Workgroup: Network Working Group
Published: 7 February 2023
Intended Status: Experimental
Expires: 11 August 2023
Authors: M. Petit-Huguenin
Impedance Mismatch LLC

Computerate Specification

Abstract

This document specifies computerate specifications, which are the combination of a formal and an informal specification such as parts of the informal specification are generated from the formal specification.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

An informal specification is any document that is the combination of text, diagrams, tables, lists, and examples and whose purpose is to describe an engineering system. An RFC is an example of informal specification when it describes network elements and the protocols that help these elements to interoperate.

Informal specifications are nowadays written on a computer, which makes them easier to write but not easier to reason about their correctness. Computers can be used to model the engineering system that a specification is meant to describe, to verify that some properties hold for that model, and to automatically derive parts of the text, diagrams, list and examples that compose that specification. This could be done in an ad-hoc way by copying and pasting results from external software, but this adds the burden for the specification authors of having to keep the software model and the specification synchronized.

A well-known consequence is when an example in a specification does not match the text, most likely because one of the two was modified without verifying that the other was updated.

To simplify the integration of code in a specification we introduce the concept of "Computerate Specifying", which is similar in appearance to literate programming [LitProg], but reversing the intent. Literate programming is about interspersing code with the elements of a document, with the intent of producing a piece of code that can then be executed. On the other hand computerate specifying is about interspersing a document with code, with the intent of producing a document that have some of its elements generated from the result of computations that uses that code.

"Computerate Specifying" is a play on "Literate Computing", itself a play on "Structured Computing" (see [Knuth92] page 99). Note that "computerate" is a British English word. To keep on the joke, an informal specification could be said to be an incomputerate specification, "incomputerate" being an Australian English word.

2. Terminology

AsciiDoc: The formal language in which the document part of a computerate specification is written.

Asciidoc: An Idris2 package that is used to generate AsciiDoc fragments.

Asciidoctor: An extensible processor for AsciiDoc document.

3. AsciiDoc

There is a large variety of target formats in which a specification can be written (HTML, PDF, Word, Confluence), so we use [AsciiDoc] as a common format from which any of the target formats can be generated.

AsciiDoc is a text format, making it easy to mix with code and to integrate with the tools that programmers are most familiar with. It has been chosen because it is derived from DocBook and because its main implementation, [Asciidoctor], can be easily extended and already have a large set of backends for various target formats:

* The HTML5, DocBook5, and Manpage back-ends are built-in.

* The PDF, Epub3, Reveal.js (slides) and many others are distributed as add-ons.

* [Metanorma] distributes a set of back-end add-ons for the development of standard documents.

* A alternate XML2RFC v3 [draft-petithuguenin-xml2rfc-asciidoc] backend add-on is also available.

* A Confluence backend add-on is under development.

The AsciiDoc syntax is enriched with several extensions designed to support computerate specifications:

* the support for literate programming, by commenting out any line that starts with the '>' character in the leftmost column
* an inline macro that inserts the result of a computation as AsciiDoc formatted text
* a block macro that inserts the result of a computation as AsciiDoc blocks

These syntax extensions, which are described in the following sections, are implemented in the "asciidoctor-idris2" add-on.

3.1. Literate Programming

Idris2 shares with Haskell and other Haskell relatives built-in support for literate programming. We use the Bird mode which marks lines of code with a '>' symbol in the leftmost column. The literate programming rule that states that a block of code must always be separated from normal text by at least one blank line applies.

The following example contains a block of two lines of Idris2 code:

```
Some text

> a : Int
> a = 2

More text
```

Note that the '>' symbol is also used by AsciiDoc as an alternate way of defining a blockquote which is consequently not available in a computerate specification. The normal syntax for blockquotes must be used instead.

The code will not appear in the rendered document, but self-inclusion can be used to copy part of the code into the document, thus guaranteeing that this code is verified by the type-checker:

```
> -- tag::currying[]
> total
> currying : ((a, b) -> c) -> (a -> b -> c)
> currying f = \x => \y => f (x, y)
> -- end::currying[]

....
include::myfile.lidr[tag=currying]
....
```

Because Idris2 does not resolve forward references by default, self-inclusion can also be used to reorder paragraphs when the flow of explanations does not coincide with the flow of the code.

