
# An iXML Grammar … is an iXML Sentence

Developing flexible iXML grammar applications,

by exploiting the sentence structure

John Lumley
*john@johnlumley.net*

# What is an iXML Grammar?

A set of grammatical production rules with
- terminal and non-terminal components,
- sequence, alternative, optional and repetition operators

describing a valid *sentence of characters*

A set of declarations attached to those rules that define for a valid sentence
- how to project the resulting *parse tree* as an XML tree of element, attribute and text nodes

There is an iXML grammar for a (commentable) textual representation of any iXML grammar, which:
- will generate the equivalent XML tree for the grammar
- where *all these components* are elements, attributes ~~*or text nodes*~~

A *textually defined* valid iXML Grammar:

1. **MUST** be a valid sentence against the iXML grammar
2. **MUST** satisfy a set of additional rules:
   - No multiply-defined non-terminals
   - No undefined but referenced non-terminals
   - etc. (See spec. ;-)
3. Preferably **SHOULD** satisfy additional desiderata:
   - No inherent ambiguity, e.g. doubly-accounted whitespace
   - No infinite recursion
   - etc.

Develop flexible iXML grammar applications,
by temporarily ignoring the *all but the first* requirement

# Assumptions

- You are reasonably familiar with iXML
  *and understand the XML form of a sample grammar*
- You are used to XML toolsets, in particular XSLT
  *and can write XSLT to write XSLT*
- You are the developer of a mixed toolchain based on XML, using iXML as a component
- You value flexibility in your design

# Two techniques

Use the XML representation of a grammar as data and generate other grammars and transforms

1. Add annotations to a grammar and use these to guide build-time ‘specialisation’
2. Analyse and transform the grammar as a semantic ‘graph’

See other approaches in my 2024 Balisage paper

https://www.balisage.net/Proceedings/vol29/html/Lumley01/BalisageVol29-Lumley01.html

# Example 1: Variant Grammars
## (Grammar Annotation)

```
<events>
   <event>Invisible XML Symposium,Netherlands@26/2/2026</event>
   <event>Balisage,USA@8/3/2026</event>
   <event>XMLPrague,Czech Republic@5/6/2026</event>
</events>
```

**If** the input contains a USA location:
parse dates as MonthDayYear
**otherwise**
parse dates as DayMonthYear

# Example 2: Partial Binding
## (Grammar analysis and transformation)

```
    order: customer,items.
    items: item++lf,lf?.
     item: s?,-"item:",~[","]+,quantity,s?.
@quantity: -",",digits.
 customer: person,lf,street....
     …
  -digits: ["0"-"9"]+.
     -lf: -#a | -#d,-#a.
```

We already have a valid parse sub-tree for **customer**

```
<customer>
  <person>
    <given>Steven</given>
    <surname>Pemberton</surname>
  </person>
  <street no="21" streetname="Sandridge Road"/>
  <city>St Albans</city>
  <postcode>AL1 4BY</postcode>
  <country>United Kingdom</country>
</customer>
```

```
item:ballpoint pen,3
item:notebook,1
```

```
order: customer, items.
 items: item++lf, lf?.
  item: S?, -"item:", ~[#a;","]+,
        quantity, S?.
@quantity: -",", digits.
 customer: ?????. …
  -digits: ["0"-"9"]+.
       -S: " "+.
      -lf: -#a | -#d, -#a.
```

```
<order>
    <customer>
      <person>
        <given>Steven</given>
        <surname>Pemberton</surname>
      </person>
      <street no="21" streetname="Sandridge Road"/>
      <city>St Albans</city>
      <postcode>AL1 4BY</postcode>
      <country>United Kingdom</country>
    </customer>
    <items>
      <item quantity="3">ballpoint pen</item>
      <item quantity="1">notebook</item>
    </items>
</order>
```

