GitHub Integration and Tooling (git)
Internet-Draft
Intended status: Best Current Practice
Expires: August 29, 2019

K. Watsen
Watsen Networks

February 25, 2019

eXtract or Insert artwork And source code to/from Xml (xiax)
draft-kwatsen-git-xiax-automation-01

Abstract

This document describes motivations behind and solutions for tooling to automate the extraction/insertion of artwork and source code to/from `rfc2xml` documents.

While much may appear to be working, the author believes that, in order for such automation to be maximally useful, it is necessary to solicit broad input from the community (co-authors are welcomed, both on the draft and the tool).

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 August 29, 2019.

Table of Contents

1. Introduction

This document describes motivations behind and solutions for tooling to automate the extraction/insertion of artwork and source code to/from `rfc2xml` v2 [RFC7749] and v3 [RFC7991].

For authors, adoption of the automation ensures completely up-to-date <artwork> and <sourcecode> inclusions every time the document is published.

For reviewers (especially shepherds, doctors, and copy editors), use of the automation offers assurance that the <artwork> and <sourcecode> inclusions are syntactically valid, and the ability to quickly verify that they are when needed.

2. Applicability Statement

At the time of this writing, `rfc2xml` v3 [RFC7991] is not yet in production, and thus the tooling support described herein is intended to apply to both v2 and v3.

Whenever ambiguity may arise, this document will fully write out text such as "...source code stored in the v2 <artwork> element...".

3. Notational 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.

4. Terminology

This document uses the following terms (sorted by name):

Artwork: The term "artwork" is used throughout to represent two-dimensional imagery (e.g., ASCII art), such as would be referenced by the <artwork> element defined in Section 2.5 of [RFC7991].

Source code: The term "source code" is used throughout to represent a structured sequence of lines, such as would be referenced by the <sourcecode> element defined in Section 2.48 of [RFC7991].

5. Updates to RFC 7991

This section is just a placeholder for now, but it is expected that [RFC7991] will need to be modified in order to support some of this work.

At a minimum, [RFC7991] should be updated to support attributes from other namespaces, such that the `rfc2xml` tool would neither process nor discard them.

6. Motivation

The driving motivation for this work is twofold:

- To ensure the correctness of works in progress.
- To simplify the formal verification process.

Firstly, far too often, throughout the lifecycle of a draft, is it that authors overlook updating artwork and/or source code in their drafts, leading to confusion and wasting some of the precious little time of other working group members. While repeated encouragement from chairs and others to embrace automation, it seems that the bar is too high for some authors to bother for their one and perhaps only draft. It is actually a self-defeating strategy, as it has been shown time and again that the effort invested to do necessary script-fu would be recouped by the authors themselves over the lifetime of their draft.

Next, for formal verifications, the YANG Doctor [yang-doctors], reviews are first in mind, but the automation is equally useful for any structured syntax other than YANG [RFC6020] [RFC7950], such as ASN.1 [ITU.X690.2015] and ABNF [RFC5234] [RFC7405]. That said, publication process experience shows that doctor reviews are often out of synch with the document submitted for publication, sometimes even by several draft revisions. Thusly, it is common for the draft shepherds themselves to verify the correctness of inclusions when doing the shepherd writeup. Further, the document may be subsequently updated by IESG and/or copy editor reviews, steps for which the automation would continue to support.

7. Previous Work

- Section 3.2 of [RFC8407] states that normative YANG modules and submodules contained within Internet-Drafts and RFCs must be bracketed by <CODE BEGINS> and <CODE ENDS> markers. Section 3.1.18 of [I-D.levkowetz-xml2rfc-v3-implementation-notes] notes support for this in `xml2rfc` through the use of a `markers` attribute in the <sourcecode> element. [PS: these markers attempt to support extraction from plain-text documents but, as this document shows, extraction from XML is superior and, besides, there are many interesting things to extract beyond YANG modules.]

- The `xym` [xym] and `rfcstrip` [rfcstrip] utilities have been developed to extract YANG modules from Internet-Drafts and RFCs using the <CODE BEGINS> and <CODE ENDS> markers.

