RAML to HTML, through Java Parser

Rahul S. Agasthya

Department of Computer Science

SUNY Stony Brook University

Stony Brook, NY-11794

Author Note

This project is for the documentation of Infinera Products, for an efficient translation of data from one language to another.

Infinera India Pvt. Ltd.

Prestige Solitaire, 401, Level 4

6, Brunton Road

Bangalore KA - 560025

Abstract

The Documentation work at Infinera India Pvt. Ltd. included manual translation of data in RAML files to PDF files. To ease this process, the RAML to HTML Parser was developed that produces a HTML page of the Schemas in the RAML file. With minimal modification, the final piece can be pasted in the end-user document.

Keywords: RAML, HTML, Schema, Parser.

Contents

RAML to HTML, through Java Parser

This RAML to HTML Java Parser is an effective tool to document the schemas of a particular RAML file.

What is RAML?

RESTful API Modeling Language (RAML) is a YAML-based language for describing RESTful APIs. It provides all the information necessary to describe RESTful or practically-RESTful APIs. Although designed with RESTful APIs in mind, RAML is capable of describing APIs that do not obey all constraints of REST (hence the description "practically-RESTful"). It encourages reuse, enables discovery and pattern-sharing, and aims for merit-based emergence of best practices. (Galiegue, 2014)

What is HTML?

Hyper Text Markup Language, commonly referred to as HTML, is the standard markup language used to create web pages. (Merriam-Webster, 1989)

Web browsers can read HTML files and render them into visible or audible web pages. Browsers do not display the HTML tags and scripts, but use them to interpret the content of the page. HTML describes the structure of a website semantically along with cues for presentation, making it a markup language, rather than a programming language.

HTML elements form the building blocks of all websites. HTML allows images and objects to be embedded and can be used to create interactive forms. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. It can embed scripts written in languages such as JavaScript which affect the behavior of HTML web pages.

Web browsers can also refer to Cascading Style Sheets (CSS) to define the look and layout of text and other material. The World Wide Web Consortium (W3C), maintainer of both the HTML and the CSS standards, has encouraged the use of CSS over explicit presentational HTML since 1997. (World Wide Web Consortium, 1997)

What Editor is required to run this parser?

Idea IntelliJ Community Edition is the recommended editor for this job.

Installation link: `https://www.jetbrains.com/idea/download/`

This also requires the Java Development Kit, Version 7 or higher.

Installation link:

`http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html`

IntelliJ IDEA is a Java integrated development environment (IDE) for developing computer software. It is developed by JetBrains (formerly known as IntelliJ), and is available as an Apache 2 Licensed community edition, and in a proprietary commercial edition. IntelliJ IDEA is not based on Eclipse like MyEclipse or Oracle Enterprise Pack for Eclipse. (Kudtyashov, 2015)

System Requirements for Idea IntelliJ (JetBrains, 2015):

- Windows 8/7/Vista/2003/XP or MacOS X: 10.5 – 10.9 or Linux: GNOME or KDE desktop.
- 1 GB RAM minimum, recommended 2 GB.
- 300 MB hard disk space + minimum 1 GB cache memory.
- 1024×768 minimum screen resolution.
- JDK 1.6 or higher. 7 or higher for JavaFX

