



Processing XML and Spreadsheet in Go



续 日

Gopher China Conference
Beijing 2021 6/26 - 6/27

Self Introduction



GitHub: @xuri

Twitter: @xurime

Blog: <https://xuri.me>

The author of the Excelize - Go language spreadsheet library. Familiar with Go language programming, middleware, and big data solution.

Working Experiences

Alibaba Group - Software Engineer

Baidu Inc. - Software Engineer

Qihoo 360 – Server-side Software Engineer



GopherChina 2021

Agenda

Serialize and Deserialize

01

- Document Object Model
- Event-driven (Simple API for XML)
- Serialize and Deserialize Control

Handle Complex XML

02

- Partial Load
- Namespace & Entity
- Ser/Deserialize Idempotence

High Performance Processing

03

- XML Schema Definition
- DOM or SAX

OOXML Spreadsheets

04

- Excel XML Specification
- Charset Encoding
- Streaming I/O

Serialize and Deserialize

Document Object Model

```
<?xml version="1.0" encoding="utf-8"?>
<Person>
  <Name>Tom</Name>
  <Email where="home">
    <Addr>tom@example.com</Addr>
  </Email>
</Person>
```



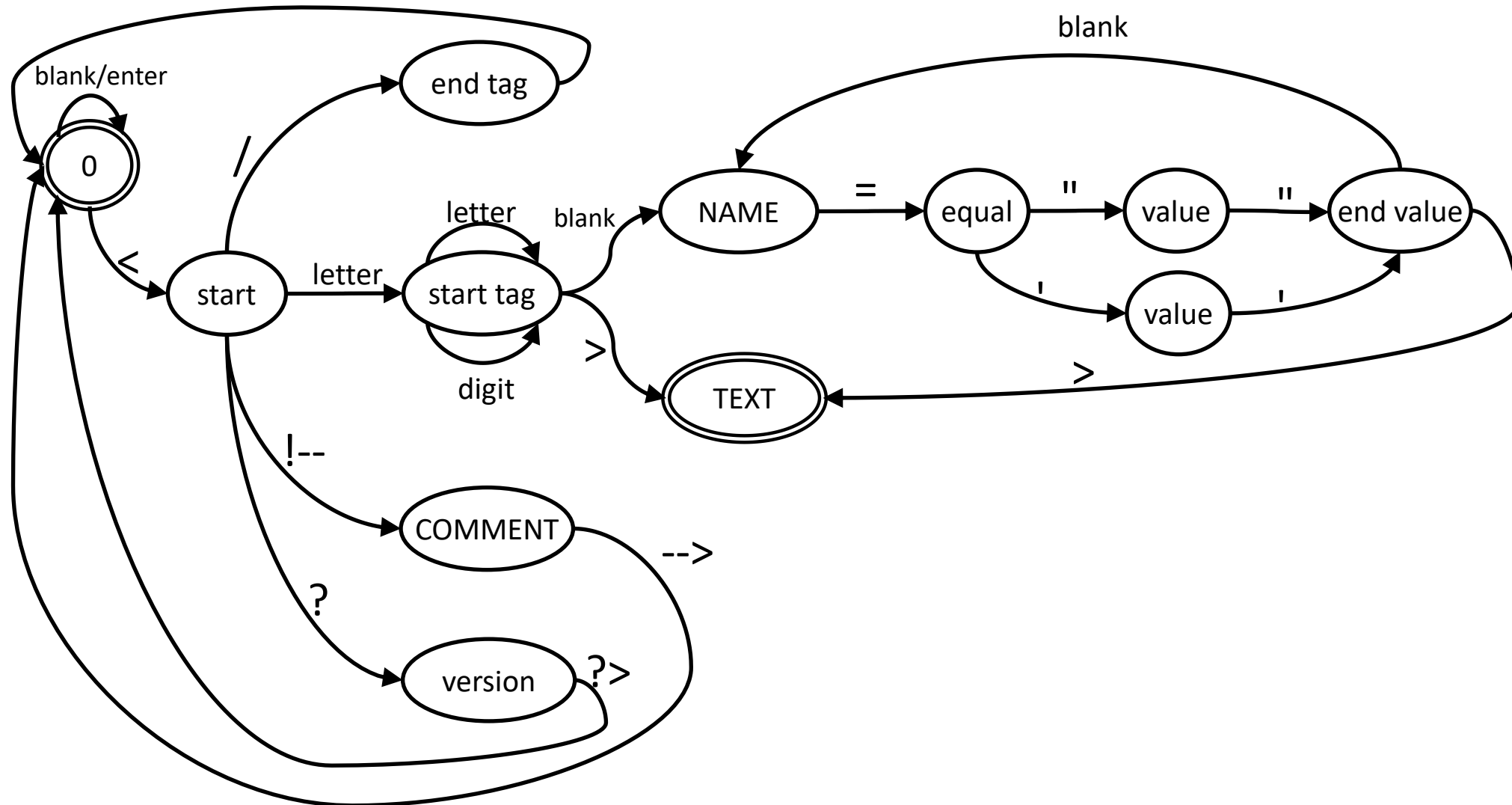
```
type Person struct {
  Name string
  Email struct {
    Where string `xml:"where,attr"`
    Addr string
  }
}
```

encoding/xml

```
var p Person
if err := xml.Unmarshal([]byte(data), &p); err != nil {
  fmt.Println(err)
}
fmt.Printf("%+v\n", p)

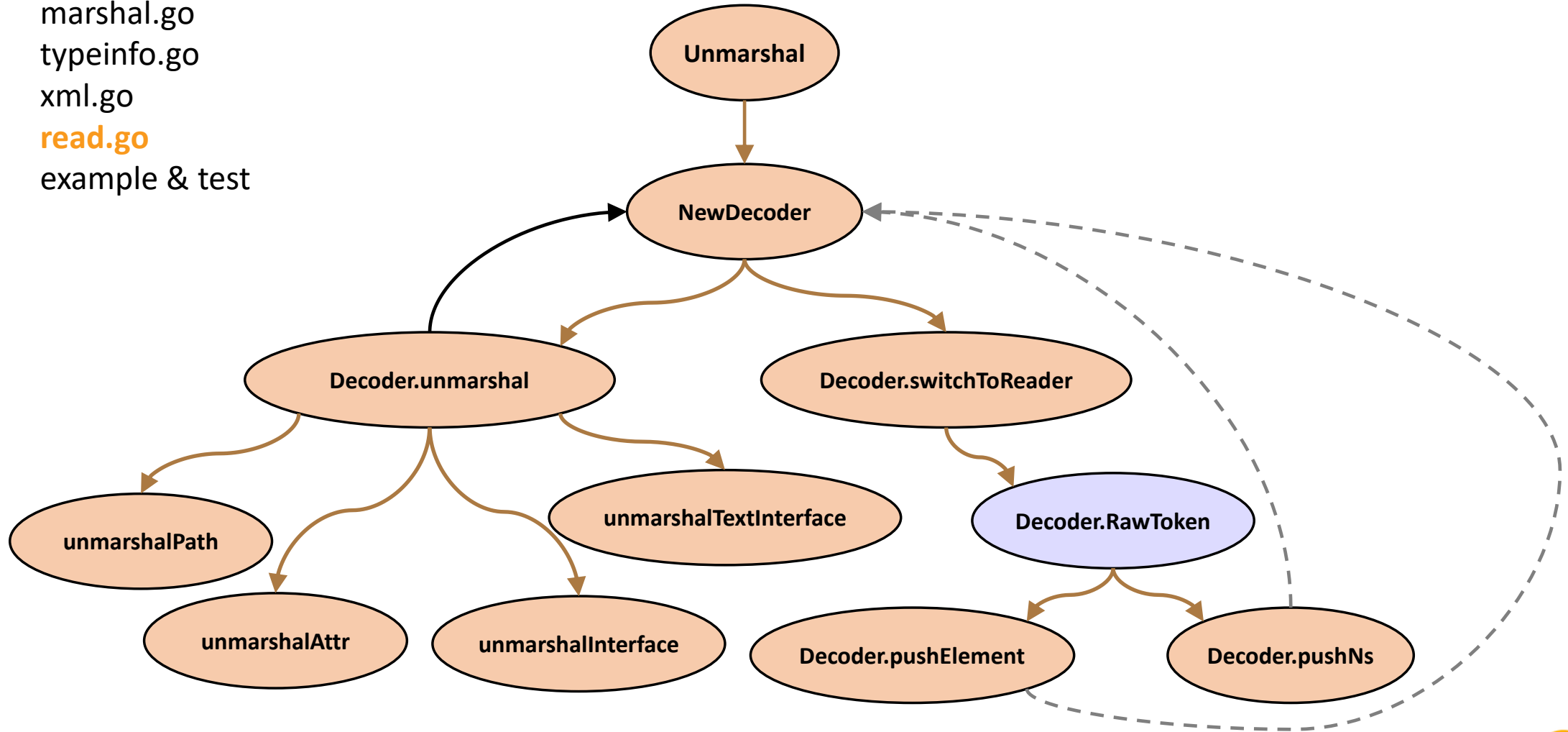
// {Name:Tom Email:{Where:home Addr:tom@example.com}}
```

