-tag'.";
}
enum error-message {
  description
    "YANG statement 'error-message'.";
}
enum extension {
  description
    "YANG statement 'extension'.";
}
enum extension-instance {
  description
    "Any extension instance YANG statement.";
}
enum feature {
  description
    "YANG statement 'feature'.";
}
enum fraction-digits {
  description
    "YANG statement 'fraction-digits'.";
}
```

```
enum identity {
  description
    "YANG statement 'identity'.";
}
enum if-feature {
  description
    "YANG statement 'if-feature'.";
}
enum import {
  description
    "YANG statement 'import'.";
}
enum include {
  description
    "YANG statement 'include'.";
}
enum length {
  description
    "YANG statement 'length'.";
}
enum mandatory {
  description
    "YANG statement 'mandatory'.";
}
enum max-elements {
  description
    "YANG statement 'max-elements'.";
}
enum min-elements {
  description
    "YANG statement 'min-elements'.";
}
enum must {
  description
    "YANG statement 'must'.";
}
enum node {
  description
    "YANG statement 'container', 'leaf', 'leaf-list', 'list',
    'anydata', 'anyxml', 'rpc', 'action', or 'notification'. If
    the parsed schema is also compared, it can additionally be
    'choice', 'case', 'input', 'output', 'uses', 'augment', or
    'grouping'.";
}
enum ordered-by {
  description
    "YANG statement 'ordered-by'.";
}
```

```
enum organization {
  description
    "YANG statement 'organization'.";
}
enum path {
  description
    "YANG statement 'path'.";
}
enum pattern {
  description
    "YANG statement 'pattern'.";
}
enum prefix {
  description
    "YANG statement 'prefix'.";
}
enum presence {
  description
    "YANG statement 'presence'.";
}
enum range {
  description
    "YANG statement 'range'.";
}
enum reference {
  description
    "YANG statement 'reference'.";
}
enum refine {
  description
    "YANG statement 'refine'.";
}
enum require-instance {
  description
    "YANG statement 'require-instance'.";
}
enum revision-date {
  description
    "YANG statement 'revision-date'.";
}
enum status {
  description
    "YANG statement 'status'.";
}
enum type {
  description
    "YANG statement 'type'.";
}
```

```
enum typedef {
  description
    "YANG statement 'typedef'.";
}
enum units {
  description
    "YANG statement 'units'.";
}
enum unique {
  description
    "YANG statement 'unique'.";
}
enum when {
  description
    "YANG statement 'when'.";
}
enum yang-version {
  description
    "YANG statement 'yang-version'.";
}
}
description
  "Type of the statement that a change affects.";
}

typedef revision-or-empty {
  type union {
    type empty;
    type yanglib:revision-identifier;
  }
  description
    "Module or submodule revision. If it has none, an empty value is
    used.";
}

grouping if-feature-stmts {
  description
    "Describes changes of all the if-features on a statement.";
  leaf-list if-feature {
    type string;
    description
      "Value of an if-feature statement.";
  }
}

grouping ext-instance-stmt {
  description
    "Describes changes of an extension instance.";
```

```
leaf module {
  type yang:yang-identifier;
  mandatory true;
  description
    "Module with the extension definition of an extension
    instance.";
}
leaf name {
  type yang:yang-identifier;
  mandatory true;
  description
    "Name of the extension defition of an extension instance.";
}
leaf argument {
  type string;
  description
    "Argument used of an extension definition.";
}
anydata substatements {
  description
    "Any substatements of the extension instance.";
}
}

grouping ext-instance-stmts {
  description
    "Describes changes of all the extension instances.";
  list ext-instance {
    description
      "List of extension-instance substatements.";
    uses ext-instance-stmt;
  }
}

grouping status-stmt {
  description
    "Describes a status statement change.";
  leaf status {
    type enumeration {
      enum current {
        description
          "Current effective status of a statement.";
      }
      enum deprecated {
        description
          "Deprecated effective status of a statement.";
      }
      enum obsolete {
```