Alternating paragraphs of text and code permits to keep both representations as close as possible and is an effective way to quickly discover that the code and the text are diverging. The convention is to insert the code beneath the text it is related to.

To be treated as a literate programming file, the AsciiDoc document must use the suffix ".lidr". This generally means that a document will be split in multiple documents, some with the ".adoc" extension and some with the ".lidr" extension. These files are put back together by using, directly or indirectly, AsciiDoc "include" statements in the files.

3.2. Code Macros

Using code macros in lieu of the equivalent constants in a document ensures that the text stays consistent during the development of the specification.

The type of the code passed to the macro must implement the Asciidoc interface so it is converted into properly escaped Asciidoc. Section 4.1.3 explains how to implement the Asciidoc interface for user defined types.

A code macro searches first for an Asciidoc implementation that has the same name than the back-end, then fallback to an unnamed implementation when a named implementation cannot be found. This permits to use all the extensions specific to a particular back-end.

3.2.1. Inline Code Macro

The code2:[] inline macro is used when the result is to be inserted as AsciiDoc inline text.

For instance the following excerpt taken from the computerate specification of [RFC8489]:

```
> retrans' : Nat -> Int -> Maybe (List1 Int)
> retrans' rc = fromList . take rc . scanl (+) 0
>   . unfoldr (bimap id (*2) . dup)

> retrans : Nat -> Int -> String
> retrans rc = maybe "Error"
>    (foldr1By (\e, a => show e ++ " ms, " ++ a)
>      (\x => "and " ++ show x ++ "ms")) . retrans' rc

> timeout : Nat -> Int -> Int -> String
> timeout rc rto rm = maybe "Error"
>    (\e => show (last e + rm * rto)) (retrans' rc rto)

> rc : Nat; rc = 7
> rto : Int; rto = 500
> rm : Int; rm = 16

For example, assuming an RTO of code:[rto]ms, requests would be
sent at times code:[retrans rc rto].
If the client has not received a response after code:[timeout] ms,
the client will consider the transaction to have timed out.
```

is rendered as

"For example, assuming an RTO of 500ms, requests would be sent at times 0 ms, 500 ms, 1500 ms, 3500 ms, 7500 ms, 15500 ms, and 31500ms. If the client has not received a response after 39500 ms, the client will consider the transaction to have timed out."

3.2.2. Block Code Macro

The "code::[]" block macro (notice the double colon) is used to generate AsciiDoc blocks in a way similar to the inline code macro.

4. Idris2

The code in a computerate specification uses the programming language [Idris2] in literate programming [Knuth92] mode. Although most programming languages could have been used, Idris2 has been chosen for the following features:

*purely functional
*eager evaluation, with optional laziness
*totality checking
*dependent and linear type system.
*reflection and meta-programming support
*REPL.

The most important of these features are totality checking and the dependent type system, which permit to ensure an high level of correctness to the code. Generating portions of a document from a programming language that lacks these features would only bring marginal improvements in the quality of a specification.

The next sections describe an Idris2 package that can be used to simplify the generation of IdrisDoc. Appendix B lists the API for that same package.

4.1. "Asciidoc" Module

The Asciidoc module provides a way to programmatically build an AsciiDoc fragment without having to worry about the particular formatting details.

The elements of an AsciiDoc document are grouped in 4 categories:

*Content
*Block
*Top-level section
*Document

4.1.1. Content

"Content" is the sum type of the possible parts that compose an AsciiDoc text:

"TextContent": Plain text.
"BreakContent": An hard line break.
"ItalicContent": Italicized text.
"LinkContent": An hyperlink.
"IndexContent": A word that will be cross-referenced in the index.
"BoldContent": Bold text.
"SubscriptContent": Subscripted text.
"SuperscriptContent": Superscripted text.
"MonospaceContent": Monospaced text.
"CrossrefContent": A cross reference to another part of the whole document.
"PassContent": Text that is copied in the output document without further processing.

"OtherContent" is used to extend the Content sum type with content that is specific to a backend.

