Final Project (part of)
(bk112)
2023-09-15

Contents

Introduction

This project is a JavaScript based tool, that would be a part of a set of tools designed to create easier interoperability amongst web-based systems and APIs. The tool proposed would be able to convert a common web based data structure, being an XML document of a particular schema, into other XML document conforming to a different schema. In summary mapping between two XML schema's.

A tool like this would be able to give application's and API's the ability to declare their own XML schema and handle all communications with their declaration, opposed to the norm of systems needing to transform the data to and from each source individually.

Motivation

According to the literature, where there are highlights of the past and current issues with web-based applications. As of now there isn't a standardises approach nor a seamless solution to the following issues:

- Interoperability amongst applications and their data structures and schema's. [1]
- Re-usability of data-types and schema's: In terms of standardising data structures as well as re-usability of knowledge and the data itself. [2][3][4]
- Citable applications, such as knowledge representations and machine understandable data representations. [1-1]

Within the literature there is evidence of researchers trying to standardise and integrate knowledge representations of their data, industry by industry. Though a clear struggle these domains are having is the attempt to standardise data-structures and schema's, where there is evidence this solution isn't ideal for interoperability. The literature does show that knowledge representations are a good solution though the lack of ordered structure from them make them less useful for majority of web services where structure is needed [2-1]. So i believe work on understanding current data representations and schema's, in particular, transforming data in a way that client systems can understand it, is a better method opposed to trying to standardise domains or have their data machine understandable directly.

In the past 20 years it has been clear that most applications have been moving towards the web, being web-based and functioning around APIs and external sources of data, seem to be the future. With this in mind i would like to see; applications not needing to be concerned with the differences in data-structures and schema's when using new or multiple APIs.

Primary-Issues

The issues i would like solutions to are as follows:

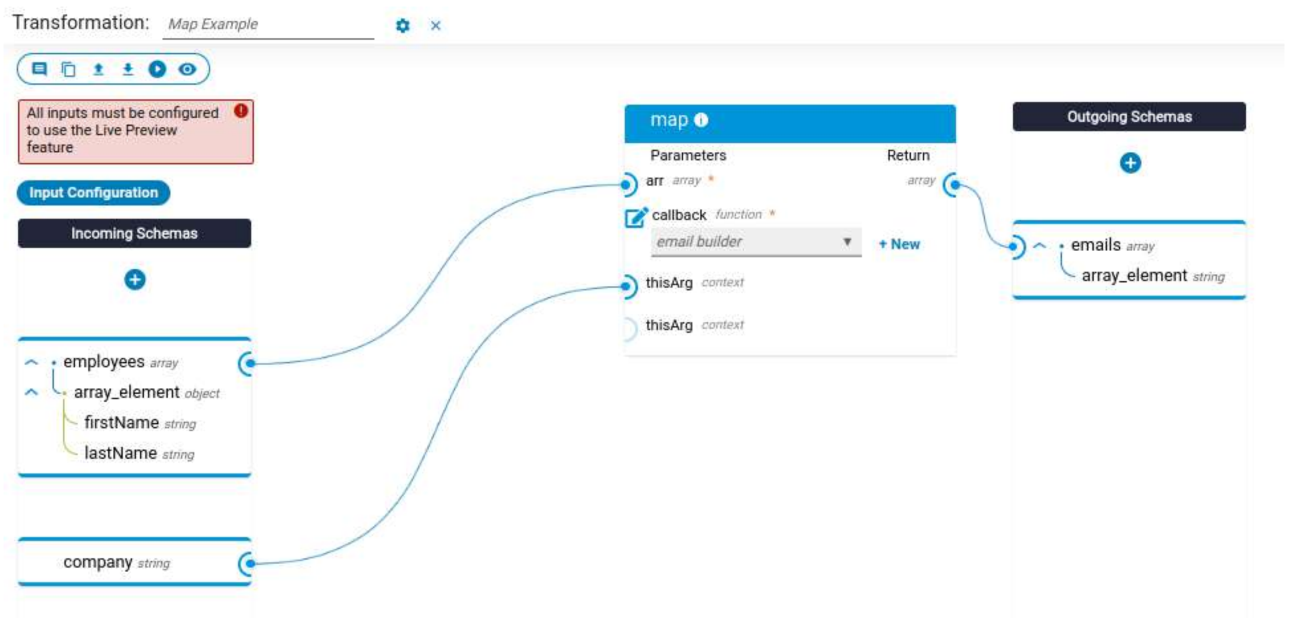
- Currently we have technologies for interoperability such as XSLT, though creating XSLT conversion documents require a deep understanding of both the source file and target file types and in every situation requires a knowledgeable developer of XSLT to design and implement these.