XML Finite State Machine



Unmarshal

encoding/xml
marshal.go
typeinfo.go
xml.go
read.go
example & test



Go XML Parser

encoding/xml:xml.go

```
type Decoder struct {  
    Strict bool  
    AutoClose []string  
    Entity map[string]string  
    CharsetReader func(charset string, input io.Reader) (io.Reader, error)  
    DefaultSpace string  
    r io.ByteReader  
    t TokenReader  
    buf bytes.Buffer  
    saved *bytes.Buffer  
    stk *stack  
    free *stack  
    needClose bool  
    toClose Name  
    nextToken Token  
    nextByte int  
    ns map[string]string  
    err error  
    line int  
    offset int64  
    unmarshalDepth int  
}
```

StartElement

EndElement

CharData

Comment

Proclnst

Directive

Event-driven (Simple API for XML)

```
<?xml version="1.0" encoding="utf-8"?>
<Person>
  <Name>Tom</Name>
  <Email where="home">
    <Addr>tom@example.com</Addr>
  </Email>
</Person>
```



```
decoder := xml.NewDecoder(strings.NewReader(data))
for {
  token, _ := decoder.Token()
  if token == nil {
    break
  }
  switch element := token.(type) {
  case xml.StartElement:
    fmt.Printf("%+v\n", element)
  case xml.EndElement:
    fmt.Printf("%+v\n", element)
  }
}
```

```
{Name:{Space: Local:Person} Attr:[]}
{Name:{Space: Local:Name} Attr:[]}
{Name:{Space: Local:Name}}
{Name:{Space: Local:Email} Attr:[{Name:{Space: Local:where} Value:home}]}
{Name:{Space: Local:Addr} Attr:[]}
{Name:{Space: Local:Addr}}
{Name:{Space: Local:Email}}
{Name:{Space: Local:Person}}
```

Serialize and Deserialize Control

encoding/xml:typeinfo.go

```
switch flag {  
case "attr":  
    finfo.flags |= fAttr  
case "cdata":  
    finfo.flags |= fCDATA  
case "chardata":  
    finfo.flags |= fCharData  
case "innerxml":  
    finfo.flags |= fInnerXML  
case "comment":  
    finfo.flags |= fComment  
case "any":  
    finfo.flags |= fAny  
case "omitempty":  
    finfo.flags |= fOmitEmpty  
}
```

```
type Person struct {  
    Name string  
    Email struct {  
        Where string `xml:"where,attr,omitempty"`  
        Addr  string  
    }  
}
```

attribute with the field name in the XML element

written as character data, not as an XML element

written as character data wrapped in one or more `<![CDATA[...]>` tags

written verbatim, not subject to the usual marshaling procedure

unmatched rule, maps the sub-element to that struct field

omitted if the field value is empty

Partial Load

```
<?xml version="1.0" encoding="utf-8"?>
<Person>
  <Name>Tom</Name>
  <Email>
    <Addr>tom@example.com</Addr>
  </Email>
</Person>
```

```
var p Person
err := xml.Unmarshal([]byte(data), &p)
if err != nil {
    fmt.Println(err)
}
fmt.Printf("%+v\n", p)
```

```
type Person struct {
    Name string
    Email partialXML
}

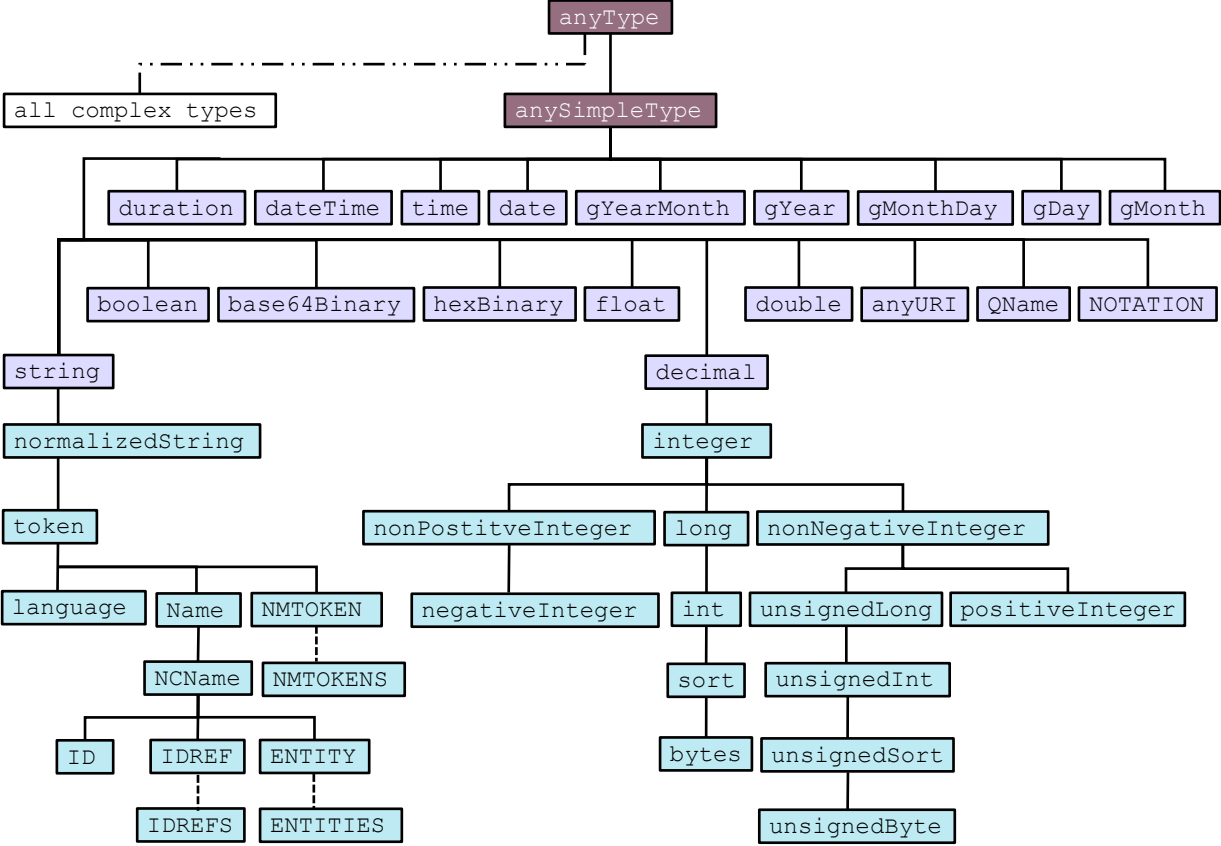
type partialXML struct {
    Content string `xml:",innerxml"`
}
```

```
{Name:Tom Email:{Content:
  <Addr>tom@example.com</Addr>
}}
```