# Example 1: Variant Grammars

```
<events>
   <event>Invisible XML Symposium,Netherlands@26/2/2026</event>
   <event>Balisage,USA@8/3/2026</event>
   <event>XMLPrague,Czech Republic@5/6/2026</event>
</events>
```

```
event: description,-",",location,-"@", date.
description: ~[","]+.
location: ~["@"]+.

date: day,sep,month,sep,year.
date: month,sep,day,sep,year.
-sep: -"/".
@day: ["012"]?,[Nd]; "3", ["01"].
@month: [Nd]; "1", ["12"].
@year: [Nd],[Nd],[Nd],[Nd].
```

```
{variant;USA;contains(.,"USA")}
event: description,-",",location,-"@", date.
description: ~[","]+. location: ~["@"]+.

date{#default}: day,sep,month,sep,year.
date{#USA}: month,sep,day,sep,year.

-sep: -"/".
@day: ["012"]?,[Nd]; "3", ["01"].
@month: [Nd]; "1", ["12"].
@year: [Nd],[Nd],[Nd],[Nd].
```

```
<ixml>
 <comment>variant;USA;contains(.,"USA")</comment>
 <rule name="event">
  <alt>
   <nonterminal name="description"/>
   <literal tmark="-" string=","/>
   <nonterminal name="location"/>
   <literal tmark="-" string="@"/>
   <nonterminal name="date"/>
  </alt>
 </rule>…
```

```
<rule name="date">
 <comment>#default</comment>
 <alt>
  <nonterminal name="day"/>
  <nonterminal name="sep"/>
  <nonterminal name="month"/>
  <nonterminal name="sep"/>
  <nonterminal name="year"/>
 </alt>
</rule>
```

```
<rule name="date">
 <comment>#USA</comment>
 <alt>
  <nonterminal name="month"/>
  <nonterminal name="sep"/>
  <nonterminal name="day"/>
  <nonterminal name="sep"/>
  <nonterminal name="year"/>
 </alt>
</rule>
```

```
ixml/comment[starts-with(., 'variant;')]

map{
  'USA':     ¨contains(.,'USA')¨,
  'default': ¨true()¨
}
```

```
<xsl:template match="rule[starts-with(comment[1], '#')]">
    <xsl:param name="variant" as="xs:string"/>
    <xsl:if test="substring-after(comment[1], '#') eq $variant">
       <xsl:next-match/>
    </xsl:if>
</xsl:template>
```

# Build-time processing with XSLT

**`ixml` Grammar** (iXML text)

```
{version 2022-05-17}
        ixml: s, rule++RS, s.

          -s: (whitespace; comment)*.
         -RS: (whitespace; comment)+.
  -whitespace: -[Zs]; tab; lf; cr.
….
```

IXML Processor

**`event` Grammar** (iXML text)

***{variant;USA;contains(.,"USA")}***
event: description,-",",location,-"@", date.
description: ~[","]+. location: ~["@"]+.

date***{#default}***: day,sep,month,sep,year.
date***{#USA}***: month,sep,day,sep,year.

-sep: -"/".
@day: ["012"]?,[Nd]; "3", ["01"].
@month: [Nd]; "1", ["12"].
@year: [Nd],[Nd],[Nd],[Nd].

**`event` Grammar** <XML/>

```
<ixml>
 <comment>variant;USA;
      contains(.,"USA")</comment>
 <rule name="date">
   <comment>#default</comment>
   <alt>
     <nonterminal name="day"/>
     <nonterminal name="sep"/>
     <nonterminal name="month"/>
     <nonterminal name="sep"/>
     nonterminal name="year"/>
   </alt>
 </rule>
 <rule name="date">
   <comment>#USA</comment>
   <alt>
     <nonterminal name="month"/>
     <nonterminal name="sep"/>
     <nonterminal name="day"/>
     <nonterminal name="sep"/>
     <nonterminal name="year"/>
   </alt>
  </rule>
```

**`variant.xsl`**

```
<xsl:template match="rule">
   .....
</xsl:template>
```

**`event.default.ixml`**

event: description,-",",location,-"@", date.
description: ~[","]+. location: ~["@"]+.
date***{#default}***: day,sep,month,sep,year.
-sep: -"
@day:
@mont
@year:

**`event.USA.ixml`**

event: description,-",",location,-"@", date.
description: ~[","]+. location: ~["@"]+.
date***{#USA}***: month,sep,day,sep,year.
-sep: -"/".
@day: ["012"]?,[Nd]; "3", ["01"].
@month: [Nd]; "1", ["12"].
@year: [Nd],[Nd],[Nd],[Nd].