```
        description
            "Obsolete effective status of a statement.";
    }
}
mandatory true;
description
    "Status substatement value.";
}
}

grouping must-stmts {
    description
        "Describes changes of all the must statements of a specific parent
        statement.";
    list must {
        description
            "List of must substatements.";
        leaf condition {
            type string;
            description
                "Condition substatement value.";
        }
        uses restriction-substmts;
    }
}

grouping when-stmts {
    description
        "Describes changes of all the when statements of a specific parent
        statement.";
    list when {
        description
            "List of when substatements.";
        leaf condition {
            type string;
            description
                "Condition substatement value.";
        }
        leaf description {
            type string;
            description
                "Description substatement value.";
        }
        leaf reference {
            type string;
            description
                "Reference substatement value.";
        }
    }
}
```



```
    uses status-stmt;
    uses ext-instance-stmts;
  }
}

grouping restriction-substmts {
  description
    "Common substatements shared by all restriction statements.";
  leaf description {
    type string;
    description
      "Description substatement value.";
  }
  leaf reference {
    type string;
    description
      "Reference substatement value.";
  }
  leaf error-message {
    type string;
    description
      "Error-message substatement value.";
  }
  leaf error-app-tag {
    type string;
    description
      "Error-app-tag substatement value.";
  }
  uses ext-instance-stmts;
}

grouping type-substmts {
  description
    "Describes a type statement change.";
  leaf base-type {
    type enumeration {
      enum int8 {
        description
          "YANG built-in type 'int8'.";
      }
      enum int16 {
        description
          "YANG built-in type 'int16'.";
      }
      enum int32 {
        description
          "YANG built-in type 'int32'.";
      }
    }
  }
}
```

```
enum int64 {
  description
    "YANG built-in type 'int64'.";
}
enum uint8 {
  description
    "YANG built-in type 'uint8'.";
}
enum uint16 {
  description
    "YANG built-in type 'uint16'.";
}
enum uint32 {
  description
    "YANG built-in type 'uint32'.";
}
enum uint64 {
  description
    "YANG built-in type 'uint64'.";
}
enum decimal64 {
  description
    "YANG built-in type 'decimal64'.";
}
enum string {
  description
    "YANG built-in type 'string'.";
}
enum boolean {
  description
    "YANG built-in type 'boolean'.";
}
enum enumeration {
  description
    "YANG built-in type 'enumeration'.";
}
enum bits {
  description
    "YANG built-in type 'bits'.";
}
enum binary {
  description
    "YANG built-in type 'binary'.";
}
enum leafref {
  description
    "YANG built-in type 'leafref'.";
}
```

```
    enum identityref {
      description
        "YANG built-in type 'identityref'.";
    }
    enum empty {
      description
        "YANG built-in type 'empty'.";
    }
    enum union {
      description
        "YANG built-in type 'union'.";
    }
    enum instance-identifier {
      description
        "YANG built-in type 'instance-identifier'.";
    }
  }
  description
    "Type substatement value.";
}
container range {
  description
    "Range statement substatements.";
  list interval {
    description
      "Intervals of a range statement.";
    leaf min {
      type int64;
      description
        "Lower boundary of an interval of a range statement.";
    }
    leaf max {
      type int64;
      description
        "Upper boundary of an interval of a range statement.";
    }
  }
}
uses restriction-substmts;
}
container length {
  description
    "Length statement substatements.";
  list interval {
    description
      "Intervals of a length statement.";
    leaf min {
      type uint64;
      description
```

```
        "Lower boundary of an interval of a length statement.";
    }
    leaf max {
        type uint64;
        description
            "Upper boundary of an interval of a length statement.";
    }
}
uses restriction-substmts;
}
leaf fraction-digits {
    type uint8;
    description
        "Fraction-digits substatement value.";
}
list pattern {
    description
        "List of pattern statements.";
    leaf expression {
        type string;
        description
            "Pattern substatement value.";
    }
    leaf inverted {
        type boolean;
        description
            "A value of true indicates that the pattern's 'modifier'
            substatement is 'invert-match'.";
        reference
            "RFC 7950 Section 9.4.6";
    }
    uses restriction-substmts;
}
list enum {
    key "name";
    ordered-by user;
    description
        "List of enum statements. Keeps the same order of the statements
        as in the module.";
    leaf name {
        type yang:yang-identifier;
        description
            "Enum statement name.";
    }
    uses if-feature-stmts {
        if-feature "parsed-schema";
    }
    leaf description {
```