4.1.2. Block

"Block" is the sum type of the possible blocks that compose an AsciiDoc document:

"ParagraphBlock": A paragraph.
"LiteralBlock": A pretty-printed block of text.
"ImageBlock": An image.
"SourceBlock": A pretty-printed block of source code.
"SidebarBlock": Visually separated content.
"QuoteBlock": A prose excerpt, quote, or verse.
"DefinitionListBlock": An association list.
"OrderedListBlock": An ordered list.
"UnorderedListBlock": An unordered list.
"TableBlock": An table.
"IndexBlock": A word that will be cross-referenced in the index.
"PassBlock": Text that is copied in the output document without further processing.

"OtherBlock" is used to extend the Block sum type with blocks that are specific to a backend.

4.1.3. Implementing Asciidoc

User-defined Idris2 types can implement the Asciidoc interface to streamline their conversion into AsciiDoc, in a way that is similar to the use of the standard Show and Pretty interfaces.

The Asciidoc interface defines two functions, contents and blocks which, after implementation, generates respectively a non-empty list of Content instances or a non-empty list of Block instances. The former is called when using an inline code macro, the latter when using a block code macro. Both functions are implemented by default so it is mandatory to implement only one of the two.

For example the following fragment defines how to render the result of a decision function in a specification:

```
> Asciidoc (Dec a) where
>   contents (Yes prf) = singleton (TextContent "yes")
>   contents (No contra) = singleton (TextContent "no")

Is 1 + 1 = 2? code:[decEq (1 + 1) 2]. +
Is 1 + 1 = 3? code:[decEq (1 + 1) 3].
```

The "Asciidoc" interface is implemented for the builtin types "String", "Char", "Integer", "Int", "Int8", "Int16", "Int32", "Int64", "Bits8", "Bits16", "Bits32", "Bits64", and "Double".

Additional implementations of the "Asciidoc" interface that are useful when writing an Internet-Draft will be documented in separate documents.

4.2. Xml2rfc Module

This module supplements the Asciidoc module with types that implement the AsciiDoc extensions specific to the "xml2rfc" Asciidoctor backend.

4.2.1. Content

"CrossrefXml2rfc" extends the "Crossref" content with additional attributes

"LinkXml2rfc" extends the "Link" content with additional attributes

"Bcp14" is an additional content.

"Comment" is an additional content.

"Unicode" is an additional content.

4.2.2. Block

"ParagraphXml2rfc" extends the "Paragraph" block with additional attributes

"LiteralXml2rfc" extends the "Literal" block with additional attributes

"ImageXml2rfc" extends the "Image" block with additional attributes

"SourceXml2rfc" extends the "Source" block with additional attributes

"Alt" is an additional block.

"Figure" is an additional block.

"DefinitionListXml2rfc" extends the "DefinitionList" block with additional attributes

"OrderedListXml2rfc" extends the "OrderedList" block with additional attributes

"UnorderedListXml2rfc" extends the "UnorderedList" block with additional attributes

4.2.3. Top-level Section

"TopSection" is the sum type of the possible top-level sections that compose an AsciiDoc document:

"Note" is an additional top-level section.

"AbstracTopSection": An abstract section.
"NormalTopSection": A section.
"BibliographyTopSection": A bibliography.
"AppendixTopSection": An appendix .
"IndexTopSection": An index.

4.2.4. Document

"Document" represents a complete AsciiDoc document.

5. Informative References

[AsciiDoc] "AsciiDoc", Accessed 23 April 2021, 8 March 2021, <https://en.wikipedia.org/wiki/AsciiDoc/>.

[Asciidoctor] "Asciidoctor Documentation Home :: Asciidoctor Docs", Accessed 23 April 2021, <https://docs.asciidoctor.org/home/>.

[draft-petithuguenin-xml2rfc-asciidoc] Petit-Huguenin, M., "Mappings Between XML2RFC v3 and AsciiDoc", Work in Progress, Internet-Draft, draft-petithuguenin-xml2rfc-asciidoc-01, 30 January 2023, <https://datatracker.ietf.org/doc/draft-petithuguenin-xml2rfc-asciidoc/01>.

[Idris2] "Documentation for the Idris 2 Language — Idris2 0.0 documentation", Accessed 31 January 2023, <https://idris2.readthedocs.io/en/latest/>.

[Knuth92] Knuth, D. E., "Literate Programming", 1 January 1992.