Handle Complex XML

Datatypes

| Go Datatypes | XML Datatypes |
|--------------|---|
| string | anyType, ENTITY, ID, IDREF, NCName, NMTOKEN, Name, anyURI, duration, language, normalizedString, string, token, xml:lang, xml:space, xml:base, xml:id |
| []string | ENTITIES, IDREFS, NMTOKENS, NOTATION |
| xml.Name | QName |
| []byte | base64Binary, hexBinary, unsignedByte |
| bool | boolean |
| byte | byte |
| float64 | decimal, double, float, |
| int64 | int, integer, long, negativeInteger, nonNegativeInteger, nonPositiveInteger, positiveInteger, short |
| uint64 | unsignedInt, unsignedLong, unsignedShort |
| time.Time | date, dateTime, gDay, gMonth, gMonthDay, gYear, gYearMonth, time |



- ur types
- build-in primitive types
- build-in derived types
- complex types
- derived by restriction
- Derived by list
- Derived by extension or restriction



Entity

XML Entity

```
<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE person[
  <!ENTITY name "Tom">
  <!ENTITY email "tom@example.com">
]>

<person>
  <name>&name;</name>
  <address>&email;</address>
</person>

type Person struct {
  XMLName xml.Name `xml:"person"`
  Name     string   `xml:"name"`
  Address  string   `xml:"address"`
}
```

Get Entity

```
exp := `<!ENTITY\s+([^\s]+\s+"([^"]+)")>`
entities := map[string]string{}
d := xml.NewDecoder(strings.NewReader(input))
var rEntity = regexp.MustCompile(exp)
for {
  tok, err := d.Token()
  if err != nil {
    break
  }
  dir, ok := tok.(xml.Directive)
  if !ok {
    continue
  }
  fmt.Println(string(dir))
  for _, m := range rEntity.FindAllSubmatch(dir, -1) {
    entities[string(m[1])] = string(m[2])
  }
}
fmt.Println("entities", entities)
```

```
entities map[name:Tom email:tom@example.com]
```



Entity

XML Entity

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE person[
  <!ENTITY name "Tom">
  <!ENTITY email "tom@example.com">
]>
<person>
  <name>&name;</name>
  <address>&email;</address>
</person>
```

```
type Person struct {
  XMLName xml.Name `xml:"person"`
  Name    string  `xml:"name"`
  Address string  `xml:"address"`
}
```

Decode with Entity

```
d = xml.NewDecoder(strings.NewReader(input))
d.Strict = false

d.Entity = entities
err := d.Decode(&v)
if err != nil {
    fmt.Printf("error: %v", err)
    return
}
fmt.Printf("%+v\n", v)
```

```
{XMLName:{Space: Local:company} Name:Jack Address:Tom}
```

Namespace & Ser/Deserialize Idempotence

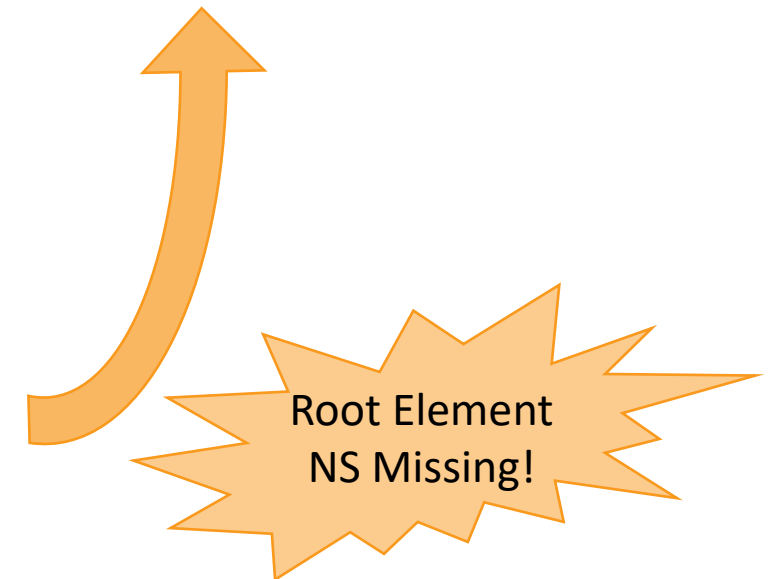
```
<?xml version="1.0" encoding="utf-8"?>
<person
  xmlns="http://example.com/default"
  xmlns:m="http://example.com/main"
  xmlns:h="http://example.com/home"
  xmlns:w="http://example.com/work">
  <name>Tom</name>
  <m:email h:addr="HOME" w:addr="WORK" />
</person>
```

```
<person xmlns="http://example.com/default">
<name>Tom</name>
<email xmlns="http://example.com/main"
  xmlns:home="http://example.com/home"
  home:addr="HOME"
  xmlns:work="http://example.com/work"
  work:addr="WORK"></email>
</person>
```

Inline Namespace Declare

```
type Person struct {
  XMLName xml.Name `xml:"http://example.com/default person"`
  Name     string   `xml:"name"`
  Email    struct {
    XMLName xml.Name `xml:"http://example.com/main email"`
    HomeAddr string `xml:"http://example.com/home addr,attr"`
    WorkAddr string `xml:"http://example.com/work addr,attr"`
  } // TAG NOT HERE: `xml:"email"`
}
```

Namespace Local Name



Ser/Deserialize Idempotence

encoding/xml/xml.go

```
type Name struct {  
    Space, Local string  
}
```

```
type Attr struct {  
    Name Name  
    Value string  
}
```

```
type StartElement struct {  
    Name Name  
    Attr []Attr  
}
```

```
type Token interface{}
```

```
type EndElement struct {  
    Name Name  
}
```

```
// getRootElementAttr extract root element attributes by  
// given XML decoder.
```

```
func getRootElementAttr(d *xml.Decoder) []xml.Attr {  
    tokenIdx := 0  
    for {  
        token, _ := d.Token()  
        if token == nil {  
            break  
        }  
        switch startElement := token.(type) {  
        case xml.StartElement:  
            tokenIdx++  
            if tokenIdx == 1 {  
                return startElement.Attr  
            }  
        }  
    }  
    return nil  
}
```

Ser/Deserialize Idempotence

```
<?xml version="1.0" encoding="utf-8"?>
<person
  xmlns="http://example.com/default"
  xmlns:m="http://example.com/main"
  xmlns:h="http://example.com/home"
  xmlns:w="http://example.com/work">
  <name>Tom</name>
  <m:email h:addr="HOME" w:addr="WORK" />
</person>
```