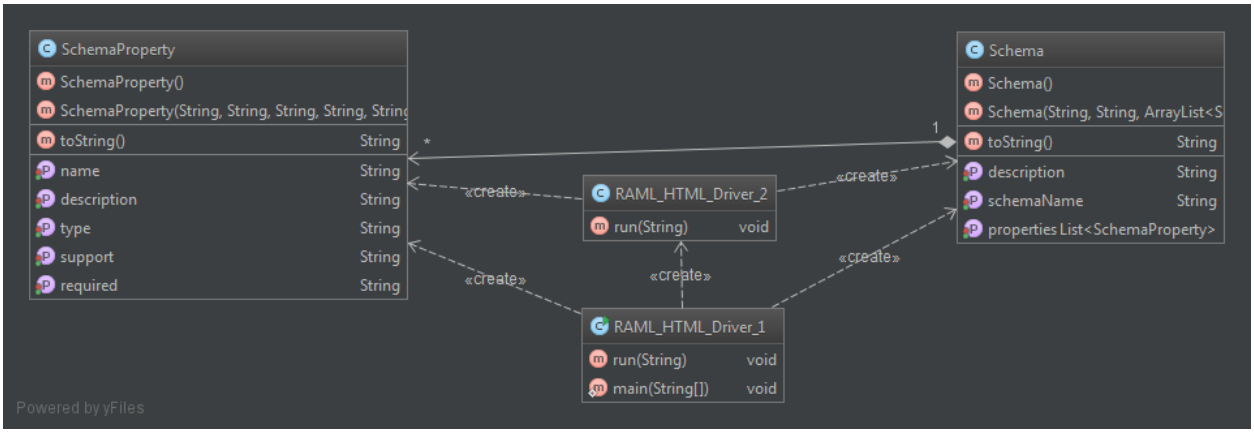
Project Overview

Project Content

The main contents of the project include:

1. Java Classes:
    a. SchemaProperty
    b. Schema
    c. RAML_HTML_Driver_1
    d. RAML_HTML_Driver_2
2. JAR Files:
    a. raml-java-parser-master
3. User Input File:
    a. One RAML file in the folder src. (Name is user discretion)
4. Output file:
    a. One HTML file in project main folder. (Name variable)

Project Class Diagrams:



Main Classes. The three main classes that are part of this project:

RAML_HTML_Driver_2. The `RAML_HTML_Driver_2` class, implements ArrayLists of Schema objects, which is used to generate a HTML file of the Schemas. This contains the `run()` method.

The `run()` method is the core method for the generation of the HTML file. It uses the raml-java-parser that is used to read RAML files. Parameter passed is *`@param`* `filename`. Exception thrown *`@throws`* `IOException`.

Using the raml-java-parser, a RAML file readable by Java is built. An object array called schemas extract all the schemas to an array. The variable title extracts the heading of the RAML file. Using the `FileWriter` object and the `BufferedWriter`, a new HTML file is created with the headers and titles specified. The title of HTML file will be the heading of the RAML file. Every Schema in the array is stored as a `String s`. This will be split on basis of new line (`\n`) to an `ArrayList array`. The variable `numberOfProperties` is used to find the total number of properties present in the Schema. An iterator will check for `"},"` and increment the value of `numberOfProperties` if found. Using `String.split()`, `String.substring()`, `String.trim()` and `String.replaceAll()`, the name, description, type, required condition and the supporting values are extracted. Similarly, the name and description of the schema are extracted. All schemas were added to the ArrayList. Simultaneously, the writer calls the write function and sends it to separate HTML files, whose name is the name of the Schema.

RAML_HTML_Driver_1. The `RAML_HTML_Driver_1` class, implements ArrayLists of Schema objects, which is used to generate a HTML file of the Schemas. This class contains two methods: `run()` and `main()`.

The `run()` method is the core method for the generation of the HTML file. It uses the raml-java-parser that is used to read RAML files. Parameter passed is *@param* `filename`. Exception thrown *@throws* `IOException`.

