**PROPOSAL**

# JSON Support for Golang libraries

Mentors: Rishabh Bhatnagar and Steve Winslow

By

**Ujjwal Agarwal**

## Index of Contents

## Personal Details

**Name:** Ujjwal Agarwal

**Email Address:** agarwalujjwal012@gmail.com

**Github username:** specter25

**Other contact methods:**

**Linkedin :** https://www.linkedin.com/in/agarwalujjwal012/
**Skype Id:** https://join.skype.com/invite/nb3kujV912Kg

**College:** Thapar Institute of Engineering and Technology , Patiala , India

**Current Programme :** Bachelors of Technology in Computer Science And Engineering.

**Graduation Year :** 2023

**Country of residence:** India

**Timezone:** Asia/Kolkata(+0530)

**Primary Language:** English

## 1. Project Synopsis

tools-golang is a collection of Go packages intended to make it easier for Go programs to work with SPDX® files. The Project is under active development and implements the following functionalities

- Parses spdx files in tag-value format
- Saves a spdx file in tag-value format
- Parses a spdx file in rdf format
- Builds "empty" SPDX document (with hashes) for directory contents
- Searches for SPDX short-form IDs and builds SPDX document
- Compares concluded licenses between files in two packages
- Generates basic license count report from SPDX document
- Various utility functions that support the other tools-golang packages

After the introduction of Spdx Specifications v2.2 JSON, YAML, and a development version of XML have been added as supported file formats. However , the tools-golang package currently does not have the support to parse the spdx files nor has the support to save a spdx doc in JSON format .

The **main objective** of this project is to add support in the tools-golang package so that it can parse as well as save SPDX® v2.2 files in JSON format .

## 2. Project Description

JSON (*JavaScript Object Notation*) is one of the most popular data exchange formats on the web. It's a text encoded format, which means JSON data is a string of characters written in valid JSON format. You can follow the format of JSON from RFC 7159 documentation.

To add JSON support to the tools-golang package the project will make use of the official go JSON package( https://golang.org/pkg/encoding/json/ ) . The spdx specification has a JSON schema defined for the specification and has also provided a sample spdx document in JSON example format .
The Project mainly comprises the **JSON loader** and the **JSON saver .** The former converts JSON to spdx Document struct while the latter generates JSON from spdx Document struct . The key points to consider in this project are :

## 2.1. Why is using struct tags not a solution ?

The first basic solution that one thinks of while going through the JSON package docs and project description is that the JSON can be unmarshalled into the general spdx struct with the help of struct tags and the Marshall and Unmarshall functions provided by the package , the reasons behind not using struct tags are

When we observe the structure of the spdx struct a lot of fields which are normal arrays in the json specifications are configured as maps in spdxDocument2.1 and spdxDocument2.2 . Some of these properties are :

- Files
- Packages
- External Document References
- Unpackaged files
- File Checksums
- Package Checksums

Some fields like Spdx Identifiers and DocumentReferences need some string pre-formatting to be done to them before they are saved into the document .

The JSON specification differs greatly in structure from the spdx Document struct like some properties inside the  creation Information struct in the sdpx Document are in the document root in the JSON specification .

## 2.2. Decoding JSON (Unmarshall)

To decode JSON into a valid data structure like map or struct, we first need to make sure if a JSON is valid.

```
func Valid(data []byte) bool
```

We can use the json.Valid function to check if JSON is valid. This function returns true if JSON data is valid or false otherwise.

We have to use the json.Unmarshal function to decode JSON data into a data structure like map or struct.

```
func Unmarshal(data []byte, v interface{}) error
```

The Unmarshal function takes the JSON data as the first argument and the container v that will hold the data as the second argument. The v argument is either a pointer to a valid data structure or to an interface. If v is either nil (except for a nil interface) or not a pointer then Unmarshal returns json.InvalidUnmarshalError error. It also returns an error if the JSON can not be unmarshalled into the value stored in v.

One important thing to consider while unmarshalling JSON through structs is that unmarshalled Unexported fields not unmarshalled i.e. if any field in the defined struct has a lowercase initial letter .