```
<?xml version="1.0" encoding="utf-8"?>
<person xmlns="http://example.com/default"
  xmlns:m="http://example.com/main"
  xmlns:h="http://example.com/home"
  xmlns:w="http://example.com/work" >
  <name>Tom</name>
  <email xmlns="http://example.com/main"
    xmlns:home="http://example.com/home"
    home:addr="HOME"
    xmlns:work="http://example.com/work"
    work:addr="WORK"></email>
</person>
```

```
decoder := xml.NewDecoder(strings.NewReader(data))
marshalXML := ""
for {
  token, _ := decoder.Token()
  if token == nil {
    break
  }
  switch element := token.(type) {
  case xml.StartElement:
    for _, attr := range element.Attr {
      if element.Name.Local == "person" {
        colon := ""
        if attr.Name.Space != "" {
          colon = ":"
        }
        marshalXML += fmt.Sprintf("%s%s%s=\"%s\" ",
          attr.Name.Space, colon,
          attr.Name.Local, attr.Value)
      }
    }
  }
}
fmt.Printf("<person %s>\n", marshalXML)
```

High Performance Processing

XML Components Data Model

```
<?xml version="1.0"?>
<note xmlns:m="http://example.com/main">
  <to>Tom</to>
  <from>Bob</from>
  <heading>Reminder</heading>
  <m:body>Don't forget me this weekend!</m:body>
</note>
```

shared.xsd

```
<?xml version="1.0"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="body" type="xs:string"/>
</xs:schema>
```

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:m="http://example.com/main">
<xsd:import namespace="http://example.com/main" schemaLocation="shared.xsd"/>
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="m:body" use="required"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

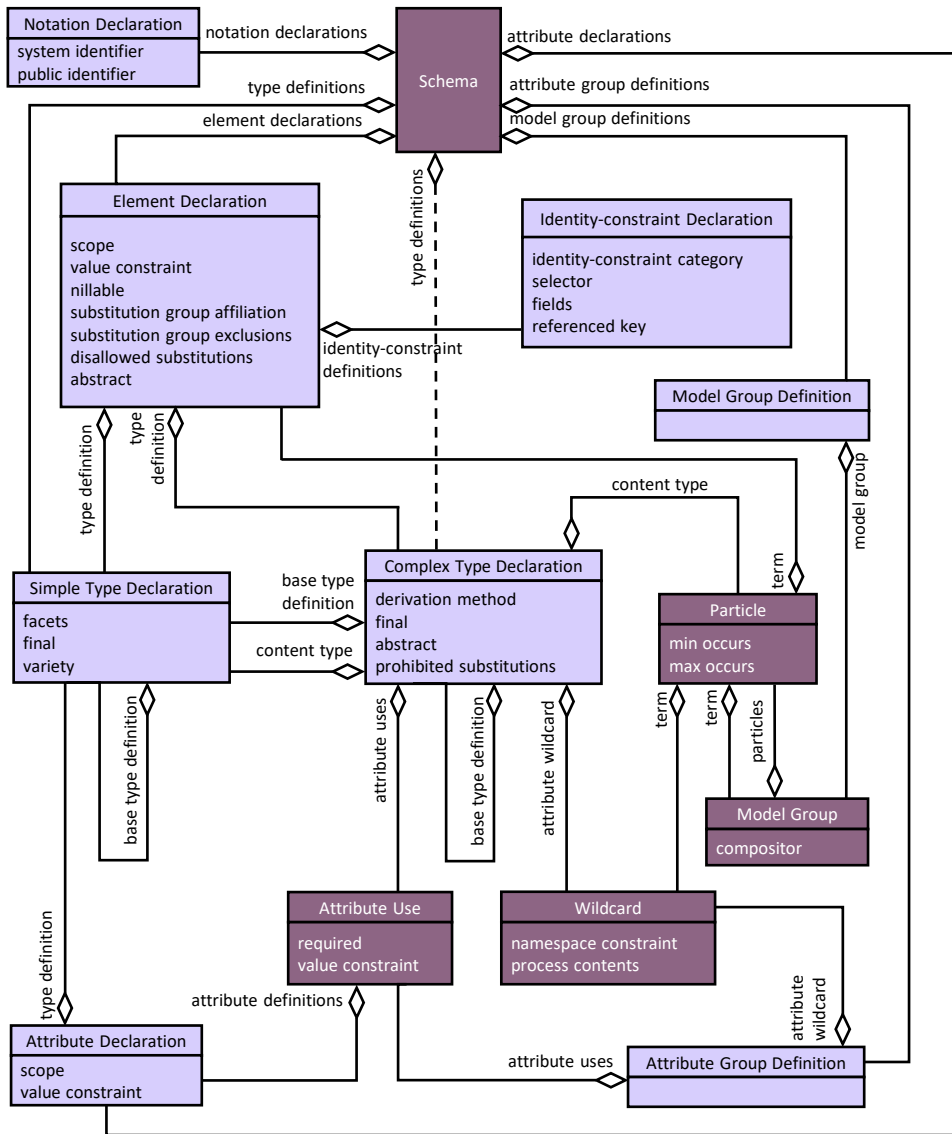
XML Components Data Model

```
<?xml version="1.0"?>  
<note xmlns:m="http://example.com/main">  
  <to>Tom</to>  
  <from>Bob</from>  
  <heading>Reminder</heading>  
  <m:body>Don't forget me this weekend!</m:body>  
</note>
```