- Client applications needing to have a full understanding of both their own data-structures as well as external data-structures such as the APIs they utilise. I would like each application only needing to worry about their own data-structures and schema's [1-2].
- The difficulty of client application utilising multiple APIs or changing APIs, due to each source having its own data-structure and schema.

Solution-Envisioned

The method i envision that can be a possible solution to these issues are as follows:

- Server-sided tooling that is able to convert source data from one data-structure and schema to another data-structure and schema. Primarily a tool that takes as input a source XSD schema, a target XSD schema and a "mapping object" between the two, then returns an XSLT that can be used on the source data to transform it to the desired target data-structure and schema.
- Client side tooling, such as a browser UI that is able to allow developers to easily create "mapping objects" from source data-structures and schema's to their own data-structure and schema without the need to fully understand the sources data, though just understand the vocabulary used in the data. The UI would be of a similar design to *image_2*, which shows a source data-structure and a target data-structure with nodes and mapping aggregation functions connecting the two.


Transformation: Map Example

All inputs must be configured to use the Live Preview feature

image_2: Illustration of a node based diagram approach to mapping schema's

Domain-Users

The domain and users i will be catering to, would be content providers, such as APIs publishing any type of data, or any other types of web-based service that makes data available to multiple external client applications.

- Giving these content providers the ability to have a much wider range of users, by allowing an easier way that client applications can integrate with their services.
- This will be accomplished by transforming data to the clients desired data-structure and schema as well as allow data from client applications to be received in the clients data-structure and schema.

A secondary domain i will be catering to would be client applications that might be utilising multiple APIs or data providers, as well as adding and changing these sources frequently.

- Giving a gateway tool that is able to retrieve multiple different data-structures and schema's and transform such data into the required data-structure and schema.

Scope

In the early stages of the project, i had set an ambitious scope, where i was wanting to implement conversions between JSON, YAML and XML datatypes as well their schema's, though due to the hiccups along the way and time, i had opted to focus primarily on the XML datatype and the transformation of their schema's.

I will be focusing on the server-side tooling that is able to utilise a mapping file and XML schema's to transform data. Though i will not be doing the UI tooling for creating such mappings. I will be focusing on the following:

- A tool that can achieve conversions on vocabulary, data structures and can apply aggregate functions during conversions.
- A tool that includes some error handling.
- A tool that can return some insights into the results.

Although both the server-side tooling and the browser mapping UI would be needed for the full and seamless implementation of this solution, i will be focusing on the main function of the project which will happen within the server-side library. And leave the UI for a future project.

Design

The design envisioned for the workings of this project.

Feature-Goals

For a complete set of tools to tackle the "Solution-Envisioned" in this project. I would require two developed components, being a browser based mapping UI as well as a server-side transformation library.

Browser based mapping UI

An independent library in JavaScript that supplies an interactive tool that it used to create the mapping component of the system, taking the form illustrated in *image_1*. This tool ideally would return a "mapping object" that would then be used for creation of the XSLT style-sheet for the transformations.

This UI component is out of the scope of the project and I will not be developing this, though in the implementation I will go through the design of the "mapping object".

Server-side Library

A JavaScript library that has the following primary features:

- Creation of an XSLT style-sheet document based on an XSD and a "mapping object".
- Methods of validating XML, XSD and XSLT files.
- Methods of converting data based on one data-structure and schema, and returning the data according to a clients required data-structure and schema.
- A report indicating how much data has been lost or not being sourced within the XSD.

Both the deployed APIs and the application developers could integrate this as a middle-ware on their servers, giving the ability to send and receive XML documents conforming to custom XSD schema's, apposed to having to create XSLT transformations for each integration.

A library in the JavaScript language, in particular for a runtime environment such as NodeJS. I feel this framework is a good starting point as this tool would be highly beneficial if client based processing could eventually also be done within the client browsers, which then would also be a JavaScript environment.