Using the raml-java-parser, a RAML file readable by Java is built. An object array called schemas extract all the schemas to an array. The variable title extracts the heading of the RAML file. Using the `FileWriter` object and the `BufferedWriter`, a new HTML file is created with the headers and titles specified. The title of HTML file will be the heading of the RAML file. Every Schema in the array is stored as a `String s`. This will be split on basis of new line (`\n`) to an `ArrayList array`. The variable `numberOfProperties` is used to find the total number of properties present in the Schema. An iterator will check for `"},"` and increment the value of `numberOfProperties` if found. Using `String.split()`, `String.substring()`, `String.trim()` and `String.replaceAll()`, the name, description, type, required condition and the supporting values are extracted. Similarly, the name and description of the schema are extracted. All schemas were added to the ArrayList. Simultaneously, the writer calls the write function and sends it to the HTML file. This method also creates an HTML File which is a contents page for the list of Schemas, whose HTML files is generated in `RAML_HTML_Driver_1.`

The `main()` method is the main function of the project. The name of the RAML file is accepted from the user and checks whether the file exist or not. It then creates an object of `RAML_HTML_Driver_1` and `RAML_HTML_Driver_2`, called `driver1` and `driver2`. The driver object then calls the `run()` function of the objects and passes the file name.

SchemaProperty. The `SchemaProperty` class is used to create an object for `Schema` class.

INVARIANTS:

`name` stores name of the property.

`type` stores the type of variable, like int, float, String, array, enum, etc.

`description` stores the description of the Schema Property.

`required` stores a boolean value as String, if required true, else false.

`support` stores the Supporting Variables mainly required for Arrays and Enums.

Schema: The `Schema` class, implements ArrayLists of `SchemaProperty`, which is used to create an object for `RAML_HTML_Driver` class.

INVARIANTS:

`schemaName` stores the name of the schema.

`description` stores the Schema Description.

`properties` is an ArrayList of type `SchemaProperty` and is used to store the list of properties that follow.

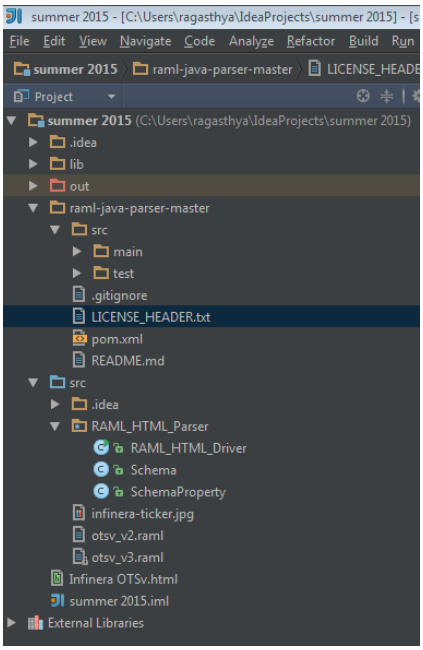
JAR Files. To convert the RAML files to Java readable, a JAR File was used.

raml-java-parser-master. The JAR contains a RAML java parser compatible with version 0.8 of the RAMLspecification. The parser depends on SnakeYaml, a Java YAML parser.

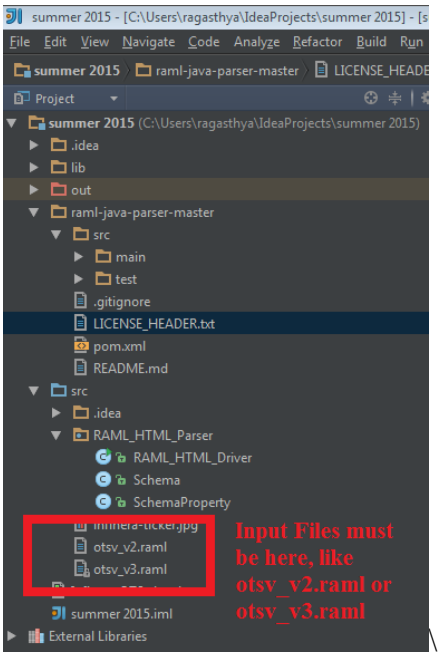
Using the RAML-HTML Parser

Check whether all input files are in the right place:

The main directory will look something like this:



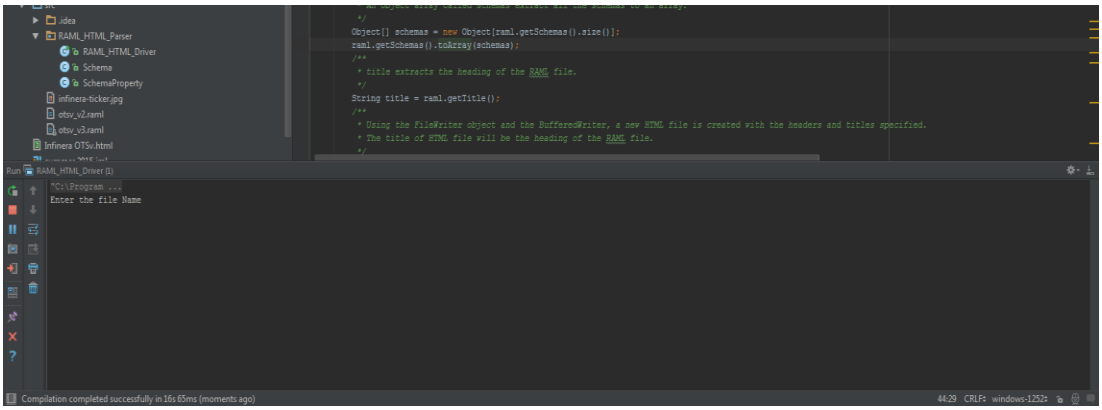
The input files should go in `RAML_HTML_Parser` folder.



Using the Parser

Open `RAML_HTML_Driver_1` and press `Shift + F10`.

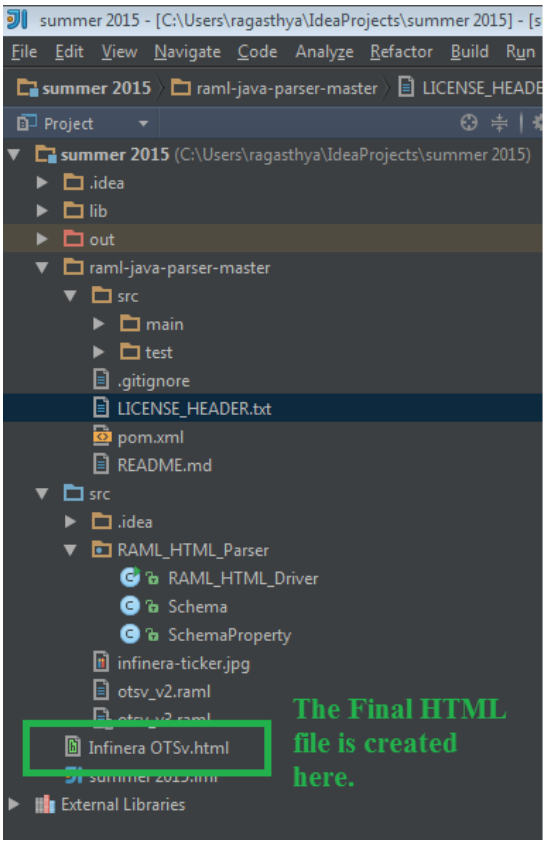
A similar screen will appear:



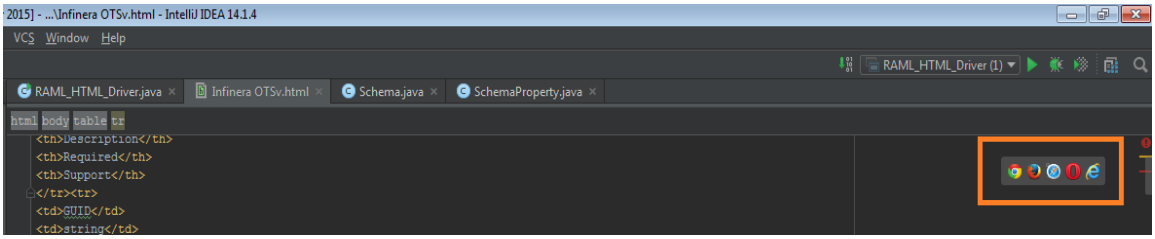
On the console, enter the same input file name exactly, with the extension of `.raml`.