[LitProg] "Literate programming", Accessed 31 January 2023, 9 January 2023, <https://en.wikipedia.org/wiki/Literate_programming>.

[Metanorma] Inc, R., "Metanorma", Accessed 23 April 2021, <https://www.metanorma.com/>.

[RFC8489] Petit-Huguenin, M., Salgueiro, G., Rosenberg, J., Wing, D., Mahy, R., and P. Matthews, "Session Traversal Utilities for NAT (STUN)", RFC 8489, DOI 10.17487/RFC8489, February 2020, <https://www.rfc-editor.org/info/rfc8489>.

Appendix A. Installation

A computerate specification can be converted into an informal specification with the "computerate" command that is distributed as a Docker image that can be built as follow:

```
git clone --recursive git://shalmaneser.org/yathewEngod7 \
  computerate
cd computerate
docker build -t computerate .
```

An AsciiDoc file can then be converted with the following command:

```
docker run --rm -v $(pwd):/workspace computerate <file>
```

Note that only the files in the repository and sub-repositories where the command is executed are visible to that command. That means that files or symbolic links to files outside that hierarchy cannot be used. On the other hand external directories mounted with the "--bind" option can be used.

The "computerate" command is configured to include the following Asciidoctor add-ons:

*asciidoctor-xml2rfc [draft-petithuguenin-xml2rfc-asciidoc]
*asciidoctor-pdf
*asciidoctor-revealjs
*asciidoctor-epub3
*asciidoctor-idris2 [This document]
*asciidoctor-diagram

The following diagram generators are available for use by asciidoctor-diagram:

*actdiag
*blockdiag
*graphviz
*nwdiag
*packetdiag
*plantuml
*rackdiag
*seqdiag

The following additional Idris2 packages are installed in the Docker image:

*asciidoc [This document]

Appendix B. Package asciidoc

An Idris2 package to generate a document, an embeddable document, or an inline document in AsciiDoc.

Version: 0.0
Author(s): Marc Petit-Huguenin
License: AGPL-3.0-or-later
Dependencies: contrib

B.1. Module Asciidoc

A module that defines types for the generic AsciiDoc syntax.

data Align : Type
Alignment.

Left : Align
Left alignment.

Center : Align
Center alignment.

Right : Align
Right alignment.

interface Asciidoc a
Things that have an AsciiDoc representation.

Implemented by String, Char, Integer, Int, Int8, Int16, Int32, Int64, Bits8, Bits16, Bits32, Bits64, Double.

contents : a -> List1 Content
Converts a value into inline AsciiDoc.

blocks : a -> List1 Block
Converts a value into embedded AsciiDoc.

data Block : Type
Types of blocks.

ParagraphBlock : Paragraph -> Block
A block of text.

LiteralBlock : Literal -> Block
A pretty-printed block of text.

ImageBlock : Image -> Block
An image.

SourceBlock : Source -> Block
A pretty-printed block of source code.

SidebarBlock : Sidebar -> Block
Visually separated blocks.

QuoteBlock : Quote -> Block
A block of prose excerpt, quote or verse.

DefinitionListBlock : DefinitionList -> Block
An association list.

OrderedListBlock : OrderedList -> Block
An ordered list.

UnorderedListBlock : UnorderedList -> Block
An unordered list.

TableBlock : Table -> Block
A table.

IndexBlock : Index -> Block
An entry in the index.

PassBlock : Pass -> Block
Text that is passed directly to the backend.

NullBlock : Block

OtherBlock : Renderer a => a -> Block
Extended block.

data Content : Type
Types of inline content.

TextContent : String -> Content
Plain text content.

BreakContent : Content
An hard line break.

ItalicContent : List Content -> Content
Italicized content.

LinkContent : Link -> Content
An hyperlink.

IndexContent : Index -> Content
An entry in the index.

BoldContent : List Content -> Content
Bold content.

SubscriptContent : List Content -> Content
Subscript content.

SuperscriptContent : List Content -> Content
Superscript content.

MonospaceContent : List Content -> Content
Monospaced content.

CrossrefContent : Crossref -> Content
A cross-reference to another part of the document.

PassContent : String -> Content
Text that is passed directly to the backend.

OtherContent : Renderer a => a -> Content
Extended content.

record Crossref
A cross-reference content.