Hypothesis

Currently the way developers integrate APIs in their application involve the developer having to read and understand the APIs documentation, then including the desired API endpoints within their application. The application then needs to 'understand' the data being received from these APIs as well as make sure these APIs understand the data the application sends to them. This gets done by manually transforming each data pipeline for each API [2-2], or designing the application around the data required to send or receive[3-1].

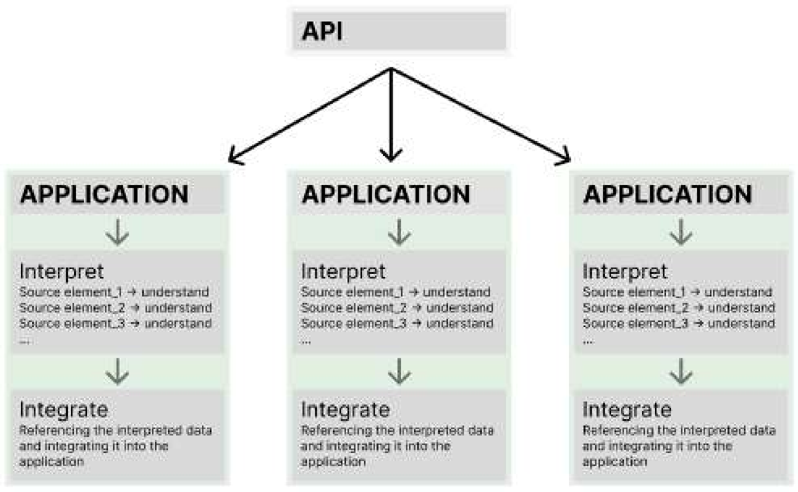

API
APPLICATION
Interpret
Source element_1 → understand
Source element_2 → understand
Source element_3 → understand
...
Integrate
Referencing the interpreted data and integrating it into the application
APPLICATION
Interpret
Source element_1 → understand
Source element_2 → understand
Source element_3 → understand
...
Integrate
Referencing the interpreted data and integrating it into the application
APPLICATION
Interpret
Source element_1 → understand
Source element_2 → understand
Source element_3 → understand
...
Integrate
Referencing the interpreted data and integrating it into the application

Image_3: Current process of API integrations

Within both hypothesis below the developer would create a "mapping object" using a graphical UI, which would describe the transformation required of the schema, the schema could either be of the data sent or received to and from the API. Then the actual transformation could either be done on the developer or the API's server.

Hypothesis 1
The "mapping object" created by the developer would be stored by the API and the transformation of data would happen on the API's server for data received from the client as well as the data sent

to the client. The API would know which mapping to use as the client would just need to include a mapping code in their request.

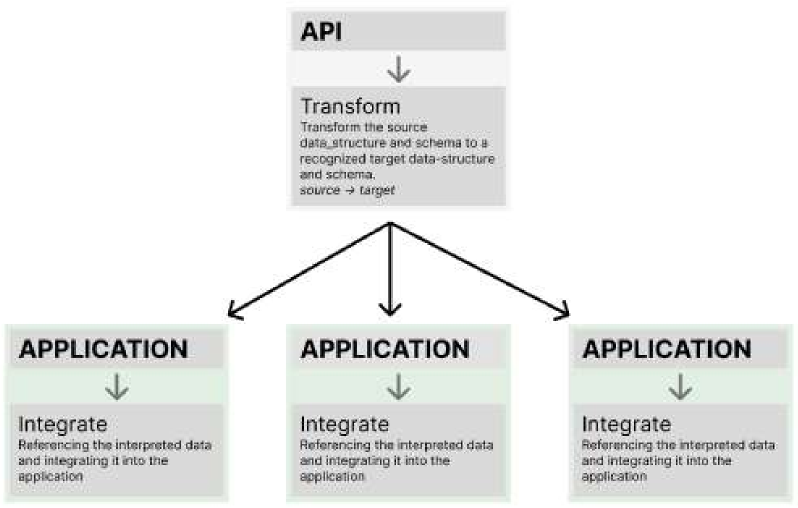

API
Transform
Transform the source data_structure and schema to a recognized target data-structure and schema.
source → target
APPLICATION
Integrate
Referencing the interpreted data and integrating it into the application
APPLICATION
Integrate
Referencing the interpreted data and integrating it into the application
APPLICATION
Integrate
Referencing the interpreted data and integrating it into the application

image_4: Process of hypothesis with the API applying the transformation

Hypothesis 2
The "mapping object" created by the developer would be stored by the developer and the transformation of data would happen on the developer's server for data sent and received to the API. The tooling will act as a gateway between the client and their integrated APIs to transform the data.