```
        type string;
        description
            "Description substatement value.";
    }
    leaf reference {
        type string;
        description
            "Reference substatement value.";
    }
    leaf value {
        type int32;
        description
            "Value substatement value.";
    }
    uses status-stmt;
    uses ext-instance-stmts;
}
list bit {
    key "name";
    ordered-by user;
    description
        "List of bit statements. Keeps the same order of the statements
        as in the module.";
    leaf name {
        type yang:yang-identifier;
        description
            "Bit statement name.";
    }
    uses if-feature-stmts {
        if-feature "parsed-schema";
    }
    leaf description {
        type string;
        description
            "Description substatement value.";
    }
    leaf reference {
        type string;
        description
            "Reference substatement value.";
    }
    leaf position {
        type uint32;
        description
            "Position substatement value.";
    }
    uses status-stmt;
    uses ext-instance-stmts;
```

```
    }
    leaf path {
      type string;
      description
        "Path substatement value.";
    }
    leaf require-instance {
      type boolean;
      description
        "Require-instance substatement value.";
    }
    leaf-list base {
      type yang:yang-identifier;
      description
        "List of base substatement values.";
    }
    uses ext-instance-stmts;
  }

  grouping type-parsed-substmts {
    description
      "Describes a parsed type statement change.";
    leaf name {
      type yang:yang-identifier;
      description
        "Type argument substatement value.";
    }
    container range {
      description
        "Range statement substatements.";
      leaf restriction {
        type string;
        description
          "Range argument substatement value.";
      }
      uses restriction-substmts;
    }
    container length {
      description
        "Length statement substatements.";
      leaf restriction {
        type string;
        description
          "Length argument substatement value.";
      }
      uses restriction-substmts;
    }
    leaf fraction-digits {
```

```
    type uint8;
    description
        "Fraction-digits substatement value.";
}
list pattern {
    description
        "List of pattern statements.";
    leaf expression {
        type string;
        description
            "Pattern substatement value.";
    }
    leaf inverted {
        type empty;
        description
            "Inverted substatement value.";
    }
    uses restriction-substmts;
}
list enum {
    key "name";
    ordered-by user;
    description
        "List of enum statements. Keeps the same order of the statements
        as in the module.";
    leaf name {
        type yang:yang-identifier;
        description
            "Enum statement name.";
    }
    uses if-feature-stmts;
    leaf description {
        type string;
        description
            "Description substatement value.";
    }
    leaf reference {
        type string;
        description
            "Reference substatement value.";
    }
    leaf value {
        type int32;
        description
            "Value substatement value.";
    }
    uses status-stmt;
    uses ext-instance-stmts;
```

```
}
list bit {
  key "name";
  ordered-by user;
  description
    "List of bit statements. Keeps the same order of the statements
     as in the module.";
  leaf name {
    type yang:yang-identifier;
    description
      "Bit statement name.";
  }
  uses if-feature-stmts;
  leaf description {
    type string;
    description
      "Description substatement value.";
  }
  leaf reference {
    type string;
    description
      "Reference substatement value.";
  }
  leaf position {
    type uint32;
    description
      "Position substatement value.";
  }
  uses status-stmt;
  uses ext-instance-stmts;
}
leaf path {
  type string;
  description
    "Path substatement value.";
}
leaf require-instance {
  type boolean;
  description
    "Require-instance substatement value.";
}
leaf-list base {
  type yang:yang-identifier;
  description
    "List of base substatement values.";
}
uses ext-instance-stmts;
}
```



```
grouping refine-deviate-node-substmts {
  description
    "All substatements of a data-definition node that can be changed
    by both a refine and deviate statement.";
  uses must-stmts;
  leaf-list default {
    type string;
    description
      "List of default substatement values.";
  }
  leaf config {
    type boolean;
    description
      "Config substatement value.";
  }
  leaf mandatory {
    type boolean;
    description
      "Mandatory substatement value.";
  }
  leaf min-elements {
    type uint32;
    description
      "Min-elements substatement value.";
  }
  leaf max-elements {
    type uint32;
    description
      "Max-elements substatement value.";
  }
}

grouping node-substmts {
  description
    "All substatements of a data-definition node.";
  uses if-feature-stmts {
    if-feature "parsed-schema";
  }
  uses when-stmts;
  leaf description {
    type string;
    description
      "Description substatement value.";
  }
  leaf reference {
    type string;
    description
      "Reference substatement value.";
  }
}
```