- The RFC Submit [submit] tool has been modified to test YANG modules contained within I-Ds, and the resulting document page in Datatracker [datatracker] displays a new "Yang Validation" field containing a varying color yin-yang symbol (green if no errors, red if errors) along with counts. This tool is okay for what it is, but it neither aids authors between updates nor validates anything beyond YANG modules.

- The YANG Validator site [yang-validator] provides an Internet-facing service, with a REST-based API, for validating YANG modules in drafts. Having a REST API enables its use throughout a document's lifecycle but, again, it doesn't validate anything beyond YANG modules.

8. Automated Construction

When asked to build a submittable `rfc2xml` document, the automation should perform the following steps, in order:

1. Prime artwork and source code as needed. Known priming steps include:

   i Draft revision addition/substitution for the `docName` attribute in the <rfc> element as well as in the filename.

   ii Date substitution (e.g., replacing the string "YYYY-MM-DD" with the current date. This substitution needs to occur both within files and in filenames.

   iii Generation of derived views (e.g., YANG tree diagrams [RFC8340]). Technically, the derived views should be generated after the validation (discussed next) but, said generation is rightly part of the "priming" step and, besides, if there is an error, the validation step would still catch it, so there's no harm in generating the derived views first.

2. Validate source code and artwork. This step includes:

   i Validating data models (e.g., YANG modules) against the schema describing their syntax.

   ii Validating data instance examples (e.g., a snippet of configuration) against the governing data models (e.g., the aforementioned YANG modules).

   iii Validating the derived views. Technically, this step is not needed during the insertion process, since the derived

views were just generated in the previous step but, since the same validation logic is used by automated verification (see Section 9), it is automatically executed here as well. Validating derived views is accomplished by running the script to generate the view and then comparing the result to the view extracted in the draft.

3. Pack the final submittable XML file. This step includes:

   i Pasting the contents referenced by attributes in the <artwork> and <sourcecode> elements into the XML document, and storing information enabling the content to be extracted back to its original filename.

   ii Additional attributes can control if markers are added (e.g., the <CODE BEGINS> and <CODE ENDS> markers described by RFC 8407 Section 3.2).

   iii Character data (CDATA) wrappers may be added.

   iv Folding (line wrapping) may be added, per [I-D.ietf-netmod-artwork-folding].

   v Additional date substitutions (e.g., YYYY-MM-DD) in the body of the draft may be needed.

   vi Draft revision substitution (i.e., replacing placeholder "-latest" with, e.g., "-03").

9. Automated Verification

When asked to extract/verify a submitted `rfc2xml` document, the automation should perform the following steps, in order:

1. Extract the contents of the <artwork> and <sourcecode> elements to their original file-based forms, retaining their file names and directory paths.

2. Extract any additional files that may be necessary to generate the derived views and/or validate the inclusions.

3. Optionally, if requested, also save the "primed" or "unpacked" XML file (i.e., the source XML file before the content was packed into it).

4. At this point, the local directory tree represents the "primed" state, and thus the same validation logic described above can executed again.

10. Security Considerations

10.1. Automated Execution of Arbitrary Scripts

In order to support the auto-generation of derived views and the validation of data models and data instance examples, the automation solution must not automatically execute arbitrary scripts.

A couple solutions present themselves:

- Allow arbitrary scripts, but don't execute them automatically when a document is extracted. This solution is appealing as it still ensures these scripts were executed on the author's computer at time of construction, and the scripts themselves can be extracted and audited on the reviewer's computer. If desired, after auditing a script, a reviewer could choose to manually execute it on their own computer.

- Don't allow arbitrary scripts but, instead, support parameterized files that declare all the information necessary to construct the command(s) necessary to generate derived views and/or validate inclusions.

11. References

11.1. Normative References

[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>.

[RFC7991] Hoffman, P., "The "xml2rfc" Version 3 Vocabulary", RFC 7991, DOI 10.17487/RFC7991, December 2016, <https://www.rfc-editor.org/info/rfc7991>.

[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>.