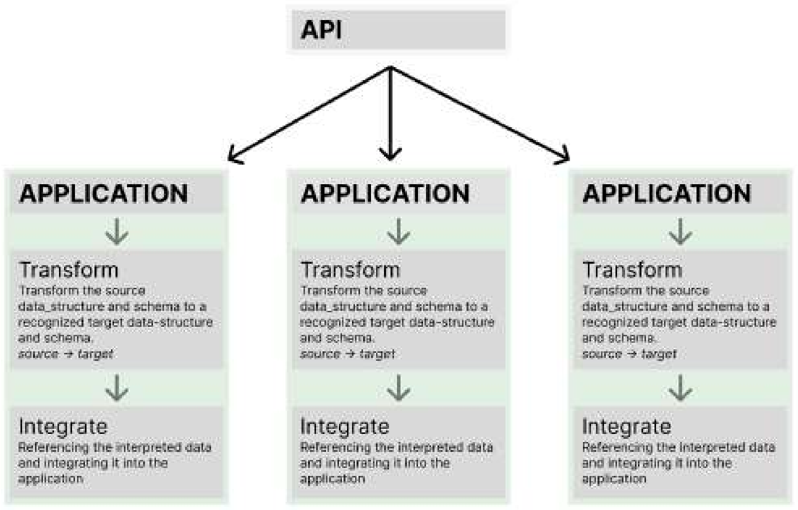

API
APPLICATION
Transform
Transform the source data_structure and schema to a recognized target data-structure and schema.
source → target
Integrate
Referencing the interpreted data and integrating it into the application
APPLICATION
Transform
Transform the source data_structure and schema to a recognized target data-structure and schema.
source → target
Integrate
Referencing the interpreted data and integrating it into the application
APPLICATION
Transform
Transform the source data_structure and schema to a recognized target data-structure and schema.
source → target
Integrate
Referencing the interpreted data and integrating it into the application

image_5: Process of hypothesis with the Client Application applying the transformation

Required-Feature-Elements

In order for the project to be useful and have a chance to achieve its hypothesis, it would need to offer the following feature elements:

A set of methods for easily integrating with the tool:

- Method which takes an xsd_source schema of the source xml_source data document, a xsd_target schema of the xsml_target data and a "mapping object" document (likely in JSON) that utilises XPath to map and state the required transformation needed.

    - This method should return an XSLT style-sheet which would be used to transform data in the xsd_source schema and datatype, to the xsd_target schema and datatype.
- Method which takes an xml_source data document and an XSLT style-sheet as inputs, and transforms the data document according to the XSLT.
    - This method should return a data document that has been transformed into the xsd_target schema.
- Method of validating the received and returned data.
    - Method which takes an XSD schema and a XML data document, and confirms such data document is valid against the schema.
    - Method which validates the data is syntactically correct and well-formed.
    - Method which should return a report, illustrating what data has been lost in the transformation and what data would be lost from using a particular XSLT.

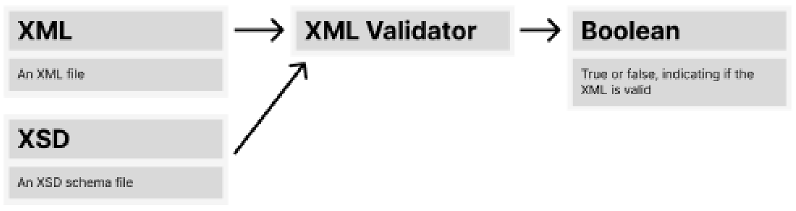


image_6: Illustration of the proposed class design (The validator method)