MkCrossref:
(target : String) -> (content : List Content) -> Crossref

target: An identifier for the target of the cross-reference.
content: The text for the cross-reference.

record DefinitionList
A definitition list.

MkDefinitionList:
(id : Maybe String) ->
(content : List1 (DefinitionTerm, Item)) -> DefinitionList

id: Identifier.
content: A non-empty list of definition term.

record DefinitionTerm
A definition term.

MkDefinitionTerm:
(id : Maybe String) -> (content : List Content) ->
DefinitionTerm

id: Identifier.
content: List of content.

record Image

An image.

MkImage:

```
(align : Maybe Align) -> (alt : Maybe String) ->
(id : Maybe String) -> (src : String) -> Image
```

align: Alignment.
alt: Alternate description.
id: Identifier.
src: SVG source.

record Index

An index term content

MkIndex:

```
(item : String) -> (primary : Maybe ()) ->
(subitem : Maybe String) -> Index
```

item: The primary term.
subitem: The secondary term.

record Item

A list or table cell.

MkItem:

```
(id : Maybe String) -> (ref : Maybe String) ->
(align : Maybe Align) -> (colspan : Maybe Nat) ->
(rowspan : Maybe Nat) ->
(content : Either (List1 Block) (List Content)) -> Item
```

id: Identifier.
align: Alignment.
content: Either a non-empty list of blocks or a list of content.

record Link

A link content.

MkLink:

```
(target : String) -> (content : String) -> Link
```

target: A URL.
content: The text for the link.

record Literal

Literal.

MkLiteral:

(id : Maybe String) -> (style : Maybe String) ->
(content : Doc ()) -> Literal

id: Identifier.
style: Style.
content: Pretty-printable content.

data Marker : Type
Unordered list label style.

Circle : Marker
Symbol.

NoBullet : Marker
No symbol, but indented.

Unstyled : Marker
No symbol, but not indented.

data NumberType : Type
Ordered list label type.

Lowercase : NumberType
Lower case alphabetic.

Uppercase : NumberType
Upper case alphabetic.

Decimal : NumberType
Decimal numbers.

LowercaseRoman : NumberType
Lower case roman numeral.

UppercaseRoman : NumberType
Upper case roman numeral.

record OrderedList
An ordered list.

MkOrderedList:
(id : Maybe String) -> (group : Maybe String) ->
(start : Maybe Nat) -> (type : Maybe NumberType) ->
(content : List1 Item) -> OrderedList

id: Identifier.
group: Numbering group.
start: Numbering start.
type: Labels type.
content: A non-empty list of items.

record Paragraph
A paragraph.

MkParagraph:
```
(id : Maybe String) -> (content : List Content) -> Paragraph
```

id: Identifier.
content: The list of content.

record Pass
Passthrough.

MkPass:
```
(id : Maybe String) -> (content : String) -> Pass
```

id: Identifier.
content: Text passed through.

record Quote
A quote.

MkQuote:
```
(id : Maybe String) -> (cite : Maybe String) ->
(quotedFrom : Maybe String) ->
(content : Either (List1 Block) (List1 Content)) -> Quote
```

id: Identifier.
cite: Source of the citation.
quotedFrom: Origin of the quote.
content: Either a non-empty list of blocks or a non-empty list of content.

record Row
A table row.

MkRow:
```
(id : Maybe String) -> (content : List1 (Either Item Item)) ->
Row
```

id: Identifier.
content: A non-empty list of items.

record Sidebar
A sidebar,

MkSidebar:
```
(id : Maybe String) -> (content : List Block) -> Sidebar
```

id: Identifier.
content: List of blocks.

record Source

MkSource:

```
(id : Maybe String) -> (type : Maybe String) ->
(content : Doc ()) -> Source
```

record Table

A table.

MkTable:

```
(id : Maybe String) -> (title : Maybe (List Content)) ->
(align : Maybe Align) -> (head : Maybe (List1 Row)) ->
(body : List1 Row) -> (foot : Maybe (List1 Row)) -> Table
```

id: Identifier.
title: Title.
align: Alignment.
head: An non-empty list of rows for the header.
body: An non-empty list of rows.
foot: An non-empty list of rows for the footer.

record UnorderedList

An unordered list.