**`event.xsl`**

```
<xsl:template match="event">
    <xsl:variable name="parserSource" as="xs:string">
      <xsl:choose>
        <xsl:when test="contains(.,&#34;USA&#34;)"
            select="'event.USA.ixml'"/>
        <xsl:when test="true()" select="'event.default.ixml'"/>
      </xsl:choose>
    </xsl:variable>
    <xsl:variable name="parser"
          select="cs:load-grammar($parserSource)"/>
    <xsl:sequence select="$parser(string(.))"/>
</xsl:template>
```

*event.USA.ixml*

```
      event: description,-",",location,-"@", date.
description: ~[","]+.
   location: ~["@"]+.
       date: month,sep,day,sep,year.
       -sep: -"/".
       @day: ["012"]?,[Nd]; "3", ["01"].
     @month: [Nd]; "1", ["012"].
      @year: [Nd],[Nd],[Nd],[Nd].
```

*event.default.ixml*

```
...
...
...
date: day,sep,month,sep,year.
...
```

```
<xsl:template match="event">
    <xsl:variable name="parserSource" as="xs:string">
      <xsl:choose>
        <xsl:when test="contains(.,&#34;USA&#34;)" select="'event.USA.ixml'"/>
        <xsl:when test="true()" select="'event.default.ixml'"/>
      </xsl:choose>
    </xsl:variable>
    <xsl:variable name="parser" select="cs:load-grammar($parserSource)"/>
    <xsl:sequence select="$parser(string(.))"/>
</xsl:template>
```

```
<events>
  <event>Invisible XML Symposium,Netherlands@26/2/2026</event>
  <event>Balisage,USA@8/3/2026</event>
  <event>XMLPrague,Czech Republic@5/6/2026</event>
</events>
```

```
<events>
  <event>
    <description>Invisible XML Symposium</description>
    <location>Netherlands</location>
    <date day="26" month="2" year="2026"/>
  </event>
  <event>
    <description>Balisage</description>
    <location>USA</location>
    <date month="8" day="3" year="2026"/>
  </event>
  <event>
    <description>XMLPrague</description>
    <location>Czech Republic</location>
    <date day="5" month="6" year="2026"/>
  </event>
</events>
```

jωiXML

# Example 2: Partial Binding

```
      order: customer,items.
      items: item++lf,lf?.
       item: s?,-"item:",~[#a;","]+,quantity,s?.
  @quantity: -",",digits.
   customer: person,lf,street,lf,postcode,city,lf,country,lf ;
             person,lf,street,lf,city,postcode,lf,country,lf.
     person: (title,s?)?,(initials;given,S),surname,s?.
      title: "Mr."; "Mrs."; "Dr."; "Ms.".
   initials: initial+.
    initial: LETTER,".",s?.
    surname: name.
      given: name.
      -name: LETTER,letters.
     street: no, s,streetname, s?; streetname,s,no,s?.
@streetname: name; name,S,name.
       city: name,s?; name,S,name,s?.
    country: name,s?; name,S,name,s?.
   postcode: digits,S,LETTER,LETTER,s? ;
             LETTER,LETTER,digits,S,digit,LETTER,LETTER,s?.
        @no: digits.
    -LETTER: ["A"-"Z"].
   -letters: ["a"-"z"]*.
     -digit: ["0"-"9"].
    -digits: ["0"-"9"]+.
         -s: -" "+.
         -S: " "*.
        -lf: -#a | -#d,-#a.
```

Steven Pemberton
21 Sandridge Road
St Albans AL1 4BY
United Kingdom
item:ballpoint pen,3
item:notebook,1

```
<customer>
  <person>
    <given>Steven</given>
    <surname>Pemberton</surname>
  </person>
  <street no="21" streetname="Sandridge Road"/>
  <city>St Albans</city>
  <postcode>AL1 4BY</postcode>
  <country>United Kingdom</country>
</customer>
```

# Example 2: Partial Binding

```
    order: customer,items.
    items: item++lf,lf?.
     item: s?,-"item:",~[","]+,quantity,s?.
@quantity: -",",digits.
 customer: person,lf,street....
      …
  -digits: ["0"-"9"]+.
      -lf: -#a | -#d,-#a.
```