```
}
uses status-stmt;
uses refine-deviate-node-substmts;
leaf-list key {
  type yang:yang-identifier;
  description
    "Leaf-list of nodes referenced by a key statement.";
}
leaf ordered-by {
  type enumeration {
    enum system {
      description
        "System-ordered list or leaf-list of items.";
    }
    enum user {
      description
        "User-ordered list or leaf-list of items.";
    }
  }
  description
    "Ordered-by substatement value.";
}
container type {
  description
    "Type substatement.";
  uses type-substmts;
  list union-type {
    description
      "List of a union type substatements.";
    uses type-substmts;
  }
}
leaf units {
  type string;
  description
    "Units substatement value.";
}
list unique {
  description
    "List of unique substatements.";
  leaf-list node {
    type yang:yang-identifier;
    description
      "List of nodes referenced by a unique substatement.";
  }
}
leaf presence {
  type boolean;
```

```
    description
      "Indicates whether a presence substatement is defined for
       the container in the compiled schema. The textual argument
       of 'presence' is reported via the parsed schema comparison
       (see the 'presence' leaf in 'parsed-substmts').";
  }
  uses ext-instance-stmts;
}

grouping module-substmts {
  description
    "All non-data-definition substatements of a module.";
  leaf yang-version {
    type string;
    description
      "YANG version substatement value.";
  }
  leaf prefix {
    if-feature "parsed-schema";
    type string;
    description
      "Prefix substatement value.";
  }
  container import {
    if-feature "parsed-schema";
    presence "Exists only when there is a difference in this import.";
    description
      "Import substatements.";
    leaf module {
      type yang:yang-identifier;
      mandatory true;
      description
        "Imported module.";
    }
    leaf prefix {
      type yang:yang-identifier;
      mandatory true;
      description
        "Imported module prefix.";
    }
    leaf revision-date {
      type yanglib:revision-identifier;
      description
        "Imported module revision-date.";
    }
    leaf description {
      type string;
      description

```

```
        "Description substatement value.";
    }
    leaf reference {
        type string;
        description
            "Reference substatement value.";
    }
    uses ext-instance-stmts;
}
container include {
    if-feature "parsed-schema";
    presence
        "Exists only when there is a difference in this include.";
    description
        "Include substatements.";
    leaf submodule {
        type yang:yang-identifier;
        mandatory true;
        description
            "Included submodule.";
    }
    leaf revision-date {
        type yanglib:revision-identifier;
        description
            "Included submodule revision-date.";
    }
    leaf description {
        type string;
        description
            "Description substatement value.";
    }
    leaf reference {
        type string;
        description
            "Reference substatement value.";
    }
    uses ext-instance-stmts;
}
leaf organization {
    type string;
    description
        "Organization substatement value.";
}
leaf contact {
    type string;
    description
        "Contact substatement value.";
}
```

```
leaf description {
  type string;
  description
    "Description substatement value.";
}
leaf reference {
  type string;
  description
    "Reference substatement value.";
}
container extension {
  if-feature "parsed-schema";
  presence
    "Exists only when there is a difference in this extension.";
  description
    "Extension substatements.";
  leaf name {
    type yang:yang-identifier;
    mandatory true;
    description
      "Extension name.";
  }
  leaf argument {
    type yang:yang-identifier;
    description
      "Argument substatement value.";
  }
  uses status-stmt;
  leaf description {
    type string;
    description
      "Description substatement value.";
  }
  leaf reference {
    type string;
    description
      "Reference substatement value.";
  }
  uses ext-instance-stmts;
}
container feature {
  if-feature "parsed-schema";
  presence
    "Exists only when there is a difference in this feature.";
  description
    "Feature substatements.";
  leaf name {
    type yang:yang-identifier;
```