11.2. Informative References

[I-D.ietf-netmod-artwork-folding] Watsen, K., Wu, Q., Farrel, A., and B. Claise, "Handling Long Lines in Artwork in Internet-Drafts and RFCs", draft-ietf-netmod-artwork-folding-00 (work in progress), November 2018.

[I-D.levkowetz-xml2rfc-v3-implementation-notes]
Levkowetz, H., "Implementation notes for RFC7991, "The 'xml2rfc' Version 3 Vocabulary"", draft-levkowetz-xml2rfc-v3-implementation-notes-08 (work in progress), February 2019.

[ITU.X690.2015]
International Telecommunication Union, "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1, August 2015, <https://www.itu.int/rec/T-REC-X.690/>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[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>.

[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <https://www.rfc-editor.org/info/rfc7405>.

[RFC7749] Reschke, J., "The "xml2rfc" Version 2 Vocabulary", RFC 7749, DOI 10.17487/RFC7749, February 2016, <https://www.rfc-editor.org/info/rfc7749>.

[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>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

[rfcstrip]
"The `rfcstrip` GitHub Repository", <https://github.com/mbj4668/rfcstrip>.

[submit] "Datatracker Internet-Draft Submission Service",
<https://datatracker.ietf.org/submit>.

[xiax] "The `xiax` GitHub Repository",
<https://github.com/kwatsen/xiax>.

[xym] "The `xym` GitHub Repository",
<https://github.com/xym-tool/xym>.

[yang-doctors]
"The YANG Doctors "about" Page",
<https://datatracker.ietf.org/group/yangdoctors/about>.

[yang-validator]
"The YANG Validator Service",
<http://www.yangvalidator.com>.

Appendix A. Examples

This section illustrates bits working using the `xiax` tool [xiax].

This entire document has been processed by `xiax`, albeit with a little manual tweaking to return some of the "YYYY-MM-DD" strings (such as this one) back to their original forms.

Note that, as this is an `xml2rfc` v2 document, all examples use the `xml2rfc` v2 <artwork> element.

A.1. Static Inclusion

This example illustrates inclusion of static content with no additional processing, such as might be useful for pre-generated artwork.

This is what the original `xml2rfc` XML looked like:

```
<preamble>START FIGURE</preamble>
<artwork xiax:src="art/hello.txt"/>
<postamble>END FIGURE</postamble>
```

Here is the rendered content:

START FIGURE

```
 _          _ _
| |        | | |
| |__   ___| | | ___
| '_ \ / _ \ | |/ _ \
| | | |  __/ | | (_) |
|_| |_|\___|_|_|\___/
```

END FIGURE

A.2. Static Inclusion and Date Substitution

This example illustrates inclusion of static content along with date substitution. Date substitution is triggered by the string "YYYY-MM-DD" appearing in the name of the file being included. The date-substitution is applied to both the filename as well as to the content of the file.

This is what the original `xml2rfc` XML looked like:

```
<preamble>START FIGURE</preamble>
<artwork xiax:src="art/email-YYYY-MM-DD.txt"/>
<postamble>END FIGURE</postamble>
```

Here is the rendered content:

START FIGURE

```
+--------------------------------+
| To: all                        |
| Date: 2019-02-25               |
| Subject: hello world           |
|                                |
| ...                            |
+--------------------------------+
```

END FIGURE

The content of the original "src" file (before date-substitute):

```
+--------------------------------+
| To: all                        |
| Date: YYYY-MM-DD               |
| Subject: hello world           |
|                                |
| ...                            |
+--------------------------------+
```

A.3. Generated Inclusion and Date Substitution

This example illustrates both inclusion of generated content along with date substitution.