```
type Note struct {  
  XMLName xml.Name `xml:"note"`  
  To      string  `xml:"to"`  
  From    string  `xml:"from"`  
  Heading string  `xml:"heading"`  
  Body    string  `xml:"http://example.com/main body"`  
}
```

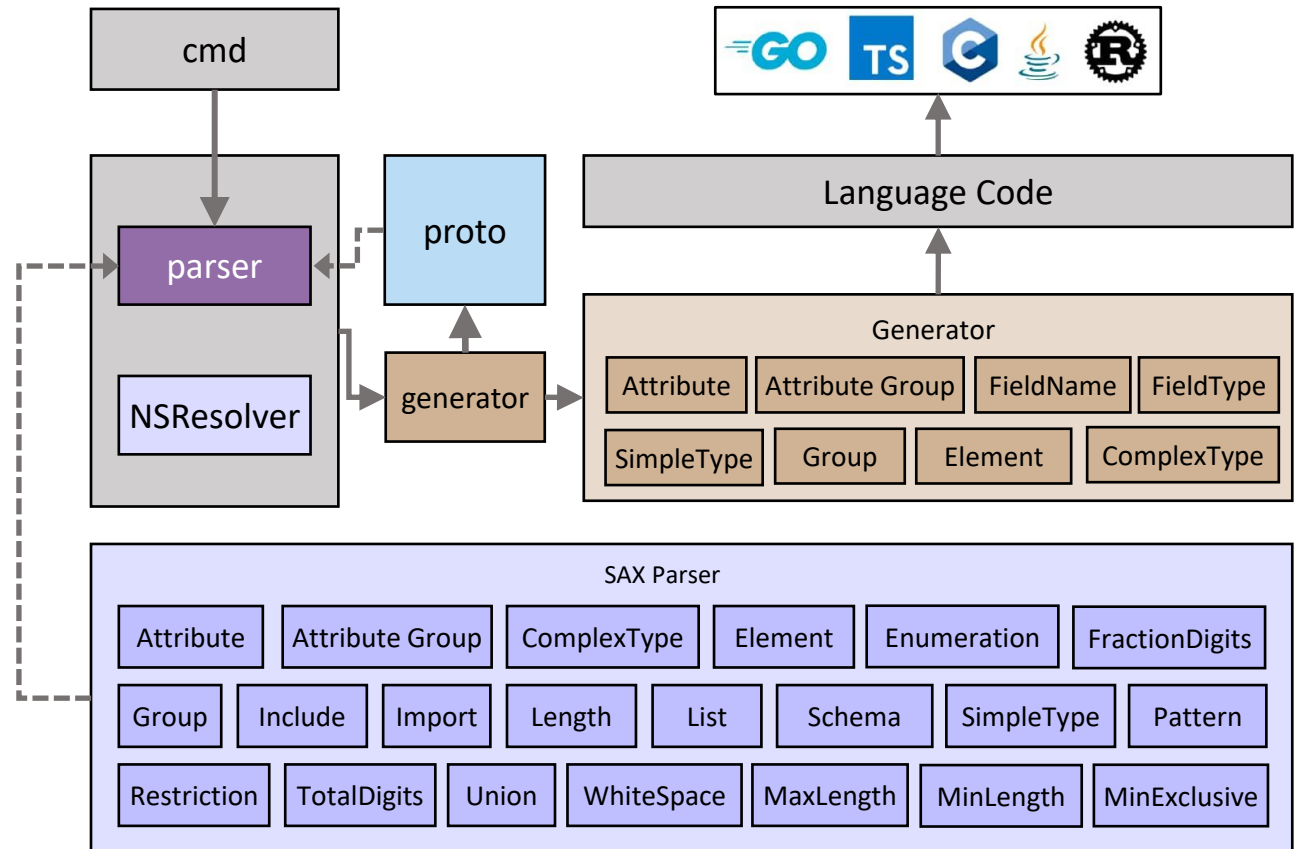


XML Components Data Model



is a named component and has two additional properties - name and target namespace
 is an un-named component

XSD: XML Schema Definition Process <https://github.com/xuri/xgen>

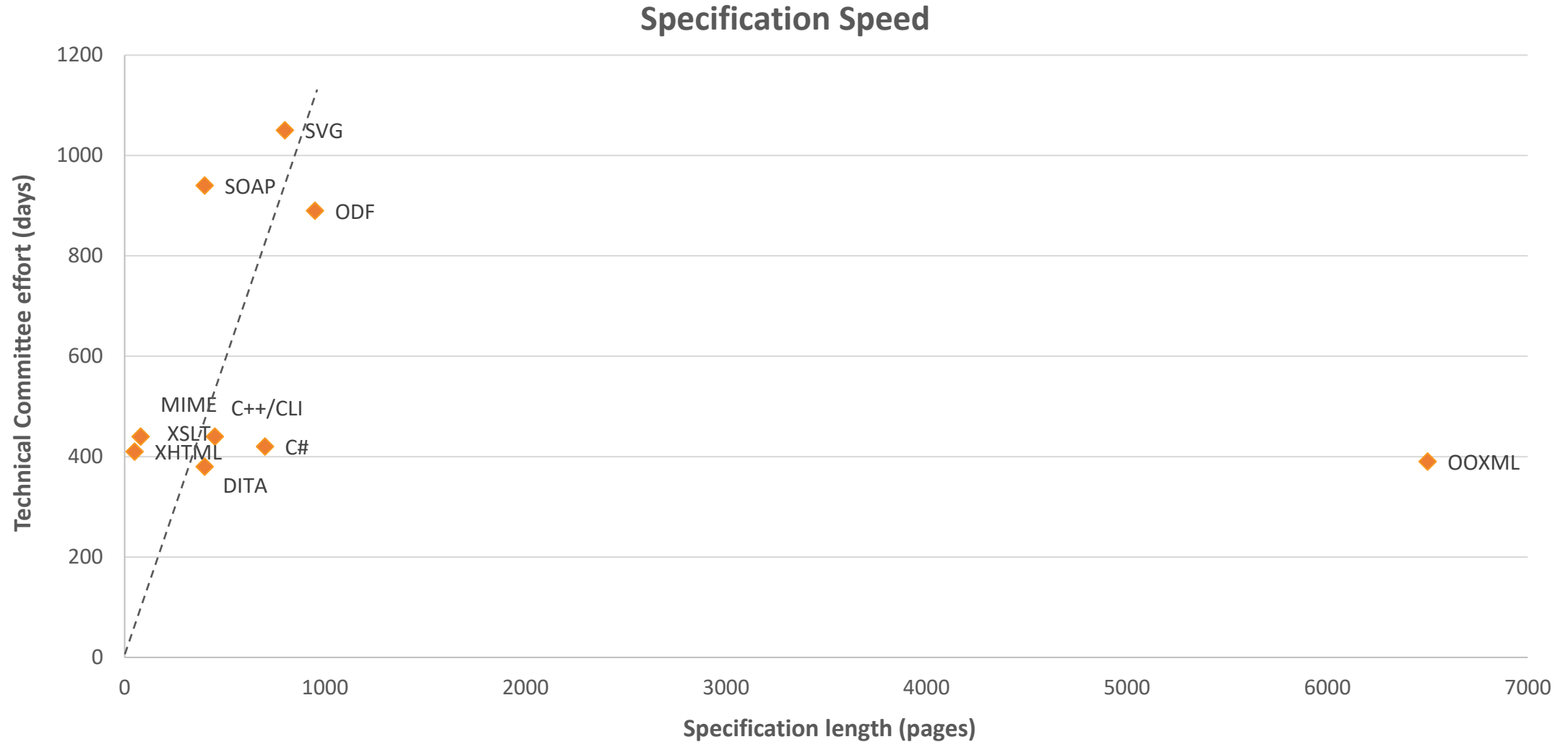


SAX or DOM

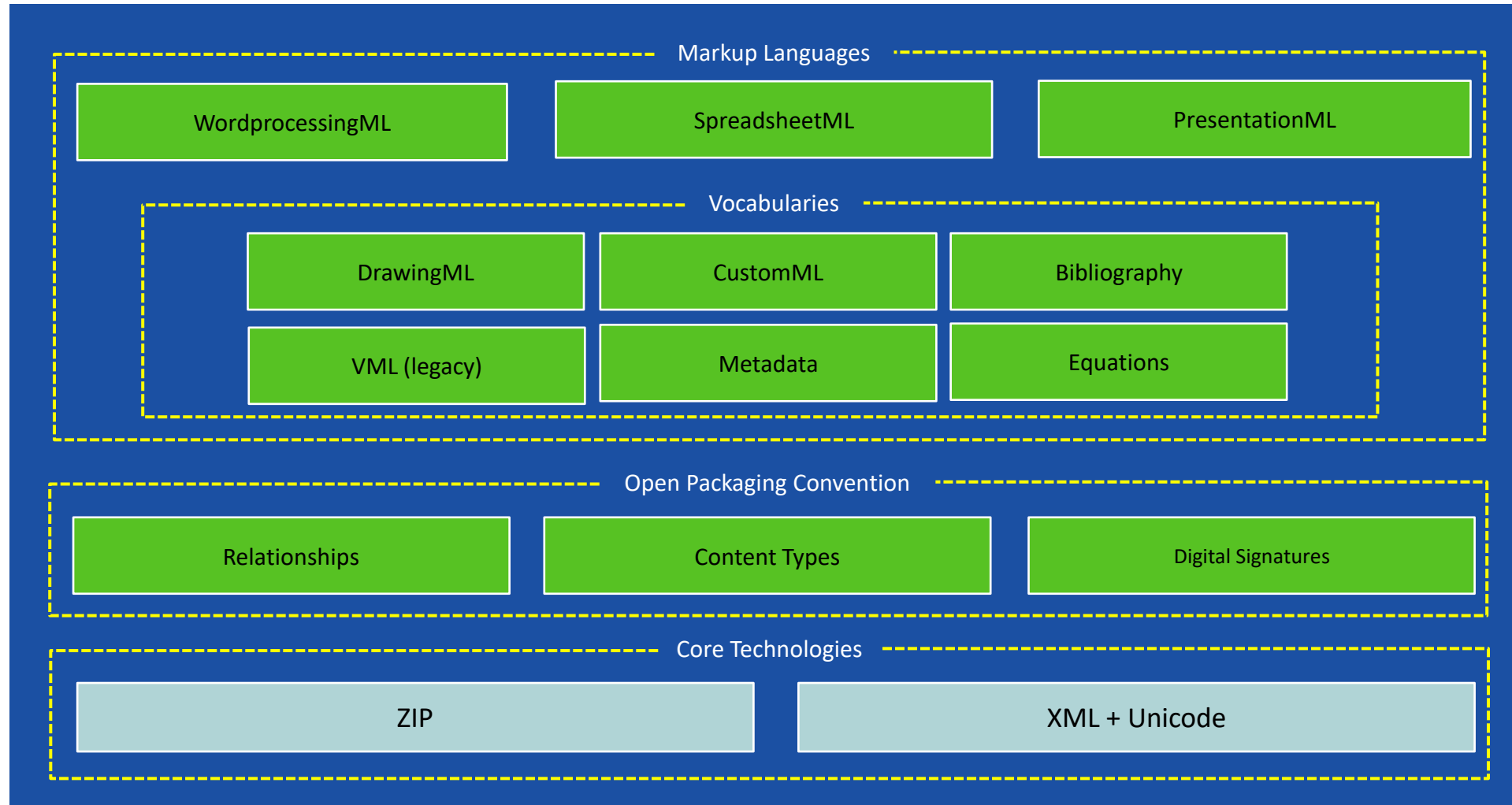
| SAX Parser | DOM Parser |
|---|---|
| Simple API for XML Parsing | Document Object Model |
| Event-based parser | Stays in a tree structure |
| Low memory usage | High memory usage |
| Best for the larger size of XML files | Best for the smaller sizes of files |
| Read-only | Insert or delete nodes |
| Backward navigation is not possible | Backward and forward search is possible |
| A small part of the XML file is only loaded in memory | It loads whole XML documents in memory |

OOXML Spreadsheets

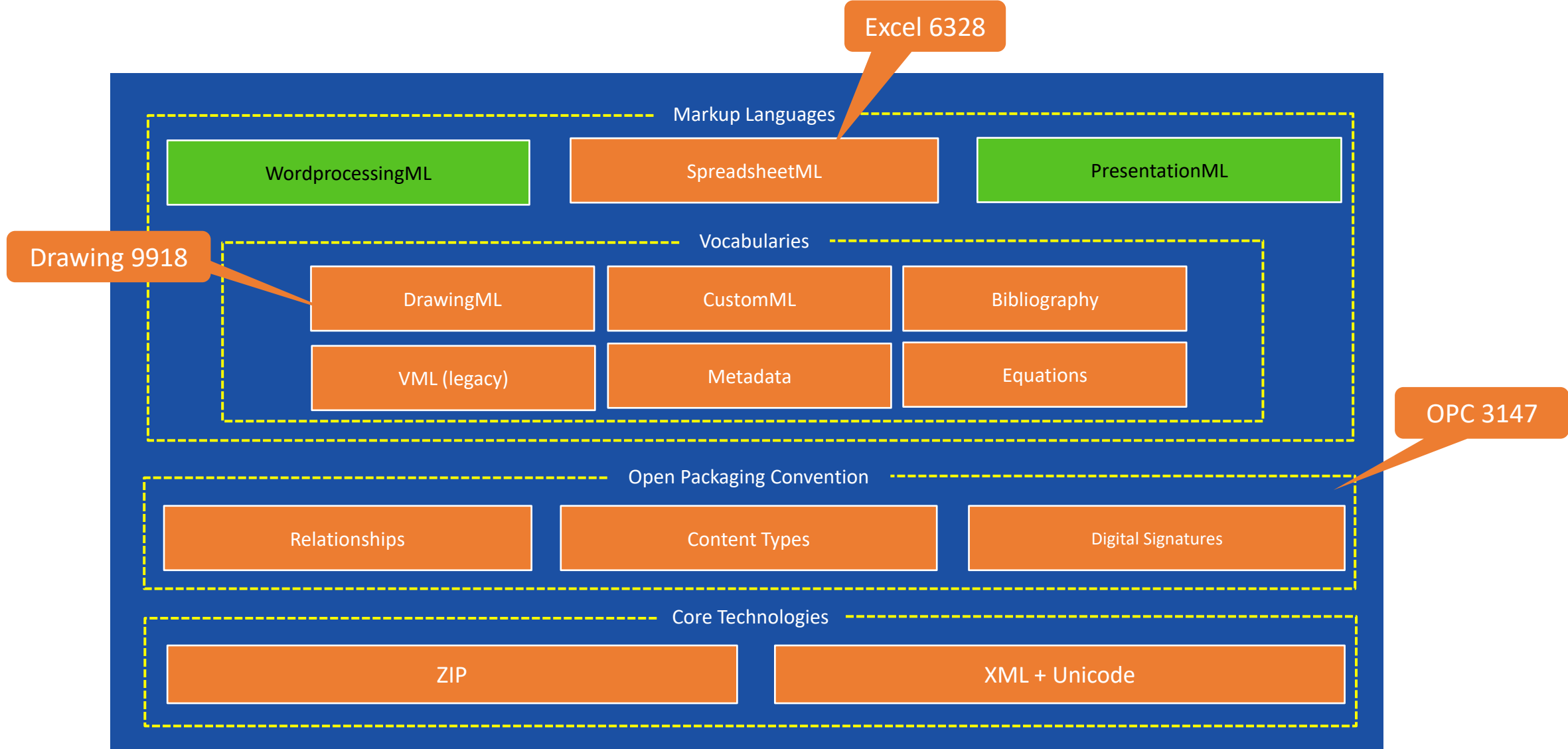
OOXML ISO/IEC 29500 ECMA-376 Specification



OOXML Specification



OOXML Specification



Typical XML of the Cell

Over 10589 Elements & Attributes

Start

| | | | | |
|--|---------|---|--------|----|
| | B | C | D | E |
| | | | Q1 | Q2 |
| | Revenue | | 412.52 | |