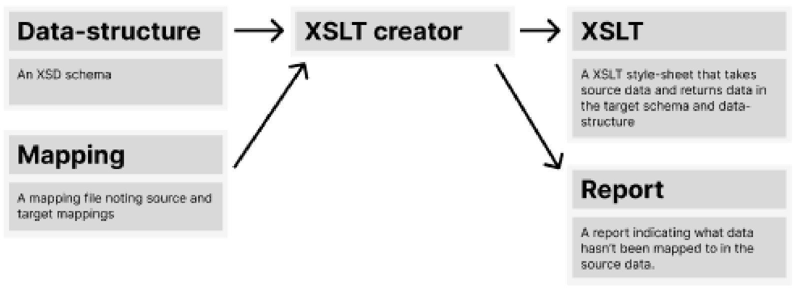

Data-structure
An XSD schema
Mapping
A mapping file noting source and target mappings
XSLT creator
XSLT
A XSLT style-sheet that takes source data and returns data in the target schema and data-structure
Report
A report indicating what data hasn't been mapped to in the source data.

image_7: Illustration of the proposed class design (The XSLT creator method)

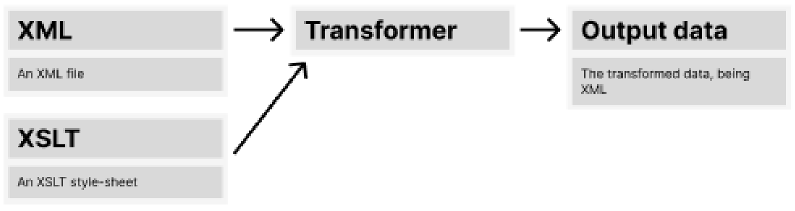


image_8: Illustration of the proposed class design (The transformer method)

Mapping object: for recording a clients mapping of a particular schema

- The mapping object will be able to include some sort of aggregate function possibilities.

Illustration of a mapping file design idea

```
[
 {
  "target": "name",
  "sources": ["root/bookstores/bookstore/name_lead",
"root/bookstores/bookstore/name_trail"]
 },
 {
  "target": "address",
  "sources": ["root/bookstores/bookstore/address"]
 }
]
```

Implementation

The implementation of the final-prototype.

Development-Environment

The environment in which the tooling what developed and tested is as follows:

- Ubuntu: The operating system being Ubuntu <22.04.3 LTS>
- NodeJS: The runtime environment being Node <v18.15.0>
- npm: The JavaScript package manager being npm version <9.5.0>

Architecture

The technology and libraries i had used are as follows:

- XSLT: A transformation style-sheet [4-1]
    - `saxonJS`: Which is an XSLT processor which can run in the browser, as well as in a NodeJS environment. This library had 20 000 downloads in my current week. The primary usage of this package would be for the methods that utilise XSLT to transform data.
- XSD: XML based schema representation [3-2]
    - `libxmljs`: Which is a library with over 35 000 downloads a week. The primary usage of this library would be to validate XML documents against an XSD schema.
- XML: Structured data format [4-2]
    - `xml2js`: Which is a library with over 15 000 000 downloads a week. The primary usage of this library would be to extract XML and transform it into a tree like JSON structure.
- XPath: A mapping language for reading XML [4-3]
    - `Xpath`: Which is a very popular npm package, with over 800 other applications using it. The primary usage of this package would be the creation of the Xpath queries.

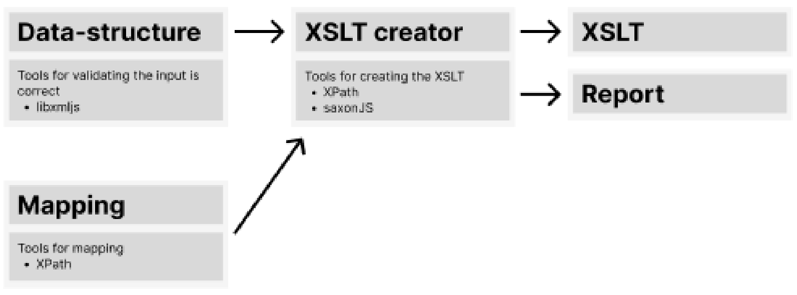


image_9: Illustration of architecture library integrations (The XSLT creator method)

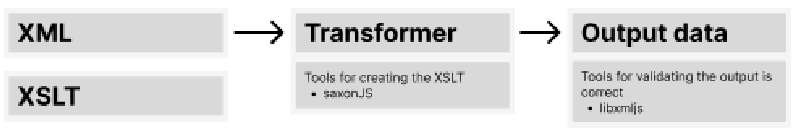

XML
XSLT
Transformer
Tools for creating the XSLT
• saxonJS
Output data
Tools for validating the output is correct
• libxmljs

image_10: Illustration of architecture library integrations (The transformer method)