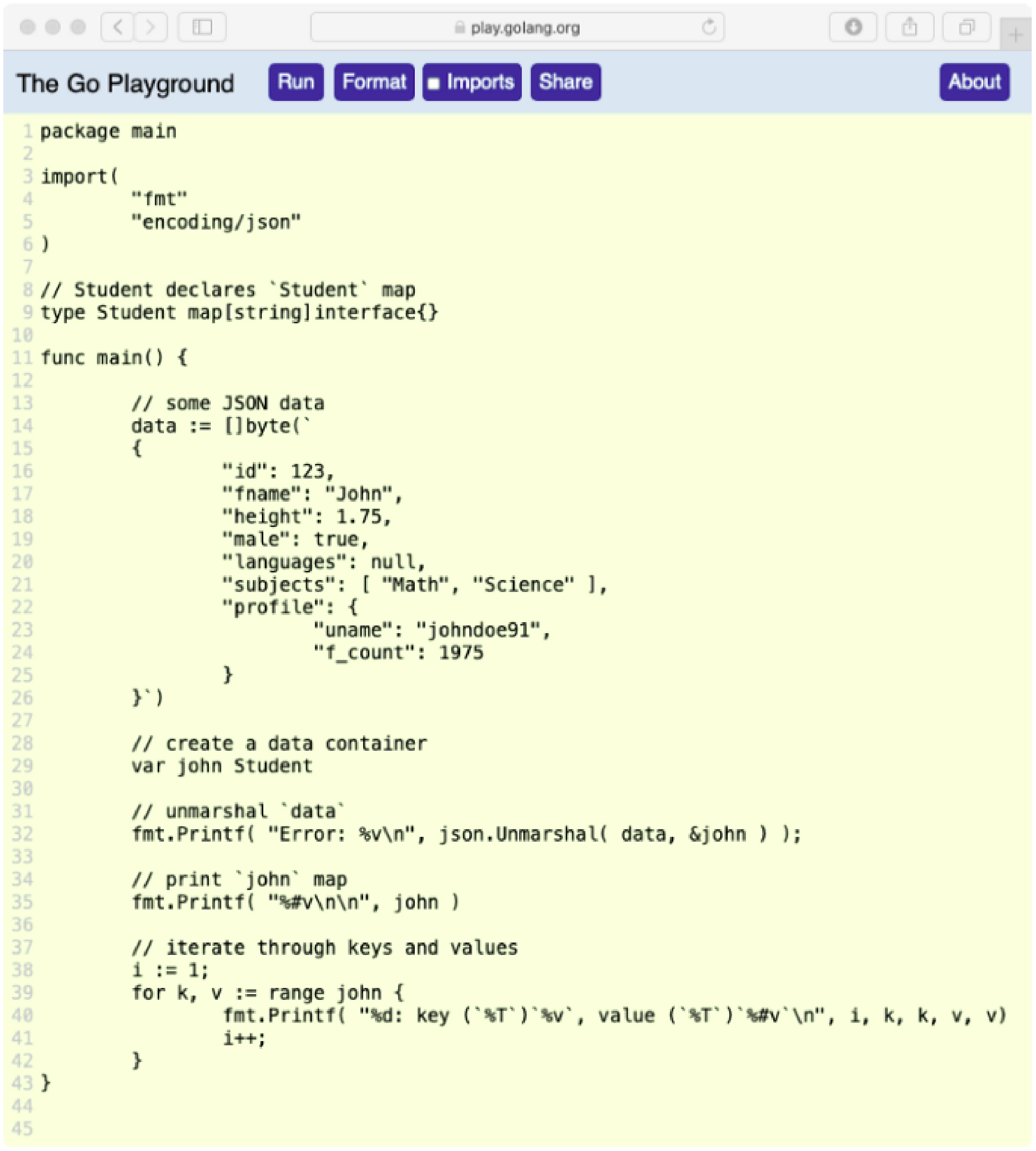
Since a JSON contains string keys and values of supported data types, a map of type map[string]interface{} is a suitable candidate for storing JSON data. We can pass a pointer to nil or non-nil pointer of the map to the Unmarshal function and all JSON field values will be populated inside the map.

Creating a new struct for unmarshalling the JSON spec is not advisable as then we will have to

- Define a format based spdx specification for all versions of spdx document
- Increase code maintenance and complexity
- Search time of maps is less as compared to structs and arrays

### 2.2.1. Working with Maps while Unmarshalling JSON

Since a JSON contains string keys and values of supported data types, a map of type map[string]interface{} is a suitable candidate for storing JSON data. We can pass a pointer to nil or non-nil pointer of the map to the Unmarshal function and all JSON field values will be populated inside the map.



```go
package main

import(
	"fmt"
	"encoding/json"
)

// Student declares `Student` map
type Student map[string]interface{}

func main() {

	// some JSON data
	data := []byte(`
	{
		"id": 123,
		"fname": "John",
		"height": 1.75,
		"male": true,
		"languages": null,
		"subjects": [ "Math", "Science" ],
		"profile": {
			"uname": "johndoe91",
			"f_count": 1975
		}
	}`)

	// create a data container
	var john Student

	// unmarshal `data`
	fmt.Printf( "Error: %v\n", json.Unmarshal( data, &john ) );

	// print `john` map
	fmt.Printf( "%#v\n\n", john )

	// iterate through keys and values
	i := 1;
	for k, v := range john {
		fmt.Printf( "%d: key (`%T`)`%v`, value (`%T`)`%#v`\n", i, k, k, v, v)
		i++;
	}
}
```