MkUnorderedList:

```
(id : Maybe String) -> (title : Maybe (List Content)) ->
(marker : Marker) -> (content : List1 Item) -> UnorderedList
```

id: Identifier.
title: Title.
marker: Type of symbol.
content: A non-empty list of items.

renderBlocks : List1 Block -> String

Converts a non-empty list of Block into an embeddable AsciiDoc fragment.

renderContents : List1 Content -> String

Converts a non-empty list of Content into an inline AsciiDoc fragment.

B.2. Module Asciidoc.Xml2rfc

A Module that defines types for the AsciiDoc extensions of the "xml2rfc" back-end.

record Abstract

MkAbstract:

```
(id : Maybe String) -> (title : Maybe (List Content)) ->
(blocks : List Block) -> Abstract
```

```
record Alt
   MkAlt:
      (id : Maybe String) -> (content : List1 Block) -> Alt

record Appendix
   MkAppendix:
      (id : Maybe String) -> (title : Maybe (List Content)) ->
      (notNumbered : Maybe ()) -> (removeInRfc : Maybe ()) ->
      (toc : Maybe Bool) -> (blocks : List Block) ->
      (sections : List Section) -> Appendix

record Bcp14
   MkBcp14:
      (content : String) -> Bcp14

record Bibliography
   MkBibliography:
      (id : Maybe String) -> (title : Maybe (List Content)) ->
      (blocks : List Block) -> Bibliography

data Category : Type
   Intended category.

   StdCategory : Category
      Standard category.

   BcpCategory : Category
      BCP category.

   ExpCategory : Category
      Experiemntal category.

   InfoCategory : Category
      Informational category.

   HistoricCategory : Category
      Historic category.

record Comment
   MkComment:
      (id : Maybe String) -> (noDisplay : Maybe ()) ->
      (source : Maybe String) -> (content : List Content) -> Comment

record CrossrefXml2rfc
   MkCrossrefXml2rfc:
      (target : String) -> (format : Maybe Format) ->
      (relative : Maybe String) -> (section : Maybe String) ->
      (sectionFormat : Maybe SectionFormat) ->
      (content : List Content) -> CrossrefXml2rfc
```

record DefinitionListXml2rfc

MkDefinitionListXml2rfc:

```
(id : Maybe String) -> (indent : Maybe Nat) ->
(newline : Maybe ()) -> (compactSpacing : Maybe ()) ->
(content : List1 (DefinitionTerm, Item)) ->
DefinitionListXml2rfc
```

record Document

MkDocument:

```
(title : String) -> (abbrev : Maybe String) ->
(category : Category) -> (consensus : Maybe ()) ->
(docName : Maybe String) -> (ipr : Maybe Ipr) ->
(obsoletes : Maybe String) -> (sortRefs : Maybe ()) ->
(submissionType : Maybe Submission) ->
(noSymRefs : Maybe ()) -> (tocDepth : Maybe Nat) ->
(noTocInclude : Maybe ()) -> (updates : Maybe String) ->
(sections : List TopSection) -> Document
```

record Figure

MkFigure:

```
(id : Maybe String) -> (name : List Content) ->
(content : List1 Block) -> Figure
```

data Format : Type

Crossref format.

Implements Show.

TitleFormat : Format

Title format.

CounterFormat : Format

Counter format.

NoneFormat : Format

No format.

record ImageXml2rfc

MkImageXml2rfc:

```
(align : Maybe Align) -> (alt : Maybe String) ->
(id : Maybe String) -> (src : String) ->
(type : Maybe String) -> (top : Maybe ()) ->
(bottom : Maybe ()) -> (content : String) -> ImageXml2rfc
```

data Ipr : Type

Intellectual Property Rights.