We already have a valid parse sub-tree for **`customer`**

1. Can we produce a reduced grammar for *all the rest* of **`order`**
2. Can we interpolate the binding for **`customer`** into the final result tree?
3. Can we extract the sub-grammar for **`customer`**?

```
<customer>
  <person>
    <given>Steven</given>
    <surname>Pemberton</surname>
  </person>
  <street no="21" streetname="Sandridge Road"/>
  <city>St Albans</city>
  <postcode>AL1 4BY</postcode>
  <country>United Kingdom</country>
</customer>
```

```
   order: customer, items.
    items: item++lf, lf?.
     item: S?, -"item:", ~[#a;","]+,
           quantity, S?.
@quantity: -",", digits.
 customer: .
  -digits: ["0"-"9"]+.
       -S: " "+.
      -lf: -#a | -#d, -#a.
```

```
      order: customer,items.
      items: item++lf,lf?.
       item: s?,-"item:",~[#a;","]+,quantity,s?.
  @quantity: -",",digits.
   customer: person,lf,street,lf,postcode,city,lf,country,lf ;
             person,lf,street,lf,city,postcode,lf,country,lf.
     person: (title,s?)?,(initials;given,S),surname,s?.
      title: "Mr."; "Mrs."; "Dr."; "Ms.".
   initials: initial+.
    initial: LETTER,".",s?.
    surname: name.
      given: name.
      -name: LETTER,letters.
     street: no, s,streetname, s?; streetname,s,no,s?.
@streetname: name; name,S,name.
       city: name,s?; name,S,name,s?.
    country: name,s?; name,S,name,s?.
   postcode: digits,S,LETTER,LETTER,s? ;
             LETTER,LETTER,digits,S,digit,LETTER,LETTER,s?.
        @no: digits.
    -LETTER: ["A"-"Z"].
   -letters: ["a"-"z"]*.
     -digit: ["0"-"9"].
    -digits: ["0"-"9"]+.
         -s: -" "+.
         -S: " "*.
        -lf: -#d?, -#a.
```

```
item:ballpoint pen,3
item:notebook,1
```

```
    order: customer, items.
    items: item++lf, lf?.
     item: s?, -"item:", ~[#a;","]+, quantity, s?.
@quantity: -",", digits.
 customer: .
  -digits: ["0"-"9"]+.
       -s: -" "+.
      -lf: -#d?, -#a.
```

```
<order>
    <customer/>
    <items>
      <item quantity="3">ballpoint pen</item>
      <item quantity="1">notebook</item>
    </items>
</order>
```

```
    order: customer,items.
    items: item++lf,lf?.
     item: s?,-"item:",~[#a;","]+,quantity,s?.
@quantity: -",",digits.
 customer: .person,lf,street,lf,postcode,city,lf,country,lf ;
           person,lf,street,lf,city,postcode,lf,country,lf.
   person: (title,s?)?,(initials;given,S),surname,s?.
    title: "Mr."; "Mrs."; "Dr."; "Ms.".
 initials: initial+.
  initial: LETTER,".",s?.
  surname: name.
    given: name.
    -name: LETTER,letters.
   street: no, s,streetname, s?; streetname,s,no,s?.
@streetname: name; name,S,name.
     city: name,s?; name,S,name,s?.
  country: name,s?; name,S,name,s?.
 postcode: digits,S,LETTER,LETTER,s? ;
           LETTER,LETTER,digits,S,digit,LETTER,LETTER,s?.
      @no: digits.
  -LETTER: ["A"-"Z"].
 -letters: ["a"-"z"]*.
   -digit: ["0"-"9"].
  -digits: ["0"-"9"]+.
      -s: -" "+.
      -S: " "*.
     -lf: -#d?,-#a.
```

Using the XML form of the grammar:

1. Replace the bound rule(s) with a stub
   **`customer: .`**
2. Find the set of all rules **`order`** now depends upon, by transitive closure.
3. Filter out all rules which are not within that set.
4. Convert to text form if required.

```
    order: customer, items.
    items: item++lf, lf?.
     item: s?, -"item:", ~[#a;","]+, quantity, s?.
@quantity: -",", digits.
 customer: .
  -digits: ["0"-"9"]+.
       -s: -" "+.
      -lf: -#d?, -#a.
```

# Partial Binding –
## i) post-process substitution

```
<order>
    <customer/>
    <items>
      <item quantity="3">ballpoint pen</item>
      <item quantity="1">notebook</item>
    </items>
</order>
```

```
$bindings := map{
 'customer': doc('steven.xml'),
 ...
}
```