( [https://play.golang.org/p/yFDL7d-yVDO](https://play.golang.org/p/yFDL7d-yVDO) )

But take a look at the data types of the map values. There are certain rules Unmarshal functions follow to store the JSON values in a map.

- A JSON string value is stored as a string.
- A JSON number value(int or float) is stored as float64.
- A JSON boolean value is stored as bool.
- A JSON null value is stored as nil value.
- A JSON array value is stored as a slice of type []interface{}.

- A JSON object value is stored as a map of type map[string]interface{}.

The array values were stored in a slice of type []interface{} and object values were stored in a map of type map[string]interface{}.

As we know, a valid JSON format can be an object (like in the example above) or an array. Since Unmarshal is capable of allocating memory for a pointer as well as it can create containers to hold decoded JSON data on its own, we can store complex JSON data without defining a container-type.

Thus , the solution I propose is to define a **UnmarshallJSON** function on the spdx.Document2_2 struct so that when the JSON is unmarshalled in it the function is called so that we can implement the process in a custom way . Then we define a new map[string]interface{} and parse it into the spdx.Document2_2 using functions defined for it's different sections .

**JSON → map[string]interface{} → spdx.Document2_2**

## 2.3. Encoding JSON (Marshall)

As we have opted not to define a custom struct based on the Json specification , we have to use some other solution to marshall the spdx document to JSON considering the fact that the structure of the JSON specification greatly differs from the spdx document struct . The main difference is in the creation info struct like some properties inside the creation Information struct in the sdpx Document are in the document root in the JSON specification . Moreover the order in which the properties JSON specification is not the same as that defined in the JSON specification . As a result if we marshall it using struct tags the order of the properties will not remain the same as . THe solution I propose is that we make a custom json marshaller . While marshalling JSON we should create a custom MarshallJSON method when instead of using the default Marshall function , we create a new buffer and write the Marshalled keys and their corresponding values to the buffer as rune's . A sample of this method implemented on a map is:

```
func (om OrderedMap) MarshalJSON() ([]byte, error) {
	var b []byte
	buf:=bytes.NewBuffer(b)
	buf.WriteRune('{')
	l:=len(om.Order)
	for i,key:=range om.Order {
		km,err:=json.Marshal(key)
		if err!=nil { return nil,err }
		buf.Write(km)
		buf.WriteRune(':')
		vm,err:=json.Marshal(om.Map[key])
		if err!=nil { return nil,err }
		buf.Write(vm)
		if i!=l-1 { buf.WriteRune(',') }
		fmt.Println(buf.String())
	}
	buf.WriteRune('}')
	fmt.Println(buf.String())
	return buf.Bytes(),nil
}

func main() {
	obj := `{"key3": "value3", "key2": "value2", "key1": "value1"}`
	var o OrderedMap
	json.Unmarshal([]byte(obj), &o)
	r, err := json.Marshal(o)
	fmt.Println(string(r),err)
}
```

( https://play.golang.org/p/TxenZEuy_u0 )

I suggest implementing something very similar to this method (go playground link mentioned above) the only difference being that inside of a map we will have a struct to parse .

## 2.4. Utility functions

Some utility function will be needed specially during the unmarshalling procedure:

extractDocElementID : Used to extract Document References and SPDX References values from an SPDX Identifier which can point either to this document or to a different one . This function comes in handy when we are unmarshalling the JSON specification .

extractElementID: used to extract SPDXRef values only from an SPDX Identifier which can point to this document only. Use extractDocElementID for parsing IDs that can refer either to this document or a different one.

### 2.5. Efficient Coding practices

Use Sync pools : One of the highlights of Go 1.3 release was sync Pool. It is a component under the sync package to create a self-managed temporary retrieval object pool.We want to keep the garbage collector overhead as little as possible. Frequent allocation and recycling of memory will cause a heavy burden to the garbage collector. sync.Pool can cache objects that are not used temporarily and use them directly (without reallocation) when they are needed next time. This can potentially reduce the garbage collector workload and improve the performance.