Code-Explanations

I have created a library called `xml2xml-schema-change`, which is a single package that consisting of three classes, being `xmlNode`, `xsltSchema` and `xsltTransformer`. With just these three classes it is possible to achieve what is required from the server-side feature.

xmlNode
This is a class used to create tree like structure representations of XML documents, similar to the `xml2js` representation of a XML document though is further simplified. This class is primarily a helper class for the other two classes and although could be used independently, there likely isn't many reasons to use this over `xml2js`.

xsltSchema
This is a class that holds all the methods to create XSLT files utilising XSD's and "mapping objects", the class also consists of methods to read and validate XSD, XSLT, and XML files.

xsltTransformer
This class consists of methods used for applying XSLT transformations on an XML, as well as XML to XSD validation methods. The class works with `xsltSchema` instances.

Instillation

The simplest way to install `xml2xml-schema-change` is to use the npm package manager `npm install xml2xml-schema-change` which will download `xml2xml-schema-change` and all dependencies. Else the package could be downloaded from the GitHub repository `github:blake214/xml2xml-schema-change`.

Both these methods of instillation's include their own readme file that gives instructions on how to utilise the package.

Mapping Object

Here I explain whats going on with the "mapping object", referenced in the design. This would likely be created through a graphical UI, though in this project they have been manually compiled. The "mapping object" is an array of zero or more objects, where these objects consist of two key value pairs being "target" and "sources".

Target
This is a string value, which is the elements name within the xsd_target that is getting mapped too. Here is an example of an XSD which would have a target name as "name" ; `<xs:element type="xs:string" name="name"/>`.

Source
This is an array of string values, which are the Xpath locations from the xsd_source. There can be multiple sources, this is to allow some aggregate functionality.

- An empty source list will just mean the target will be a "complex object" type XML node.
  - E.G. -> `[]` , This would create an element as; `<xs:element name="books">`.
- A source list with a single source, this meant the target wont be a "complex object" and will be an end XML node, which retrieves its contents from the source.
  - E.G. -> `["./Xpath"]` , This would create an element as; `<xs:element type="xs:string" name="title"/>`.
- A source list with multiple sources, this gets handled depending on the targets XSD type. I have currently only catered for aggregation functions that concatenate multiple strings into one string, and concatenate multiple objects from multiple locations.
  - E.G. -> `["./Xpath", "./Xpath"]` , if the targets XSD type is a "complex object" type which consists of multiple of the same elements, then both Xpaths will be used when creating the elements. An example of a complex object with multiple elements would appear as this; `<xs:element name="book" maxOccurs="unbounded" minOccurs="0">` where there is a attribute stating "MaxOccurs".
  - E.G. -> `["./Xpath", "./Xpath"]` , if the targets XSD type is a "string" type, then the both Xpaths will concatenated together to make a single string. An example of a string element type appears as; `<xs:element type="xs:string" name="title"/>`

Running Code

Using the library and running the code requires two stages.

Stage One
Here we initialise the `xsltSchema` instance with three primary parameters.

- xsd_source : This is a serialised version of the XSD that the parsed XML would be in.
- xsd_target : This is a serialised version of the XSD that you would require the returned XML to be in.
- mapping_object : This is a "mapping object" mapping the two together (taking the structure from the design section).

```
const { xsltSchema } = require("xml2xml-schema-change")
const xslt_schema = new xsltSchema(xsd_source, xsd_target, mapping_object)
```

```
xslt_schema.init()
.then(() => {
 const xslt_schema_serialised = JSON.stringify(xslt_schema, null, 2)
 // Can check what elements havent been mapped to the xsd_target
 console.log(xslt_schema.non_mapped)
 // -> Save this xslt_schema_serialised in your database
 }).catch((err) => {
  // Handle errors
  console.error(err);
 });
```

After initialising, the instance this can then be serialised and stored in a database. We are also able to access the "report" which is an object named non_mapped, to identify which elements from our target_xsd don't have a mapping.

Stage Two
Once initialised, we can use the xslt_schema_serialised instance to transform an XML to the new XSD schema. Here we create an instance of the transformer with one parameter:

- xslt_schema_serialised : This is the stored serialised instance from stage one.