Over here, it would be `otsv_v2.raml` or `otsv_v3.raml`.

The refreshed directory will look like:



Open the `.HTML` file and move the mouse to the upper right corner, a dialog box will pop up and choose your preferred browser.



Output

Infinera OTSv

ncdomain

The Network Control Domain (NCD) represents the scope of control for network managed by OTSv server. NCD instance is automatically instantiated per OTSv server. Network Name attribute is the naming attribute for NCD instane. Only one instance of NCD is supported in this release. No updates are supported on NCD

Schema for ncdomain

Name	Type	Description
GUID	string	Global Unique ID for Network Controller Domain Valid Values Are: • OTSv
URI	string	URI for the resource
operationalState	string	Operational state of OTSv, up value represents a fully functional network controller domain, readonly attribute Valid Values Are: • up • down
networkName	string	Network Name as defined in the OTSv configration file
apiVersions	array	Versions of the API supported by this instance.
softwareVersion	string	Version of OTSv software currently running
buildVersion	string	Build of OTSv software currently running

To Main Page

The output returns a list of schemas.

Customizing your RAML-HTML Parser

Customizing the Schema:

Editing the Schema Class.

The editor must add another variable in the Schema Class.

It should be prior to `private ArrayList<SchemaProperty> properties.`

Modify the Getter and Steer Methods, and the `toString()` method.

Ideally, for any data type other than Boolean, the getter must be:

```
public <datatype> get<VariableName>() {
      return variableName;
}
```

For a Boolean variable, the getter will be:

```
public boolean is<VariableName>() {
      return variableName;
}
```

And the setter will follow as:

```
public void set<VariableName>(<datatype> <variableName>) {
      this.variableName = variableName;
}
```

Customizing the Schema Property Class:

This procedure will be similar to customizing the Schema class.

Customizing the Driver Classes:

Check for the relevant variable in the RAML file, and extract them using the String functions. Refer Appendix 1 for further details on the use of String functions. The changes must be replicated in either of the classes' `run()` methods.

NOTE:

The position of the HTML headers and footers varies in the two driver classes. Please ensure their position remains either within the first infinite for loop (in Driver 2) or outside (in Driver 1).

References

Fodor, P. (2014, August). *Computer Science I - Procedural and object-oriented programming*. Retrieved from CSE 114: http://www3.cs.stonybrook.edu/~pfodor/courses/summer/cse114.html

Galiegue, F. Z. (2014, July 15). *RAML™ Version 0.8: RESTful API Modeling* . Retrieved from RAML 0.8: http://raml.org/spec.html

JetBrains. (2015, June 19). *IntelliJ IDEA :: Downloads:: System Requirements*. Retrieved from Download IntelliJ IDEA 14.1: https://www.jetbrains.com/idea/download/

Kudtyashov, V. (2015, July 9). *JetBrains/IntelliJ-community*. Retrieved from GitHub: https://github.com/JetBrains/intellij-community

Last Name, F. M. (Year). Article Title. *Journal Title*, Pages From - To.

Last Name, F. M. (Year). *Book Title.* City Name: Publisher Name.