Use efficient Go coding practices : To write Go, it's important to understand its properties and idioms. It's also important to know the established conventions for programming in Go, such as naming, formatting, program construction, and so on, so that programs you write will be easy for other Go programmers to understand. The go official docs provide some guidelines in this blog to write structurally effective go programs .

Version Control good practices :The contributors to repositories should know that a well-crafted Git commit message is the best way to communicate context about a change to fellow developers (and indeed to their future selves). A difference will tell you what changed, but only the commit message can properly tell you why.Fortunately, there are well-established conventions as to what makes an idiomatic Git commit message. Indeed, many of them are assumed in the way certain Git commands function. This blog clearly mentions the semantics that need to be followed when writing good commit messages

## 3. Future Implementations

### 3.1. Configure golang ci-lint

golangci-lint is a fast Go linters runner. It runs linters in parallel, uses caching, supports yaml config, has integrations with all major IDE and has dozens of linters included.This will be a test in the github actions so that whenever a pull request is raised it passes through a series of lint tests that are essentials to ensure code quality . The initial step will be to format the current code to ensure that the tests pass on the most latest code in the repository and then lay down the golangci-lint specification in the github actions file to further ensure , that whichever pull request is being merged has appropriate and has correctly formatted code .

### 3.2. Configure pre-commit hooks

The pre-commit hook is run first, before you even type in a commit message. It's used to inspect the snapshot that's about to be committed, to see if you've forgotten something, to make sure tests run, or to examine whatever you need to inspect in the code. Exiting non-zero from this hook aborts the commit, although you can bypass it with git commit --no-verify. You can do things like check for code style (run lint or something equivalent), check for trailing whitespace (the default hook does exactly this), or check for appropriate documentation on new methods. It's a useful tool to have in the development environment . We can run a series of tests to run such as :

- Whether the document is formatted correctly according to the specifications laid down by the specifications .
- Is there any unused dependency in the code which hasn't been removed due to incompetent local setup of the developer
- There are no redundant new lines introduced in the code
- The document follows the coding and semantic conventions laid down by the effective-go article .

Pre-commit hooks are generally used for structural and semantic checks in the code . Once the code is pushed the CI/CD file will include a check to test whether all the pre-commit tests have passed or not .

## 4. Schedule of Derivables

| | |
|---|---|
| May 17 - June 7<br><br>Community Bonding Period | ● Discuss with the mentor the exact procedure and workflow.<br>● I have mentioned the path and way for the solutions in the proposal but they are flexible and can change as per the common view with the mentor.<br>● Along with this, I will learn the necessary skills which will help me to make the solutions easier and |

| | |
|---|---|
| | accurate. |
| June 8th - June 14th | Start with the JSON parser (Unmarshaller) and create function to parse :<br>● Creation Information<br>● Other Licenses<br>● Annotations<br>Also create the utility functions needed i.e. extractDocElementID and extractElementID |
| June 16th - June 22rd | Create functions needed by JSON parser to parse :<br>● Files<br>● Reviews |
| June 24th - 30th June | Create functions needed by JSON parser to parse :<br>● Packages<br>● Relationships |
| July 2 - 17 July ( 12th July - 16th July Round one Evaluations ) | Write testing files for the JSON parser |
| July 18th - 25th July | Start with the JSON saver (Marshaller) and create function to marshall :<br>● Creation Information<br>● Other Licenses<br>● Annotations |
| July 26th - August 2nd | Create functions needed by JSON saver to marshall :<br>● Files<br>● Reviews |
| August 3rd - August 15th | Create functions needed by JSON saver to marshall :<br>● Packages<br>● Relationships<br>Write testing Files for the JSON saver |
| August 16 - 23, 2021 (Final Round of Evaluation) | Final Round of Evaluation |

**Apart from this, I will be maintaining a 3-part blog, writing about my work and the entire workflow and evaluations.**

## 5. Availability Schedule

During the entire GSoC period, I am available to work for **35-40 hours per week**. I don't have any pre-planned vacations or plans during the break and will be available full time.
Due to **COVID19 shutdown** in my country, our college schedule is likely to be shifted ahead by a **minimum of 2 weeks**, thus will be consuming my summer break's first few days. But that **won't be an issue** with my work schedule. I will make sure that compensation of that time is done during the following weekends of the weeks indulged in my college timings (if required).
Also, I have **kept a buffer** of few days between the tasks in my timeline, which will give me time to complete pending/delayed tasks (if any)

## 6. Plan for communication with mentors

- I am open to having weekly meetings on any platform that the organization is comfortable with (zoom , google meet , skype etc .) .
- I plan to give daily updates regarding the project to the mentor via emails .
- I am a member of the gitter channel of SPDX as well , if the mentor wants we can chat on gitter itself . However I am open to using slack , discord , rocketchat etc .