This is what the original `xml2rfc` XML looked like:

```
<preamble>START FIGURE</preamble>
<artwork xiax:gen="xiax/gen-foo-tree-diagram@YYYY-MM-DD.xml"/>
<postamble>END FIGURE</postamble>
```

Here is the rendered content:

START FIGURE

```
module: foo
  +--rw foo?   empty
```

END FIGURE

The content of the "gen" file being referenced is:

```
<generate xmlns="https://watsen.net/xiax" version="1">
  <yang-tree-diagram>
    <source>foo@YYYY-MM-DD.yang</source>
  </yang-tree-diagram>
</generate>
```

A.4. Static Inclusion, Date Substitution, and Validation

This example illustrates inclusion of static content with date substitution and behind-the-scenes validation. Note that, in this example, the date substitution occurs on the validation input file (since it references a date-substituted file).

This is what the original `xml2rfc` XML looked like:

```
<preamble>START FIGURE</preamble>
<artwork xiax:src="examples/ex-foo.json" xiax:val="xiax/val-xml-ex-foo@YYYY-MM-
DD.xml"/>
<postamble>END FIGURE</postamble>
```

Here is the rendered content:

START FIGURE

```
{
  "foo:foo": [null]
}
```

END FIGURE

The content of the "val" file being referenced is:

```
<validate xmlns="https://watsen.net/xiax">
  <xml-document>
    <using-yang>
      <yang-modules>
        <yang-module>
          <name>foo@YYYY-MM-DD.yang</name>
          <uri>foo@YYYY-MM-DD.yang</uri>
        </yang-module>
      </yang-modules>
    </using-yang>
  </xml-document>
</validate>
```

A.5. Static Inclusion, Date Substitution, Markers, and Validation

This example illustrates inclusion of static content with date substitution, <BEGIN CODE> and <END CODE> markers, and behind-the-scenes validation.

This is what the original `xml2rfc` XML looked like:

```
<preamble>START FIGURE</preamble>
<artwork xiax:src="foo@YYYY-MM-DD.yang" xiax:markers="true" xiax:val="xiax/val-
yang-foo@YYYY-MM-DD.xml"/>
<postamble>END FIGURE</postamble>
```

Here is the rendered content:

START FIGURE

<CODE BEGINS> file "foo@2019-02-25.yang"

```
module foo {
  yang-version 1.1;
  namespace "https://example.com/foo";
  prefix "f";

  revision "2019-02-25" {
    description
     "Initial version";
  }

  leaf foo {
    type empty;
  }
}
```

<CODE ENDS>

END FIGURE

The content of the "val" file being referenced is:

```
<validate xmlns="https://watsen.net/xiax">
  <xml-document>
    <using-yang>
      <yang-modules>
        <yang-module>
          <name>foo@YYYY-MM-DD.yang</name>
          <uri>foo@YYYY-MM-DD.yang</uri>
        </yang-module>
      </yang-modules>
    </using-yang>
  </xml-document>
</validate>
```

Appendix B. Details for the `xiax` Utility

B.1. The "xiax-block" Comment

In order to support extractions, `xiax` needs to encode metadata into the `xml2rfc` XML file. This metadata encodes, for instance, the original names of files, contents of the 'gen' and 'val' files, and and additional files that may have been used by the 'gen' and/or 'val' files.

There are limited options for encoding metadata into the `xml2rfc` format in a way that doesn't generate errors or is discarded during the Internet-Draft submission process.

The only option that was discovered is to encode the metadata into an XML comment. Thus, `xiax` adds a special XML comment referred to as the "xiax-block" to the end of the `xml2rfc` XML file.