```
    mandatory true;
    description
        "Feature name.";
}
uses if-feature-stmts;
uses status-stmt;
leaf description {
    type string;
    description
        "Description substatement value.";
}
leaf reference {
    type string;
    description
        "Reference substatement value.";
}
uses ext-instance-stmts;
}
container identity {
    description
        "Identity substatements.";
    leaf name {
        type yang:yang-identifier;
        mandatory true;
        description
            "Identity statement name.";
    }
    uses if-feature-stmts {
        if-feature "parsed-schema";
    }
    leaf-list base {
        if-feature "parsed-schema";
        type yang:yang-identifier;
        description
            "List of bases of an identity.";
    }
    uses ext-instance-stmts;
}
container deviation {
    if-feature "parsed-schema";
    presence
        "Exists only when there is a difference in this deviation.";
    description
        "Deviation substatements.";
    leaf target {
        type string;
        mandatory true;
        description
```

```
    "Deviation target argument.";
}
list deviate {
  min-elements 1;
  description
    "List of deviate substatements.";
  leaf argument {
    type enumeration {
      enum not-supported {
        description
          "Deviate 'not-supported' argument value.";
      }
      enum add {
        description
          "Deviate 'add' argument value.";
      }
      enum replace {
        description
          "Deviate 'replace' argument value.";
      }
      enum delete {
        description
          "Deviate 'delete' argument value.";
      }
    }
  }
  mandatory true;
  description
    "Deviate argument value.";
}
uses refine-deviate-node-substmts;
container type {
  description
    "Type substatement.";
  uses type-parsed-substmts;
  list union-type {
    description
      "List of a union type substatements.";
    uses type-parsed-substmts;
  }
}
leaf units {
  type string;
  description
    "Units substatement value.";
}
list unique {
  description
    "List of unique substatements.";
```

```
        leaf-list node {
            type yang:yang-identifier;
            description
                "List of nodes referenced by a unique substatement.";
        }
    }
}
leaf description {
    type string;
    description
        "Description substatement value.";
}
leaf reference {
    type string;
    description
        "Reference substatement value.";
}
uses ext-instance-stmts;
}
container ext-instance {
    presence
        "Exists only when there is a difference in this
        extension-instance.";
    description
        "Extension-instance substatements.";
    uses ext-instance-stmt;
}
}

grouping parsed-substmts {
    description
        "All parsed-only statements that can be direct substatements of a
        module or nested in a node.";
    uses if-feature-stmts {
        if-feature "parsed-schema";
    }
    uses when-stmts;
    leaf description {
        type string;
        description
            "Description substatement value.";
    }
    leaf reference {
        type string;
        description
            "Reference substatement value.";
    }
    uses status-stmt;
```



```
    uses refine-deviate-node-substmts;
    container type {
        description
            "Type substatement.";
        uses type-parsed-substmts;
        list union-type {
            description
                "List of a union type substatements.";
            uses type-parsed-substmts;
        }
    }
    leaf units {
        type string;
        description
            "Units substatement value.";
    }
    leaf presence {
        type string;
        description
            "Presence substatement value.";
    }
    uses ext-instance-stmts;
}

grouping conformance-type {
    description
        "Conformance type of a change.";
    leaf conformance {
        type enumeration {
            enum editorial {
                description
                    "Editorial change.";
            }
            enum backwards-compatible {
                description
                    "Backwards-compatible change.";
            }
            enum non-backwards-compatible {
                description
                    "Non-backwards-compatible change.";
            }
        }
    }
    mandatory true;
    description
        "Conformance information of a change in the compared YANG
        modules.";
}
}
```

```
grouping change-info {
  description
    "Details of a single change.";
  list changed {
    key "stmt";
    min-elements 1;
    description
      "List of changes.";
    leaf stmt {
      type stmt-type;
      description
        "Changed statement.";
    }
    leaf parent-stmt {
      type stmt-type;
      description
        "Parent statement of the changed statement.";
    }
    leaf change {
      type enumeration {
        enum modified {
          description
            "Statement existed and was modified.";
        }
        enum added {
          description
            "Statement was added.";
        }
        enum removed {
          description
            "Statement was removed.";
        }
        enum moved {
          description
            "Statement was moved and the position affects the
            meaning.";
        }
      }
      mandatory true;
      description
        "Type of change of the statement.";
    }
    uses conformance-type;
  }
}