## 7. Contributions to SPDX

| Organization | Pull Request /Issue code | Description |
|---|---|---|
| SPDX | #60 (MERGED) | Fixed the issue #49 |
| SPDX | #64 (OPEN) | Fixed the issue #26 |

## 8. Contributions to Open Source Community

| Organization | Pull Request /Issue code | Description |
|---|---|---|
| Litmus-Chaos | #2632(MERGED) | Add goimports CI check and format existing imports . |
| Litmus-Chaos | #2639 (MERGED) | Fix failing CI tests , remove go imports check from push.yml |
| Litmus-Chaos | #2649 (ISSUE) | Add golang-ci lint check in build pipeline and remove backend and frontend checks from push pipeline |
| Litmus-Chaos | #2640 | Involved in the conversations of this pull request |
| Litmus-Chaos | #2661 (OPEN) To be tested after release on 15th April | Added golangci-lint to the pipeline and fixed all the lint errors in the code . |
| Casbin | #10 (MERGED) | Fixed the issue #9 |
| Casbin | #9 (ISSUE) | Created an issue for documentation fix . |

## 9. Academic Experience

I am a 2nd year engineering undergraduate(Bachelor of Technology in Computer Science and Engineering) at Thapar Institute of Engineering and Technology, Patiala. I have always found computer science as a very intriguing subject and always felt the urge to learn new stuff in this field . How are websites built ? What are web servers ? Why does a server crash ? How to prevent a server from crashing ? were some of the questions that drove me to learning more and more in this field and I eventually took it as my major subject in my B.Tech. I program a lot, I try my level best to stay focussed on my coursework as a result I have successfully maintained a **C.G.P.A. of 10.0** and have **over 2000 contributions on Github over the last year with a longest standing streak of 154 days .**

**A link to my complete academic resume is here**

## 10. Open Source Projects

### 10.1. Slack-bots (Project Started by Developers Student club Thapar)( Contributed in this project as a project Co-lead)

These are general purpose bots that scrape articles from medium, dev. to, telegram channels and then render them in various slack, discord channels and also in google home, alexa . The telegram is powered by NLP for summarizing the content and also filtering it. The bots will run automatically as set by the administrator.

## 11. Personal Projects

### 11.1. Azure Devops- Terraform- Kubernetes CI/CD Pipeline (project Head , MERN stack Devops)

I implemented a CI/CD pipeline on a MERN stack application . Used docker for containerisation . Then wrote Kubernetes config files and created different pods and then finally set up a CI/CD pipeline using Azure devops . I used Terraform to implement infrastructure as code and automated the creation of AWS S3 buckets , EC2 instances and I am users .

### 11.2. Devconnector

A MERN stack application , which is the professional social media handle of a local group of users. It has the functionality of authentication , blogs , posts , likes , comments etc . It has the functionality of image upload . It's currently hosted on heroku.

### 11.3. Go-lang Blockchain

A custom blockchain implementation using go-lang . A low-level computing language . I optimised the process of signing the unsigned transactions by building a custom Merkle tree which reduced the mining time of the blockchain by 3 sec. Presently I am trying to implement the GHOST protocol of Ethereum but am currently unsuccessful. I even plan to implement a Proof of Authority consensus algorithm in the future .

### 11.4. Organ-donation blockchain

To build an ecosystem where the donors , recipients and others can interact with each other directly on secure lines to make the entire procedure more fluid and secure . This project is a Decentralized application based on the ethereum blockchain .

## 12. Work/Internship Experience

Software Engineering Intern at GrowthGear Solutions LLP ( 28 December 2020 – 28 March 2021)

- Completed the RTC real time video streaming and video calling module . Both client side and server side . (using Web RTC connections)
- Completed real time messaging (public and private chats , polls and Q&A (using Web sockets) module and integrated it with video calling and streaming module .
- Made the analytics and role based access server side .

## 13. Why Me

I really value this opportunity and getting it really means a lot to me . I think that I can guarantee that I will never be careless and lethargic throughout the course of three months and put in 100% efforts towards it's completion . I think this project is apt for me. It involves a lot of constant learning about new stacks and technology. I am consistent with my efforts and never leave any task in between, no matter how hard or new it is for me. I make sure that I keep researching the issue at hand and then work to make it happen. I think this skill of mine proves to be very helpful in order to achieve feature addition and bug solving for this particular project. This project will help me to equip essential real-world software skills as well as soft skills which I believe will help me a lot in the longer run.