```

<c r="D3" s="7" t="s">
  <v>0</v>
</c>
<xf numFmtId="0" fontId="4" fillId="2" borderId="2" xfd="1" applyBorder="1" />
<font>
  <sz val="11"/>
  <color theme="0"/>
  <name val="Calibri"/>
  <scheme val="minor"/>
</font>
<fill>
  <patternFill patternType="solid">
    <fgColor theme="4"/>
    <bgColor theme="4"/>
  </patternFill>
</fill>
<border>
  <left style="thick">
    <color auto="1"/>
  </left>
  <right style="thick">
    <color auto="1"/>
  </right>
  <top style="thick">
    <color auto="1"/>
  </top>
  <bottom style="thick">
    <color auto="1"/>
  </bottom>
  <diagonal/>
</border>
<a:theme xmlns:a="http://schemas.openxmlformats.org
  <a:themeElements>
    <a:clrScheme name="Office">
      <a:dk1>
        <a:sysClr val="windowText" />
      </a:dk1>
      <a:lt1>
        <a:sysClr val="window" />
      </a:lt1>
      <a:dk2>
        <a:srgbClr val="1F497D"/>
      </a:dk2>
      <a:lt2>
        <a:srgbClr val="FAF3E8"/>
      </a:lt2>
      <a:accent1>
        <a:srgbClr val="5C93B4"/>
      </a:accent1>
    </a:clrScheme>
  </a:themeElements>
</a:theme>
<cellStyleXfs count="12">
  <xf numFmtId="0" fontId="0" fillId="0" borderId="0"/>
  <xf numFmtId="0" fontId="4" fillId="2" borderId="0" applyBorder="1" applyAlignment="1" applyProtection="1"/>
</cellStyleXfs>
<cellStyles count="2">
  <cellStyle name="Accent1" xfd="1" builtinId="29"/>
  
```

(build-in)



Charset Encoding

228 Labels