grouping module-params {
  description
```

```
    "Identification details of a processed YANG module.";
  leaf module {
    type yang:yang-identifier;
    mandatory true;
    description
      "Compared module name.";
  }
  leaf revision {
    type revision-or-empty;
    mandatory true;
    description
      "Compared module revision.";
  }
  list submodule {
    key "name revision";
    leaf name {
      type yang:yang-identifier;
      mandatory true;
      description
        "Submodule name.";
    }
    leaf revision {
      type revision-or-empty;
      mandatory true;
      description
        "Submodule revision.";
    }
    description
      "Submodule of the main module.";
  }
  leaf-list enabled-feature {
    type yang:yang-identifier;
    description
      "All the enabled features of the module.";
  }
}

grouping parsed-info {
  description
    "Information about all the changed parsed-only statements.";
  list parsed-comparison {
    description
      "Information about parsed-only statement changes.";
    leaf parent-path {
      type schema-stmt-path;
      description
        "Path to the parent statement. Not set if a direct
        substatement of module or submodule.";
    }
  }
}
```

```
}
leaf identifier {
  type string;
  mandatory true;
  description
    "Identifier of the statement, may include a prefix.";
}
leaf stmt-type {
  type enumeration {
    enum choice {
      description
        "YANG statement 'choice' node.";
    }
    enum case {
      description
        "YANG statement 'case' node.";
    }
    enum input {
      description
        "YANG statement 'input' node.";
    }
    enum output {
      description
        "YANG statement 'output' node.";
    }
    enum uses {
      description
        "YANG statement 'uses' node.";
    }
    enum grouping {
      description
        "YANG statement 'grouping' node.";
    }
    enum augment {
      description
        "YANG statement 'augment' node.";
    }
    enum refine {
      description
        "YANG statement 'refine' node.";
    }
    enum typedef {
      description
        "YANG statement 'typedef'.";
    }
  }
  mandatory true;
  description
```

```
        "Type of the changed data-definition statement.";
    }
    uses change-info;
    container old {
        description
            "Changed parsed statements in the older revision of the YANG
            module.";
        uses parsed-substmts;
    }
    container new {
        description
            "Changed parsed statements in the newer revision of the YANG
            module.";
        uses parsed-substmts;
    }
}

grouping node-info {
    description
        "Information about all the changed compiled nodes.";
    list node-comparison {
        key "node";
        ordered-by user;
        description
            "Information about data-definition (node) statement substatement
            changes. Keeps the traversed order of the nodes by the
            algorithm. It SHOULD follow the depth-first search order.";
        leaf node {
            type schema-node-path;
            description
                "Path to the changed data-definition statement.";
        }
        leaf node-type {
            type enumeration {
                enum container {
                    description
                        "YANG statement 'container' node.";
                }
                enum leaf {
                    description
                        "YANG statement 'leaf' node.";
                }
                enum leaf-list {
                    description
                        "YANG statement 'leaf-list' node.";
                }
                enum list {
```

```
        description
            "YANG statement 'list' node.";
    }
    enum anydata {
        description
            "YANG statement 'anydata' node.";
    }
    enum anyxml {
        description
            "YANG statement 'anyxml' node.";
    }
    enum rpc {
        description
            "YANG statement 'rpc' node.";
    }
    enum action {
        description
            "YANG statement 'action' node.";
    }
    enum notification {
        description
            "YANG statement 'notification' node.";
    }
    }
    mandatory true;
    description
        "Type of the changed data-definition statement.";
    }
    uses change-info;
    container old {
        description
            "All the node substatements in the older revision of the YANG
            module.";
        uses node-substmts;
    }
    container new {
        description
            "All the node substatements in the newer revision of the YANG
            module.";
        uses node-substmts;
    }
    }
}

sx:structure schema-comparison {
    description
        "Schema comparison details.";
    list schema {
```

```
description
  "Instance of a schema comparison of 2 YANG modules in different
  revisions with all their imports.";
container source {
  description
    "Source module information.";
  uses module-params;
}
list source-import {
  key "module revision";
  description
    "List of source module imported modules information.";
  uses module-params;
}
container target {
  description
    "Target module information.";
  uses module-params;
}
list target-import {
  key "module revision";
  description
    "List of target module imported modules information.";
  uses module-params;
}
uses conformance-type;
list module-comparison {
  description
    "Information about direct module statement substatement
    changes.";
  uses change-info;
  container old {
    description
      "Changed statements in the older revision of the YANG
      module.";
    uses module-substmts;
  }
  container new {
    description
      "Changed statements in the newer revision of the YANG
      module.";
    uses module-substmts;
  }
}
uses parsed-info {
  if-feature "parsed-schema";
}
uses node-info;
```