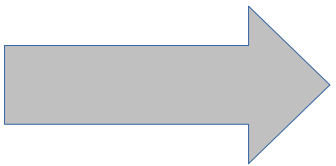
```
<xsl:template match="customer|other-bound-components">
   <xsl:param name="bindings"
              as="map(xs:string,document-node())"
              tunnel="true"/>
   <xsl:sequence select="$bindings(local-name())"/>
</xsl:template>
```

```
<order>
      <customer>
        <person>
          <given>Steven</given>
          <surname>Pemberton</surname>
        </person>
        <street no="21" streetname="Sandridge Road"/>
        <city>St Albans</city>
        <postcode>AL1 4BY</postcode>
        <country>United Kingdom</country>
      </customer>
      <items>
        <item quantity="3">ballpoint pen</item>
        <item quantity="1">notebook</item>
      </items>
</order>
```

# Partial Binding –
## ii) specialise the grammar

Generate a set of rules that will produce the binding XML tree for no input text by using insertion of text and attributes

```
<customer>
  <person>
    <given>Steven</given>
    <surname>Pemberton</surname>
  </person>
  <street no="21" streetname="Sandridge Road"/>
  <city>St Albans</city>
  <postcode>AL1 4BY</postcode>
  <country>United Kingdom</country>
</customer>
```



```
customer: person, street, city, postcode, country.
  person: given, surname.
   given: +"Steven".
 surname: +"Pemberton".
  street: no, streetname.
     @no: +"21".
@streetname: +"Sandridge Road".
    city: +"St Albans".
postcode: +"AL1 4BY".
 country: +"United Kingdom".
```

Add these rules to the partially bound grammar

If no element/attribute name is repeated, conversion is straightforward.

```
<xsl:for-each select=".//*, .//*/@*">
   <rule name="{name()}">
     <xsl:if test=". instance of attribute()">
       <xsl:attribute name="mark" select="'@'"/>
     </xsl:if>
     <alt>
       <xsl:apply-templates select="@*,node()"/>
     </alt>
   </rule>
</xsl:for-each>
```

```
<xsl:template match="*|@*">
    <nonterminal name="{name()}"/>
</xsl:template>
<xsl:template match="text()">
    <insertion string="{.}"/>
</xsl:template>
```

```
      order: customer, items.
      items: item++lf, lf?.
       item: s?, -"item:", ~[#a;","]+, quantity, s?.
  @quantity: -",", digits.
   customer: person, street, city, postcode, country.
     person: given, surname.
      given: +"Steven".
    surname: +"Pemberton".
     street: no, streetname.
       city: +"St Albans".
   postcode: +"AL1 4BY".
    country: +"United Kingdom".
        @no: +"21".
@streetname: +"Sandridge Road".
    -digits: ["0"-"9"]+.
         -s: -" "+.
        -lf: -#d?, -#a.
```

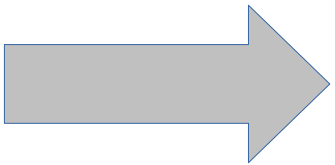
```
item:ballpoint pen,3
item:notebook,1
```

```
<order>
      <customer>
        <person>
          <given>Steven</given>
          <surname>Pemberton</surname>
        </person>
        <street no="21" streetname="Sandridge Road"/>
        <city>St Albans</city>
        <postcode>AL1 4BY</postcode>
        <country>United Kingdom</country>
      </customer>
      <items>
        <item quantity="3">ballpoint pen</item>
        <item quantity="1">notebook</item>
      </items>
</order>
```

# Partial Binding –
## ii) specialise the grammar

If element/attribute names are repeated, then we need to use renaming.

```
<customer>
  <person>
    <initials>
      <initial>J.</initial>
      <initial>W.</initial>
    </initials>
    <surname>Lumley</surname>
  </person>
  <street no="17" streetname="The Laurels"/>
  <city>Bristol</city>
  <postcode>BS12 3YZ</postcode>
  <country>United Kingdom</country>
</customer>
```



```
        customer: person, street, city, postcode, country.
          person: initials, surname.
        initials: initial0, initial1.
initial0>initial: +"J.".
initial1>initial: +"W.".
         surname: +"Lumley".
          street: no, streetname.
            city: +"Bristol".
        postcode: +"BS12 3YZ".
         country: +"United Kingdom".
             @no: +"17".
    @streetname: +"The Laurels".
```

# Conclusion

- An **iXML** sentence that conforms (just) to the iXML grammar can be the basis of a flexible suite of grammar applications
- The XML form of such sentences can be used with XSLT to extend your iXML application

Many thanks to my colleague iXML implementers for hours of valuable discussion, ideas and encouragement