An example xiax-block follows:

```
<!-- FANTASTIC RFC CONTENT ABOVE THIS LINE -->

  </back>
<!-- ##xiax-block-v1:
H4sIAMcX/82WTW/iMBCG7/wKlruZQsuhKzcqWq697aVCHEwYwMWxke206b/vOB8lgUhdsrDb
nJzxTOad8EyKjC2ViXf9LFHaPQy23u/dT4A34R3qoUYPIWkQ9fp9LnWsUieNDnd0vxd+Gy1F
vAOHsaf4L/38dgFruUktgrD+zdgdh7yoqHc2Lla0Ft5buYwoDbaolBn6zHMoo0U6lPkcGkLO
kzXuLAsT34zu2c2YjSdXVXj3nRXefu3hBvWxwsARUJytjWHeIrKVFBsrkseaYsKwqZjq11Jh
xH/Mf82m8Gi0wuNXyPZf0YaJHgajAt93oTeNxpWbJrUxRiSrriRkk13FXu5YSz2HSk3UWywi
DrnW0urS+mwYull9GTjuOr1gmIlkr9ABZkH08MUZfQLCq1DV+mhW2mE0DyuqH5/pYk9PbDZr
A+oUKSqXKncj9FmZOE1Q+3IgMofQCHx8as1hScwqpaFqjWvhQ5TiWiQFgTXpBYH5Tj01tbI9
M2wcOkFLVafeVM+hOSCHyp0TxEPy55lckJ5uvB8xcH3SJ51Ib/3MnAV6foZXx7xGCvxrCib/
kQIOh7+isdFI8SMJAAA=

-->

</rfc>
```

Note that this data is the base64-encoding of the GZIP-ed string for the encoded metadata.

Also note that the xiax-block is versioned. The intention is that, while a given version of `xiax` will only produce the "current" xiax-block version, it should be capable of extracting content from a draft produced by any prior version.

B.2. Extensible Support to Content Types

Before diving into the "xiax-block", and in the interest of full disclosure, it should be known that `xiax` currently has limited support for content types. Specifically:

- For generating content, `xiax` currently only knows how to generate YANG tree diagrams [RFC8340].

- For validating content, `xiax` currently only knows how to validate YANG modules [RFC7950] and XML/JSON documents against YANG schema.

However, the code has been developed anticipating a desire to extend it to support other content types. And, being an open source project on GitHub, it is hoped that others will take interest to add support for additional content types.

B.3. The "xiax-block" Data Model

Being that `xiax` operates on XML files, it was intuitive to use XML to encode the xiax-block.

In IETF fashion, the schema for the XML data model is defined using YANG [RFC7950].

B.3.1. Tree Diagram

Following is the YANG Tree Diagram [RFC8340] for the xiax-block:

```
   module: xiax-structures-v1

     yang-data xiax-block:
       +-- xiax-block
          +-- inclusion* [path]
             +-- path    string
             +-- src
             |  +-- attrib?   inet:uri
             |  +-- val
             |     +-- attrib?   inet:uri
             |     +-- file?     <anydata>
             +-- gen
                +-- attrib?   inet:uri
                +-- file?     <anydata>
                +-- val
                   +-- attrib?   inet:uri
                   +-- file?     <anydata>
     yang-data generate:
       +-- generate
          +-- (generate-type)?
             +--:(yang-tree-diagram)
                +-- yang-tree-diagram
                   +-- source?            string
                   +-- print-yang-data?   empty
     yang-data validate:
       +-- validate
          +-- (content-type)?
             +--:(yang-module)
             |  +-- yang-module
             |     +-- additional-yang-modules
             |        +-- additional-yang-module* [name]
             |           +-- name    string
             |           +-- uri*    inet:uri
             +--:(xml-document)
                +-- xml-document
                   +-- (schema-type)?
                   |  +--:(using-yang)
                   |     +-- using-yang
                   |        +-- yang-modules
                   |           +-- yang-module* [name]
                   |              +-- name    string
                   |              +-- uri*    inet:uri
                   +-- additional-xml-documents
                      +-- additional-xml-document* [name]
                         +-- name    string
                         +-- uri*    inet:uri
```