Merriam-Webster. (1989). *Merriam-Webster's Collegiate Dictionary.* Encyclopedia Britanica Company.

w3 Schools. (n.d.). *HTML Tutorial*. Retrieved from w3schools.com: w3schools.com

World Wide Web Consortium. (1997, December 18). *Conformance: requirements and recommendations1*. Retrieved from HTML user agent: http://www.w3.org/TR/REC-html40-971218/conform.html#deprecated

Appendix 1

String Functions in Java: (Fodor, 2014)

Package: `java.lang.String`

Function	Description
`+toLowerCase(): String`	Returns a new string with all characters converted to lowercase.
`+toUpperCase(): String`	Returns a new string with all characters converted to uppercase.
`+trim(): String`	Returns a new string with blank characters trimmed on both sides.
`+replace(oldChar: char, newChar: char): String`	Returns a new string that replaces all matching character in this string with the new character
`+replaceFirst(oldString: String, newString: String): String`	Returns a new string that replaces the first matching substring in this string with the new substring.
`+replaceAll(oldString: String, newString: String): String`	Returns a new string that replace all matching substrings in this string with the new substring
`+split(delimiter: String): String[]`	Returns an array of strings consisting of the substrings split by the delimiter.

Examples:

`"Welcome".toLowerCase()` returns a new string, "welcome".

`"Welcome".toUpperCase()` returns a new string, "WELCOME".

`" Welcome ".trim()` returns a new string, "Welcome".

`"Welcome".replace('e', 'A')` returns a new string, "WAlcomA".

`"Welcome".replaceFirst("e", "AB")` returns a new string, "WABlcome".

`"Welcome".replaceAll("e", "AB")` returns a new string, "WABlcomAB".

`"Welcome".replaceAll("el", "AB")` returns a new string, "WABcome".

```
String[] tokens = "Java#HTML#Perl".split("#");
```

The array will comprise `[“Java”, “HTML”, “Perl”]`

ArrayList Objects:

Package `java.util.ArrayList;`

Function	Description
`+ArrayList()`	Creates an empty list.
`+add(o: Object): void`	Appends a new element o at the end of this list.
`+add(index: int, o: Object): void`	Adds a new element o at the specified index in this list.
`+clear(): void`	Removes all the elements from this list.
`+contains(o: Object): boolean`	Returns true if this list contains the element o.
`+get(index: int): Object`	Returns the element from this list at the specified index.
`+indexOf(o: Object): int`	Returns the index of the first matching element in this list
`+isEmpty(): boolean`	Returns true if this list contains no elements.
`+lastIndexOf(o: Object): int`	Returns the index of the last matching element in this list.
`+remove(o: Object): int`	Removes the element o from this list.
`+size(): int`	Returns the number of elements in this list.
`+remove(index: int): Object`	Removes the element at the specified index.
`+set(index: int, o:Object): Object`	Sets the element at the specified index.

Refer `http://www3.cs.stonybrook.edu/~pfodor/courses/cse114.html`

Appendix 2

HTML Scripting tools (w3 Schools, n.d.)

Introduction:

The DOCTYPE declaration defines the document type to be HTML. The text between `<html>` and `</html>` describes an HTML document. The text between `<head>` and `</head>` provides information about the document. The text between `<title>` and `</title>` provides a title for the document. The text between `<body>` and `</body>` describes the visible page content. The text between `<h1>` and `</h1>` describes a heading. The text between `<p>` and `</p>` describes a paragraph. Using this description, a web browser can display a document with a heading and a paragraph.

HTML Tags:

HTML tags are keywords (tag names) surrounded by angle brackets:

```
<tagname>content</tagname>
```

HTML tags normally come in pairs like `<p>` and `</p>.`

The first tag in a pair is the start tag, the second tag is the end tag.

The end tag is written like the start tag, but with a slash before the tag name.

Styling HTML with CSS

CSS stands for Cascading Style Sheets. Styling can be added to HTML elements in 3 ways:

- Inline - using a style attribute in HTML elements.
- Internal - using a `<style>` element in the HTML `<head>` section.
- External - using one or more external CSS files.

Tables in HTML:

Tables are defined with the <table> tag. Tables are divided into table rows with the `<tr>` tag. Table rows are divided into table data with the `<td>` tag. A table row can also be divided into table headings with the `<th>` tag.

Refer `http://www.w3schools.com/html/`