```
<?xml version='1.0' encoding='character encoding' standalone='yes|no'?>
```

| Name | Labels |
|---|--------|
| UTF-8 | 6 |
| Legacy single-byte encodings | 168 |
| Legacy multi-byte Chinese (simplified) encodings | 10 |
| Legacy multi-byte Chinese (traditional) encodings | 5 |
| Legacy multi-byte Japanese encodings | 13 |
| Legacy multi-byte Korean encodings | 10 |
| Legacy miscellaneous encodings | 16 |

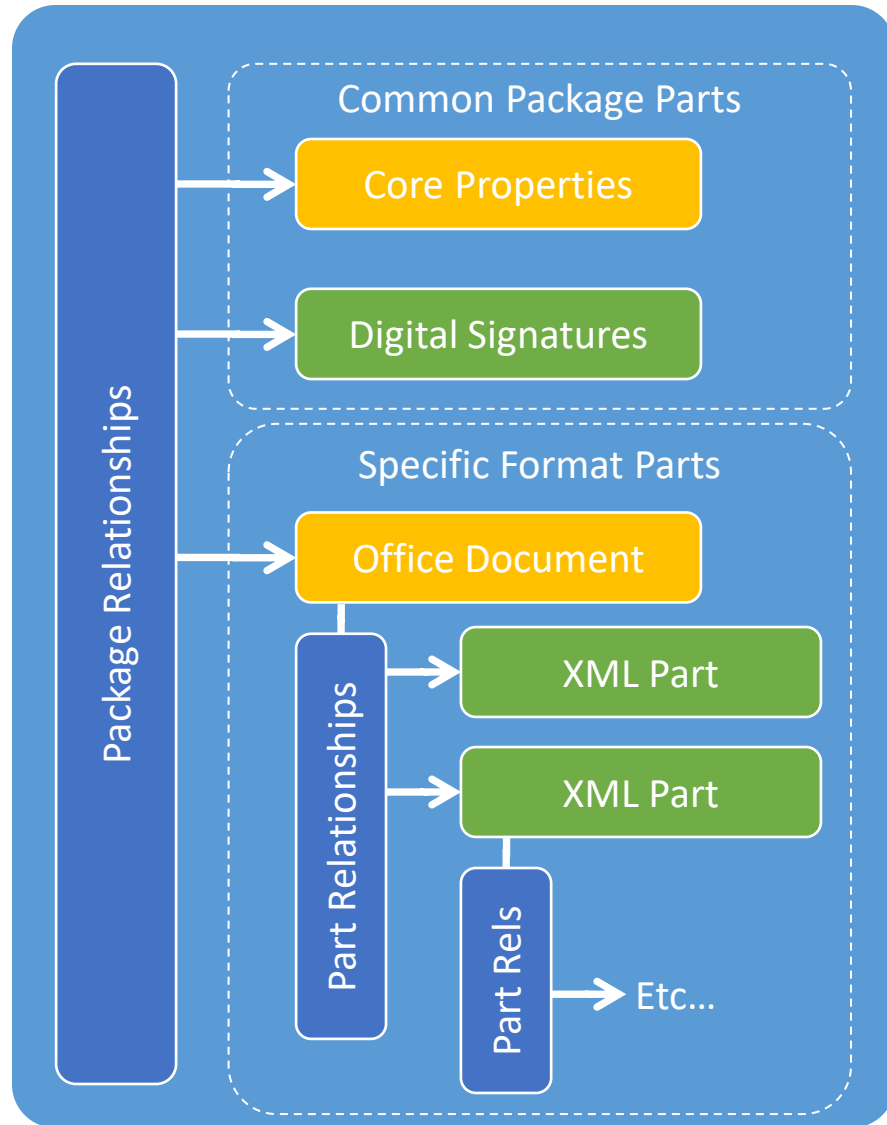
Ref: <https://encoding.spec.whatwg.org>

```
import (  
    "golang.org/x/net/html/charset"  
    "golang.org/x/text/encoding"  
    "golang.org/x/text/encoding/charmap"  
)  
decoder = xml.NewDecoder(strings.NewReader(data))  
decoder.CharsetReader = charset.NewReaderLabel
```

Custom Charset Reader

```
// CharsetReader Decoder from all codepages to UTF-8  
func CharsetReader(charset string, input io.Reader) io.Reader {  
    var enc encoding.Encoding  
    for i := range charmap.All {  
        item = charmap.All[i]  
        if strings.EqualFold(sm, nm) {  
            enc = item  
        }  
    }  
    return enc.NewDecoder().Reader(input)  
}
```

Streaming I/O



| | A | B | C |
|---|---|-----|---|
| 1 | | | |
| 2 | | 123 | |
| 3 | | | |
| 4 | | | |

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<worksheet
xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006">
  <dimension ref="B2"/>
  <sheetViews>
    <sheetView tabSelected="1" workbookViewId="0" />
  </sheetViews>
  <sheetFormatPr baseColWidth="10" defaultRowHeight="16" />
  <sheetData>
    <row r="2">
      <c r="B2">
        <v>123</v>
      </c>
    </row>
  </sheetData>
  <pageMargins left="0.7" right="0.7" />
</worksheet>
```

Set Row

```
<sheetData>
  <row r="2">
    <c r="B2">
      <v>123</v>
    </c>
  </row>
</sheetData>
```

```
type StreamWriter struct {
  File          *File
  Sheet         string
  SheetID      int
  worksheet     *xlsxWorksheet
  rawData      bufferedWriter
  mergeCellsCount int
  mergeCells   string
  tableParts   string
}
```

```
func writeCell(buf *bufferedWriter, c xlsxCell) {
  _, _ = buf.WriteString(`
```

