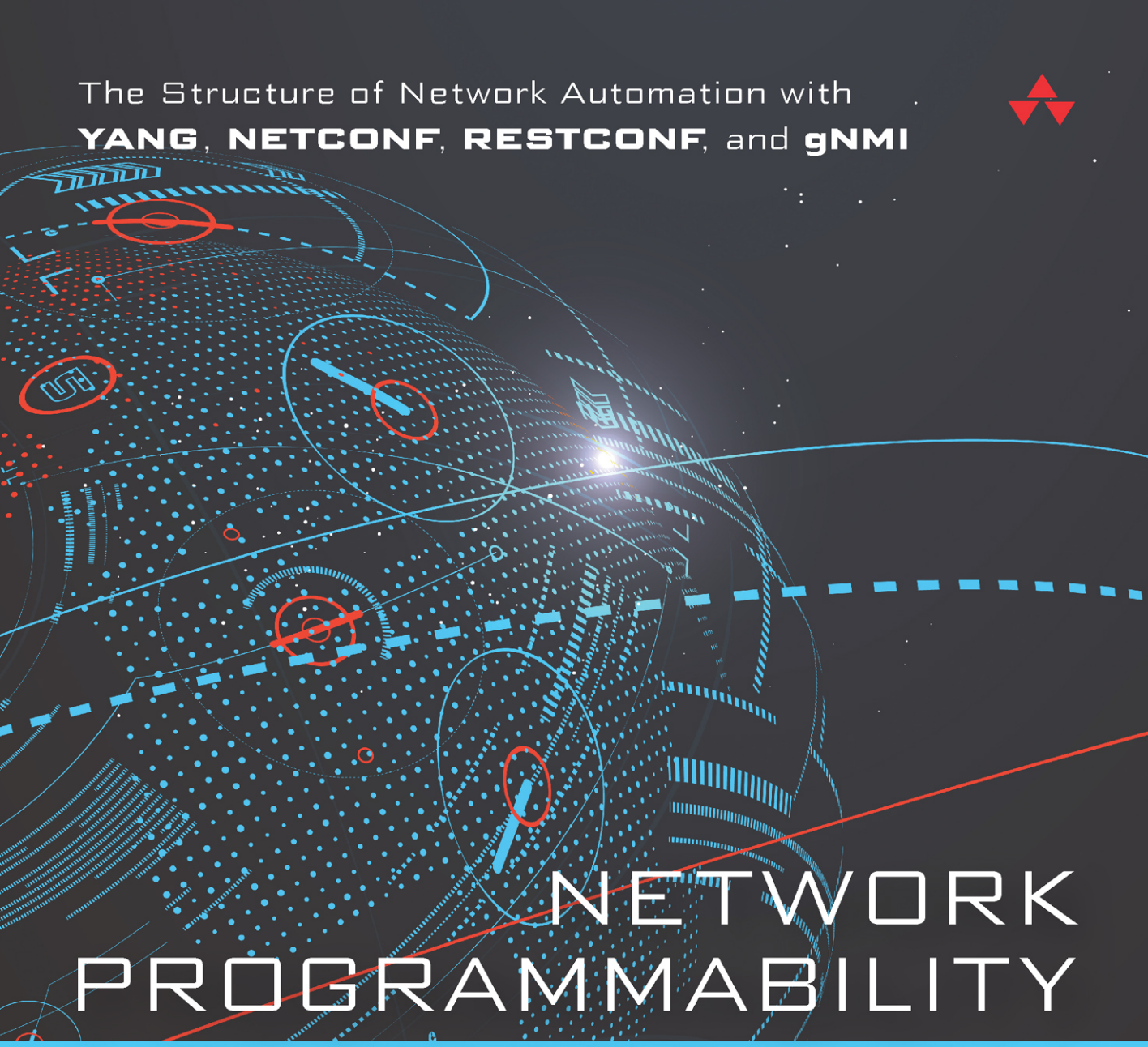
The Structure of Network Automation with
YANG, **NETCONF**, **RESTCONF**, and **gNMI**



NETWORK PROGRAMMABILITY WITH YANG

BENOIT CLAISE | JOE CLARKE | JAN LINDBLAD

Network Programmability with YANG

Use the instance data diagram in Example 3-31 to follow the navigation in Example 3-30. An instance data diagram is a cross between the YANG tree diagram shown earlier and a database dump. Here you see some sample database data in the YANG tree structure.

If, for example, the user specified "The Neverending Story" for *title*, the leafref expression so far would point to */books/book[title="The Neverending Story"]/title* (see #4).

However, you want it to point to */books/book[title="The Neverending Story"]/format/format-id*, so you go up one level (#5), then into list *format* (#6), and finally into leaf *format-id* (#7).

Now the YANG model is safe and sound. Since you updated the grouping, the fix immediately gets to all three places where you reference a *title* and *format*. This is the beauty of the grouping concept.

It is a very common YANG modeling mistake to think that a relative path and an absolute path pointing to the same leaf are equivalent. As demonstrated in Example 3-31, in YANG, they often are not. A relative path such as *deref(../title)/../format/format-id* does not evaluate to the same set of values as */books/book/title/format/format-id*, even though they point to the same *format-id* leaf, unless some XPath predicates (filters) are added.

Schema Nodes Don't Count

Another one of the most common errors when constructing or walking XPath expressions is to mistakenly pay attention to schema nodes that are not data nodes. When you're constructing the path in an XPath expression, only data nodes should be counted and referenced. Basically, there are a bunch of keywords in a YANG schema that must be disregarded in paths since they work on a "meta level." Think of them as macros, if that helps. These include `input`, `output`, `choice`, `case`, `grouping`, and `uses`.