Trust200902Ipr : Ipr

NoModificationTrust200902Ipr : Ipr

```
   NoDerivativesTrust200902Ipr : Ipr

   Pre5378Trust200902Ipr : Ipr

   Trust200811Ipr : Ipr

   NoModificationTrust200811Ipr : Ipr

   NoDerivativesTrust200811Ipr : Ipr

   Full3978Ipr : Ipr

   NoModification3978Ipr : Ipr

   NoDerivatives3978Ipr : Ipr

   Full3667Ipr : Ipr

   NoModification3667Ipr : Ipr

   NoDerivatives3667Ipr : Ipr

   Full2026Ipr : Ipr

   NoDerivativeWorks2026Ipr : Ipr

   None : Ipr

record LinkXml2rfc
   MkLinkXml2rfc:
      (angleBrackets : Maybe ()) -> (target : String) ->
      (content : String) -> LinkXml2rfc

record LiteralXml2rfc
   MkLiteralXml2rfc:
      (align : Maybe Align) -> (style : Maybe String) ->
      (alt : Maybe String) -> (id : Maybe String) ->
      (name : Maybe String) -> (type : Maybe String) ->
      (top : Maybe String) -> (bottom : Maybe String) ->
      (content : Doc ()) -> LiteralXml2rfc

record Note
   MkNote:
      (id : Maybe String) -> (title : Maybe (List Content)) ->
      (blocks : List Block) -> Note

record OrderedListXml2rfc
   MkOrderedListXml2rfc:
      (id : Maybe String) -> (group : Maybe String) ->
      (indent : Maybe Nat) -> (compactSpacing : Maybe ()) ->
      (start : Maybe Nat) ->
      (type : Maybe (NumberType, Maybe (String, String))) ->
      (content : List1 Item) -> OrderedListXml2rfc
```

record ParagraphXml2rfc
   MkParagraphXml2rfc:

```
      (id : Maybe String) -> (indent : Maybe Nat) ->
      (keepWithNext : Maybe ()) -> (keepWithPrevious : Maybe ()) ->
      (content : List Content) -> ParagraphXml2rfc
```

record Section
   MkSection:

```
      (id : Maybe String) -> (title : Maybe (List Content)) ->
      (notNumbered : Maybe ()) -> (removeInRfc : Maybe ()) ->
      (toc : Maybe Bool) -> (blocks : List Block) ->
      (sections : List Section) -> Section
```

data SectionFormat : Type
   External reference format.

   Implements Show.

   CommaSectionFormat : SectionFormat
      Comma as separator.

   ParensSectionFormat : SectionFormat
      Parentheses as separator.

   BareSectionFormat : SectionFormat
      Same link withing parentheses.

record SourceXml2rfc
   MkSourceXml2rfc:

```
      (id : Maybe String) -> (markers : Maybe ()) ->
      (name : Maybe String) -> (type : Maybe String) ->
      (top : Maybe String) -> (bottom : Maybe String) ->
      (content : Doc ()) -> SourceXml2rfc
```

data Submission : Type
   Intended stream.

   IetfSubmission : Submission
      IETF stream.

   IabSubmission : Submission
      IAB stream.

   IrtfSubmission : Submission
      IRTF stream.

IndependentSubmission : Submission
Independent stream.

EditorialSubmission : Submission
Editorial stream.

data TopSection : Type
Types of top sections.

AbstractTopSection : Abstract -> TopSection
The abstract.

NormalTopSection : Section -> TopSection
A section.

BibliographyTopSection : Bibliography -> TopSection
A bibliography.

AppendixTopSection : Appendix -> TopSection
An appendix.

IndexTopSection : TopSection
The index.

OtherTopSection : Renderer a => a -> TopSection
Extended top-level section.

record Unicode
MkUnicode:
(id : Maybe String) -> (ascii : Maybe String) ->
(format : Maybe String) -> (content : String) -> Unicode

record UnorderedListXml2rfc
MkUnorderedListXml2rfc:
(id : Maybe String) -> (title : Maybe (List Content)) ->
(marker : Marker) -> (indent : Maybe Nat) ->
(compactSpacing : Maybe ()) -> (content : List1 Item) ->
UnorderedListXml2rfc

renderDocument : (d : Document) -> String
Converts a Document into an Asciidoc document.

d: the document.

renderSection : Nat -> Section -> String

Contributors

Stéphane Bryant

Email: stephane.ml.bryant@gmail.com

Stephane is a co-founder of the Nephelion project, project that started back in 2014 during a week-end visiting national parks in Utah. Computerate Specifying is the successor of this project, and it could not have been done without the frequent reviews and video calls with Stephane during these last 9 years.

Author's Address

Marc Petit-Huguenin
Impedance Mismatch LLC

Email: marc@petit-huguenin.org