Then you call the transform method with one parameter:

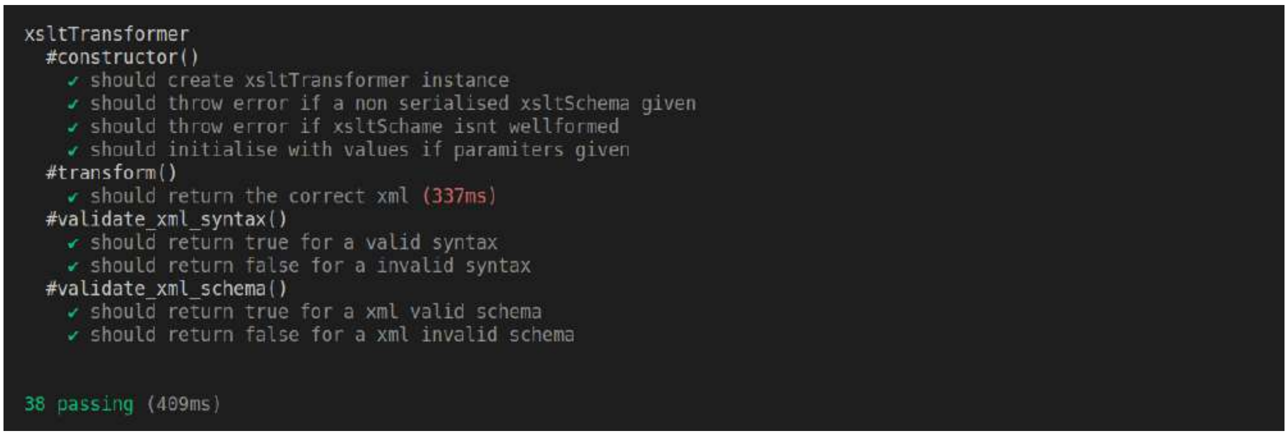
- xml_source : This is a serialised version of the source XML you want to transform.

```
const { xsltTransformer } = require("xml2xml-schema-change")
const xslt_transformer = new xsltTransformer(xslt_schema_serialised)

xslt_transformer.transform(xml_source)
.then((result) => {
 console.log(result)
 // -> Now do what you want with the transformed xml
})
```

Running Test Suit

The library includes a unit test suit, which consists of over 30 unique test cases, to run the test suits use the following command in the libraries root directory `npm test`, and results should appear as *image_11*.



```
xsltTransformer
  #constructor()
    ✓ should create xsltTransformer instance
    ✓ should throw error if a non serialised xsltSchema given
    ✓ should throw error if xsltSchame isnt wellformed
    ✓ should initialise with values if paramiters given
  #transform()
    ✓ should return the correct xml (337ms)
  #validate_xml_syntax()
    ✓ should return true for a valid syntax
    ✓ should return false for a invalid syntax
  #validate_xml_schema()
    ✓ should return true for a xml valid schema
    ✓ should return false for a xml invalid schema


38 passing (409ms)
```

image_11: Results snippet from the test

Conclusion

I have made the package available via the following channels:

- **GitHub**: `github:blake214/xml2xml-schema-change`
- **npm**: as a packaged named `xml2xml-schema-change`

Presentation video available at: https://youtu.be/YBBme-ekF18

References

---

1. Daniel Lehner, Antonio Garmendia, Manuel Wimmer: 2021, "Towards Flexible Evolution of Digital Twins with Fluent APIs", Johannes Kepler University, IEEE 26th International Conference on Emerging Technologies and Factory Automation↩↩↩
2. Horsch, M. T.; Francisco Morgado, J.; Goldbeck, G.; Iglezakis, D.; Konchakova, N. A.; Schembera, B.: 2021, "Domain-specific metadata standardization in materials modelling", Kaiserslautern, Proc. DORIC-MM 2021 pp. 12–27↩↩↩
3. John Boyer, Sandy Gao, Susan Malaika, Michael Maximilien, Rich Salz, Jerome Simeon: 2011, "Experiences with JSON and XML Transformations", Bedford USA, IBM Submission to W3C Workshop on Data and Services Integration↩↩↩
4. Clark, J., 1999. "Xsl transformations (xslt)". *World Wide Web Consortium (W3C). URL http://www.w3.org/TR/xslt 103.*↩↩↩↩