To illustrate this point, add some payment methods to */users/user* and a way to specify which payment method to use in the *purchase* action, as shown in Example 3-32.

EXAMPLE 3-32 Leafref path Crossing a Schema Node

```
container users {
  list user {
    key user-id;

    leaf user-id {
      type string;
    }
    leaf name {
      type string;
    }
    container payment-methods {
      list payment-method {
        key "method id";
```

```
      leaf method {
        type enumeration {
          enum bookzone-account;
          enum credit-card;
          enum paypal;
          enum klarna;
        }
      }
      leaf id {
        type string;
      }
    }
  }
  action purchase {
    input {
      must "deref(format)/../price>0" {
        error-message "Item not orderable at this time";
      }
      uses title-format-ref;
      leaf number-of-copies {
        type uint32 {
          range 1..999;
        }
        default 1;
      }
      container payment {
        leaf method {
          type leafref {
            path ../../../payment-methods/payment-method/method;
          }
        }
        leaf id {
          type leafref {
            path "../../../payment-methods/"+
                 "payment-method[method=current()/../method]/id";
            // The path above expressed using deref():
            // path deref(../method)/../id;
          }
        }
      }
    }
```

How to construct that leafref path in leaf *method*? The first .. is used to get out of leaf *method*. The second .. is used to get out of container *payment*. Then you have the keyword `input`, which does not count for the path, as it is not a data node. The third .. is thus used to get out of action *purchase*.

From there, you dive into container *purchase-methods*, into list *purchase-method*, and finally into leaf *method*.

In order to verify that the XPath expressions are correct, it is strongly advised that you use proper tools. Sadly, many YANG tools (compilers) historically have never checked the XPath expressions for correctness. Some still don't. This has resulted in many YANG modelers believing that their modules were good since the compiler was happy, when in fact they still contained a good deal of broken XPath.

The final, complete model is shown in Example 3-33.

EXAMPLE 3-33 Complete bookzone-example YANG Module with Revision 2018-01-05

```
module bookzone-example {
  yang-version 1.1;
  namespace 'http://example.com/ns/bookzone';
  prefix bz;

  import ietf-yang-types {
    prefix yang;
  }

  organization
    "BookZone, a fictive book store chain";

  contact
    "YANG book project:   https://github.com/janlindblad/bookzone

     Editor:   Jan Lindblad
               <mailto:janl@tail-f.com>";

  description
    "BookZone defines this model to provide a standard interface for
     inventory browser and management applications.

     Copyright (c) 2018 the YANG book project 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 Simplified BSD License
     set forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (http://trustee.ietf.org/license-info).";

  revision 2018-01-05 {
    description
```

```
      "Added constraints that
       - author needs to have an account set before listing a book
       - number of copies in stock only shows for physical items
       - makes a book not orderable unless it has a price
       - book leafrefs are chained correctly
       Added /users/user/payment-methods and a way to choose which
       one to use in action purchase.";
  }
  revision 2018-01-04 {
    description
      "Added status information about books and purchases, see
       /books/book/popularity
       /books/book/formats/number-of-copies
       /users/user/purchase-history
       Turned reference to book title & format
       into a grouping, updated references in
       /users/user/purchase
       /shipping";
  }
  revision 2018-01-03 {
    description
      "Added action purchase and notification shipping.";
  }
  revision 2018-01-02 {
    description
      "Added book formats, authors and users, see
       /books/book/format
       /authors
       /users";
  }
  revision 2018-01-01 {
    description "Initial revision. A catalog of books.";
  }

  typedef language-type {
    type enumeration {
      enum arabic;
      enum chinese;
      enum english;
      enum french;
      enum moroccan-arabic;
      enum swahili;
      enum swedish;
      // List not exhaustive in order to save space
    }
```

```
  description
    "Primary language the book consumer needs to master "+
    "in order to appreciate the book's content";
}

identity format-idty {
  description "Root identity for all book formats";
}
identity paper {
  base format-idty;
  description "Physical book printed on paper";
}
identity audio-cd {
  base format-idty;
  description "Audiobook delivered as Compact Disc";
}
identity file-idty {
  base format-idty;
  description "Book delivered as a file";
}
identity paperback {
  base paper;
  description "Physical book with soft covers";
}
identity hardcover {
  base paper;
  description "Physical book with hard covers";
}
identity mp3 {
  base file-idty;
  description "Audiobook delivered as MP3 file";
}
identity pdf {
  base file-idty;
  description "Digital book delivered as PDF file";
}
identity epub {
  base file-idty;
  description "Digital book delivered as EPUB file";
}

typedef ISBN-10-or-13 {
  type union {
    type string {
      length 10;
```

```
      pattern '[0-9]{9}[0-9X]';
    }
    type string {
      length 13;
      pattern '97[89][0-9]{10}';
    }
  }
  description
    "The International Standard Book Number (ISBN) is a unique
     numeric commercial book identifier.

     An ISBN is assigned to each edition and variation (except
     reprintings) of a book. [source: wikipedia]";
  reference
    "https://en.wikipedia.org/wiki/International_Standard_Book_Number";
}

grouping title-format-ref {
  leaf title {
    type leafref {
      path /books/book/title;
    }
  }
  leaf format {
    type leafref {
      path /books/book[title=current()/../title]/format/format-id;
      // The path above expressed using deref():
      // path deref(../title)/../format/format-id;
    }
  }
}

container authors {
  list author {
    key name;

    leaf name {
      type string;
    }
    leaf account-id {
      type uint32 {
        range 1001..max;
      }
    }
  }
}
```