```
    }  
  }  
}  
<CODE ENDS>
```

## Appendix B. Comparison Examples

This section provides various examples of the output of the comparison algorithm defined in this document to illustrate it on actual YANG modules.

### B.1. Relaxed Length Module

An example module comparison with a typedef length restriction relaxed and a new leaf added.

The generated diff includes both parsed and compiled changes. In the parsed comparison the specific range change is reported as backwards-compatible since the restriction was loosened. In the compiled node comparison the exact same change is reported because the leaf is using the same typedef, whose change also affects the leaf. The new added leaf is also reported and marked as backwards-compatible.

```
<CODE BEGINS> file "mod@2025-01-01.yang"  
module mod {  
  namespace "urn:mod";  
  prefix m;  
  yang-version 1.1;  
  
  revision 2025-01-01;  
  
  typedef my-string {  
    type string {  
      length "1..10";  
    }  
  }  
  
  container cont {  
    leaf l {  
      type my-string;  
    }  
  }  
}  
<CODE ENDS>
```



```
<CODE BEGINS> file "mod@2025-06-01.yang"
module mod {
  namespace "urn:mod";
  prefix m;
  yang-version 1.1;

  revision 2025-06-01;
  revision 2025-01-01;

  typedef my-string {
    type string {
      length "1..20";
    }
  }

  container cont {
    leaf l {
      type my-string;
    }

    leaf l2 {
      type int32;
    }
  }
}
<CODE ENDS>
```

```
<CODE BEGINS> file "mod_comparison.json"
{
  "ietf-yang-schema-comparison:schema-comparison": {
    "schema": [
      {
        "source": {
          "module": "mod",
          "revision": "2025-01-01"
        },
        "target": {
          "module": "mod",
          "revision": "2025-06-01"
        },
        "conformance": "backwards-compatible",
        "parsed-comparison": [
          {
            "parent-path": "/",
            "identifier": "my-string",
            "stmt-type": "typedef",
            "changed": [
              {
```

```
        "stmt": "length",
        "parent-stmt": "typedef",
        "change": "modified",
        "conformance": "backwards-compatible"
    }
],
"old": {
    "type": {
        "name": "string",
        "length": {
            "restriction": "1..10"
        }
    }
},
"new": {
    "type": {
        "name": "string",
        "length": {
            "restriction": "1..20"
        }
    }
}
],
"node-comparison": [
    {
        "node": "/mod:cont/1",
        "node-type": "leaf",
        "changed": [
            {
                "stmt": "length",
                "change": "modified",
                "conformance": "backwards-compatible"
            }
        ]
    },
    "old": {
        "status": "current",
        "config": true,
        "mandatory": false,
        "type": {
            "base-type": "string",
            "length": {
                "interval": [
                    {
                        "min": "1",
                        "max": "10"
                    }
                ]
            }
        }
    }
]
```

```

    }
  },
  "new": {
    "status": "current",
    "config": true,
    "mandatory": false,
    "type": {
      "base-type": "string",
      "length": {
        "interval": [
          {
            "min": "1",
            "max": "20"
          }
        ]
      }
    }
  },
  {
    "node": "/mod:cont/l2",
    "node-type": "leaf",
    "changed": [
      {
        "stmt": "node",
        "change": "added",
        "conformance": "backwards-compatible"
      }
    ],
    "new": {
      "status": "current",
      "config": true,
      "mandatory": false,
      "type": {
        "base-type": "int32"
      }
    }
  }
]
}
}
<CODE ENDS>

```

Authors' Addresses

Per Andersson (editor)  
Ionio Systems  
Email: per.ietf@ionio.se

Robert Wilton  
Cisco Systems, Inc.  
Email: rwilton@cisco.com

Michal Vásko  
CESNET  
Email: mvasko@cesnet.cz