B.3.2. YANG Module

Following is the YANG module for the xiax-block follows:

```
module xiax-structures-v1 {
  yang-version 1.1;
  namespace "https://watsen.net/xiax";
  prefix "xb";

  import ietf-inet-types {
    prefix inet;
    reference "RFC 6991: Common YANG Data Types";
  }

  import ietf-restconf {
    prefix rc;
    reference "RFC 8040: RESTCONF Protocol";
  }

  organization "Watsen Networks";

  contact "Kent Watsen <mailto:kent+ietf@watsen.net>";

  description
   "This module defines the data model for xiax data block.

    Copyright (c) 2019 Watsen Networks.  All rights reserved.";


  revision "2019-02-25" {
    description
     "Initial version";
  }

  grouping val-grouping {
    container val {
      leaf attrib {
        type inet:uri;
        description
          "The original 'xiax:val' attribute that was in the
           <sourcecode> element (<artwork> cannot be validated).";
      }
      anydata file {
        description
          "The content of the file per the 'xiax:val' attribute";
      }
    }
  }
```

```
  rc:yang-data "xiax-block" {
    container xiax-block {

      description
        "Contains lists of inclusions that were processed by `xiax`
         during its 'packing' step.";

      list inclusion {
        key path;

        description
          "A list of inclusions, one for each <artwork> and/or
           <sourcecode> element processed by `xiax`.";

        leaf path {
          type string;
          description
            "The DOM path of the <artwork> or <sourcecode> element.";
        }

        container src {
          leaf attrib {
            type inet:uri;
            description
              "The original 'xiax:src' attribute that was in the
               <artwork> or <sourcecode> element.";
          }
          uses val-grouping;
        }

        container gen {
          leaf attrib {
            type inet:uri;
            description
              "The original 'xiax:gen' attribute that was in the
               <artwork> or <sourcecode> element.";
          }
          anydata file {
            description
              "The content of the file per the 'xiax:gen' attribute";
          }
          uses val-grouping;
        }


      } // end list inclusion
    } // end container xiax-block
  } // end rc:yang-data xiax-block
```

```
  rc:yang-data "generate" {
    container generate {

      description
        "Contains instructions to `xiax` for how to generate content";

      choice generate-type {
        description
          "The type of content to generate, and information for how
           to do so.";

        container yang-tree-diagram {
          leaf source {
            type string;
            description
              "The YANG file to generate the tree-diagram from.";
          }
          leaf print-yang-data {
            type empty;
          }
        }

        /*** add more gen-types here ***/

      } // end choice gen-type
    } // end container xiax-block
  } // end rc:yang-data generate




  rc:yang-data "validate" {
    container validate {

      description
        "Contains information for how to validate content.  Currently
         just the list of modules and ";

      choice content-type {
        description
          "The type of content to validate, and information for how
           to do so.";

        container yang-module {
          description
            "Provides information for how to validate the YANG module.";
          container additional-yang-modules {
            description
```

```
            "Additional YANG documents that may be needed in order to
             resolve, e.g., import statements.   Do not include the
             YANG module being validated.";
          list additional-yang-module {
            key name;
            leaf name {
              type string;
            }
            leaf-list uri {
              type inet:uri;
              description
                "Location for where the YANG module is located.
                 Multiple URIs are used to address availability
                 concerns.  A copy of files referenced using the
                 'file' schema is embedded into the xiax-block.
                 A file will only be stored into the xiax-block
                 at most once, in case it referenced by more
                 than one validation.";
            }
          } // end list additional-yang-module
        } // end container additional-yang-modules
      } // end container yang-module

      container xml-document {
        description
          "Provides information for how to validate a XML document.";

        choice schema-type {
          description
            "Enables the schema-type to be selected.";

          container using-yang {
            description
              "Provides information for how to validate the XML
               document using YANG.";

            container yang-modules {
              list yang-module {
                key name;
                  leaf name {
                  type string;
                }
                leaf-list uri {
                  type inet:uri;
                  description
                    "Location for where the YANG module is located.
                     Multiple URIs are used to address availability
                     concerns.  A copy of files referenced using the
```

```
                    'file' schema is embedded into the xiax-block.
                    A file will only be stored into the xiax-block
                    at most once, in case it referenced by more
                    than one validation.";
                }
              } // end list yang-module
            } // end container yang-modules
          } // end container using-yang

        /*** add other XML-validating schema-types here ***/

        } // end choice schema-type

        container additional-xml-documents {
          description
            "Additional XML documents that may be needed in order to
             resolve, e.g., data references.   Do not include the XML
             document being validated.";

          list additional-xml-document {
            key name;
            leaf name {
              type string;
            }
            leaf-list uri {
              type inet:uri;
              description
                "Location for where the XML document is located.
                 Multiple URIs are used to address availability
                 concerns.  A copy of files referenced using the
                 'file' schema is embedded into the xiax-block.
                 A file will only be stored into the xiax-block
                 at most once, in case it referenced by more
                 than one validation.";
            }
          } // end additional-xml-document
        } // end container additional-xml-documents
      } // end container xml-document

      /*** add content-types here ***/

    } // end choice content-type
  } // end container validate
} // end rc:yang-data validate

} // end module xiax-structures-v1
```