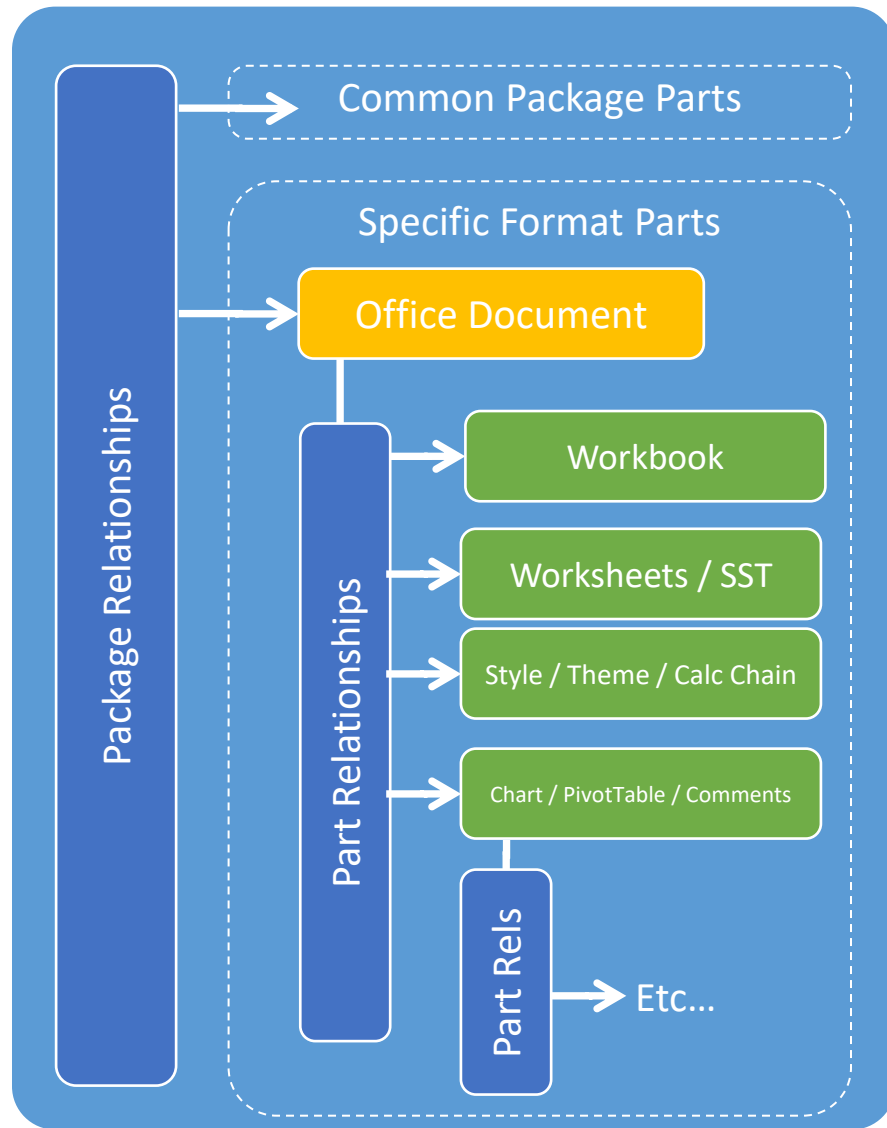
Flush

Generate XML Part

```
type StreamWriter struct {  
    File          *File  
    Sheet         string  
    SheetID      int  
    worksheet     *xlsxWorksheet  
    rawData      bufferedWriter  
    mergeCellsCount int  
    mergeCells   string  
    tableParts   string  
}
```

```
func (sw *StreamWriter) Flush() error {  
    _, _ = sw.rawData.WriteString(`</sheetData>`)  
    bulkAppendFields(&sw.rawData, sw.worksheet, 8, 15)  
    if sw.mergeCellsCount > 0 {  
        sw.mergeCells = fmt.Sprintf(`<mergeCells  
            count="%d">%s</mergeCells>`, sw.mergeCellsCount, sw.mergeCells)  
    }  
    _, _ = sw.rawData.WriteString(sw.mergeCells)  
    bulkAppendFields(&sw.rawData, sw.worksheet, 17, 38)  
    _, _ = sw.rawData.WriteString(sw.tableParts)  
    bulkAppendFields(&sw.rawData, sw.worksheet, 40, 40)  
    _, _ = sw.rawData.WriteString(`</worksheet>`)  
    if err := sw.rawData.Flush(); err != nil {  
        return err  
    }  
    // ...  
}
```


Save Spreadsheet



XML Part to ZIP

```
func (f *File) WriteToBuffer() (*bytes.Buffer, error) {
    buf := new(bytes.Buffer)
    zw := zip.NewWriter(buf)
    f.calcChainWriter()
    f.commentsWriter()
    f.contentTypeWriter()
    f.drawingsWriter()
    f.vmlDrawingWriter()
    f.workBookWriter()
    f.workSheetWriter()
    f.relsWriter()
    f.sharedStringsWriter()
    f.styleSheetWriter()

    for path, stream := range f.streams {
        // Save stream data
        stream.rawData.Close()
    }

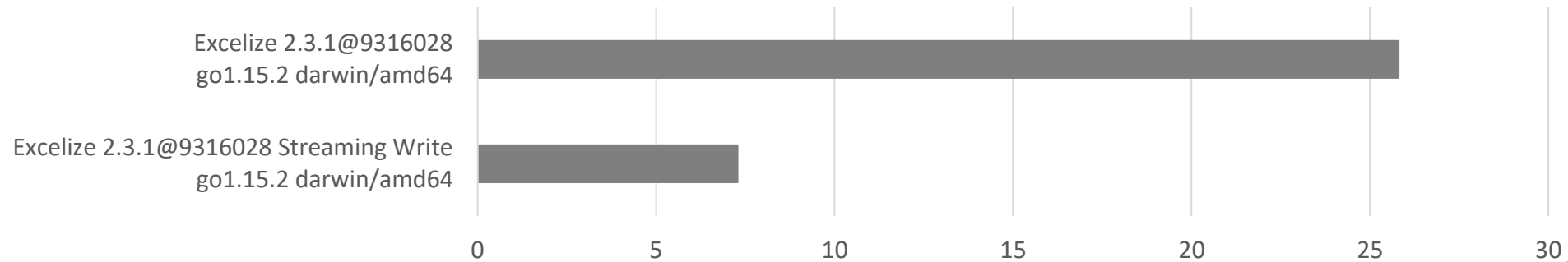
    for path, content := range f.XLSX {
        // Save preserve data
    }
}
```

Performance

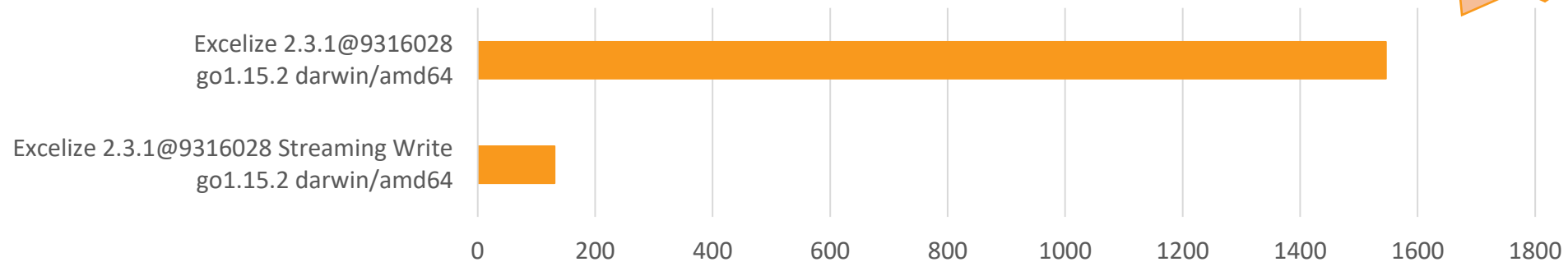
102400 Row x 50 Columns, 6 Chars / Cell

<https://github.com/xuri/excelize>

Time Cost (s)
Less is better



Memory Usage (MB)
Less is better



Processing XML and Spreadsheet in Go

Gopher China Conference
Beijing 2021 6/26 - 6/27

续日