B.4. Examples

B.4.1. A peak inside the "xiax-block"

The following is a snippet of the xiax-block used in this draft.

Thie example illustrates three inclusions:

1. A "xiax:src" attribute.

2. A "xiax:gen" attribute, including the gen-file itself.

3. Both "xiax:src" and "xiax:val" attributes, including the val-file itself.

```
<?xml version="1.0"?>
<xiax-block xmlns="https://watsen.net/xiax">
  <inclusion>
    <path>back/section[1]/section[1]/t[3]/figure/artwork</path>
    <src>
      <attrib>hello.txt</attrib>
    </src>
  </inclusion>
  <inclusion>
    <path>back/section[1]/section[3]/t[3]/figure/artwork</path>
    <gen>
      <attrib>xiax/gen-foo-tree-diagram@2019-02-25.xml</attrib>
      <file>
        <generate xmlns="https://watsen.net/xiax" version="1">
          <yang-tree-diagram>
            <source>foo@2019-02-25.yang</source>
          </yang-tree-diagram>
        </generate>
      </file>
    </gen>
  </inclusion>
  <inclusion>
    <path>back/section[1]/section[5]/t[4]/figure/artwork</path>
    <src>
      <attrib>examples/ex-foo.xml</attrib>
      <val>
        <attrib>xiax/val-xml-ex-foo@2019-02-25.xml</attrib>
        <file>
          <validate xmlns="https://watsen.net/xiax">
            <xml-document>
              <using-yang>
                <yang-modules>
                  <yang-module>
                    <name>foo@2019-02-25.yang</name>
                    <uri>foo@2019-02-25.yang</uri>
                  </yang-module>
                </yang-modules>
              </using-yang>
            </xml-document>
          </validate>
        </file>
      </val>
    </src>
  </inclusion>
</xiax-block>
```

B.4.2. A "gen" file (for a tree diagram)

This example shows what the "gen" file for generating a YANG tree diagram looks like.

```
<generate xmlns="https://watsen.net/xiax">
  <yang-tree-diagram>
    <source>xiax-block-v1@YYYY-MM-DD.yang</source>
  </yang-tree-diagram>
</generate>
```

B.4.3. A "val" file (for an YANG module)

This example shows what the "val" file for validating a YANG module looks like.

```
<validate xmlns="https://watsen.net/xiax">
  <yang-module>
    <additional-yang-modules>
      <additional-yang-module>
        <name>ietf-restconf@2017-01-26.yang</name>
        <uri>https://raw.githubusercontent.com/YangModels/yang/master/standard/
ietf/RFC/ietf-restconf%402017-01-26.yang</uri>
      </additional-yang-module>
    </additional-yang-modules>
  </yang-module>
</validate>
```

B.4.4. A "val" file (for an XML document)

This example shows what the "val" file for validating an XML document looks like.

```
<validate xmlns="https://watsen.net/xiax">
  <xml-document>
    <using-yang>
      <yang-modules>
        <yang-module>
          <name>xiax-block-v1.yang</name>
          <uri>./xiax-block-v1.yang</uri>
        </yang-module>
        <yang-module>
          <name>ietf-restconf@2017-01-26.yang</name>
          <uri>https://raw.githubusercontent.com/YangModels/yang/master/
standard/ietf/RFC/ietf-restconf%402017-01-26.yang</uri>
        </yang-module>
      </yang-modules>
    </using-yang>
  </xml-document>
```

```
</validate>
```

Author's Address

Kent Watsen
Watsen Networks

EMail: kent+ietf@watsen.net