Blasien:
programmer-friendly XML in C++11

Jos van den Oever

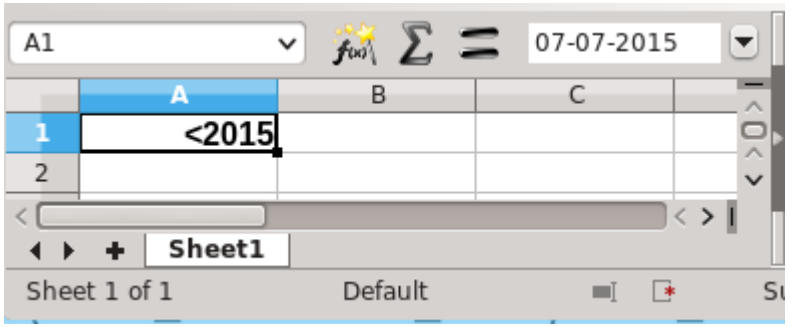
Blasien:
programmer-friendly XML in C++11

Prevention is better than cure

Jos van den Oever

ODF is XML

```
<number:date-style style:name="minimum_year">
    <style:text-properties fo:font-weight="bold"/>
    <number:text>&lt;</number:text>
    <number:year/>
</number:date-style>
```



ODF Specification

16.27.10 <number:date-style>

The <number:date-style> element specifies a style for date values.

This element can contain *one* instance of each of the following elements: <number:day>, <number:month>, <number:year>, <number:era>, <number:day-of-week>, <number:week-of-year>, <number:quarter>, <number:hours>, <number:minutes>, <number:seconds>, and <number:am-pm>.

The <number:date-style> element is usable within the following elements: <office:automatic-styles> 3.15.3 and <office:styles> 3.15.2.

The <number:date-style> element has the following attributes: number:automatic-order 19.340, number:country 19.342, number:format-source 19.347, number:language 19.349, number:rfc-language-tag 19.356, number:script 19.357, number:title 19.360, number:transliteration-country 19.361, number:transliteration-format 19.362, number:transliteration-language 19.363, number:transliteration-style 19.364, style:display-name 19.472, style:name 19.498.2 and style:volatile 19.517.

The <number:date-style> element has the following child elements: <number:am-pm> 16.27.22, <number:day> 16.27.11, <number:day-of-week> 16.27.15, <number:era> 16.27.14, <number:hours> 16.27.19, <number:minutes> 16.27.20, <number:month> 16.27.12, <number:quarter> 16.27.17, <number:seconds> 16.27.21, <number:text> 16.27.26, <number:week-of-year> 16.27.16, <number:year> 16.27.13, <style:map> 16.3 and <style:text-properties> 16.27.28.

Relax NG

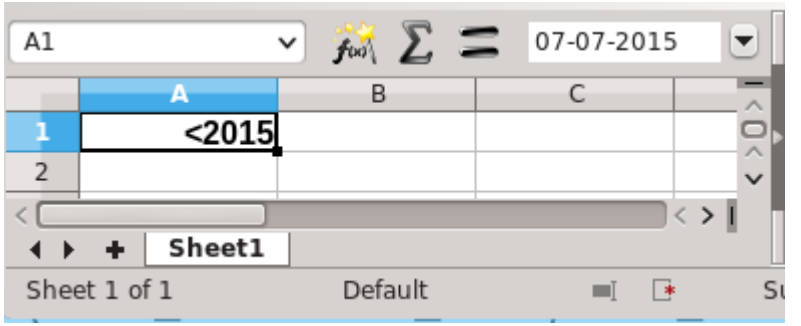
```
<rng:define name="number-date-style">
  <rng:element name="number:date-style">
    <rng:ref name="common-data-style-attlist"/>
    <rng:ref name="common-auto-reorder-attlist"/>
    <rng:ref name="common-format-source-attlist"/>
    <rng:optional>
      <rng:ref name="style-text-properties"/>
    </rng:optional>
    <rng:optional>
      <rng:ref name="number-text"/>
    </rng:optional>
    <rng:oneOrMore>
      <rng:ref name="any-date"/>
      <rng:optional>
        <rng:ref name="number-text"/>
      </rng:optional>
    </rng:oneOrMore>
    <rng:zeroOrMore>
      <rng:ref name="style-map"/>
    </rng:zeroOrMore>
  </rng:element>
</rng:define>
```

Relax NG

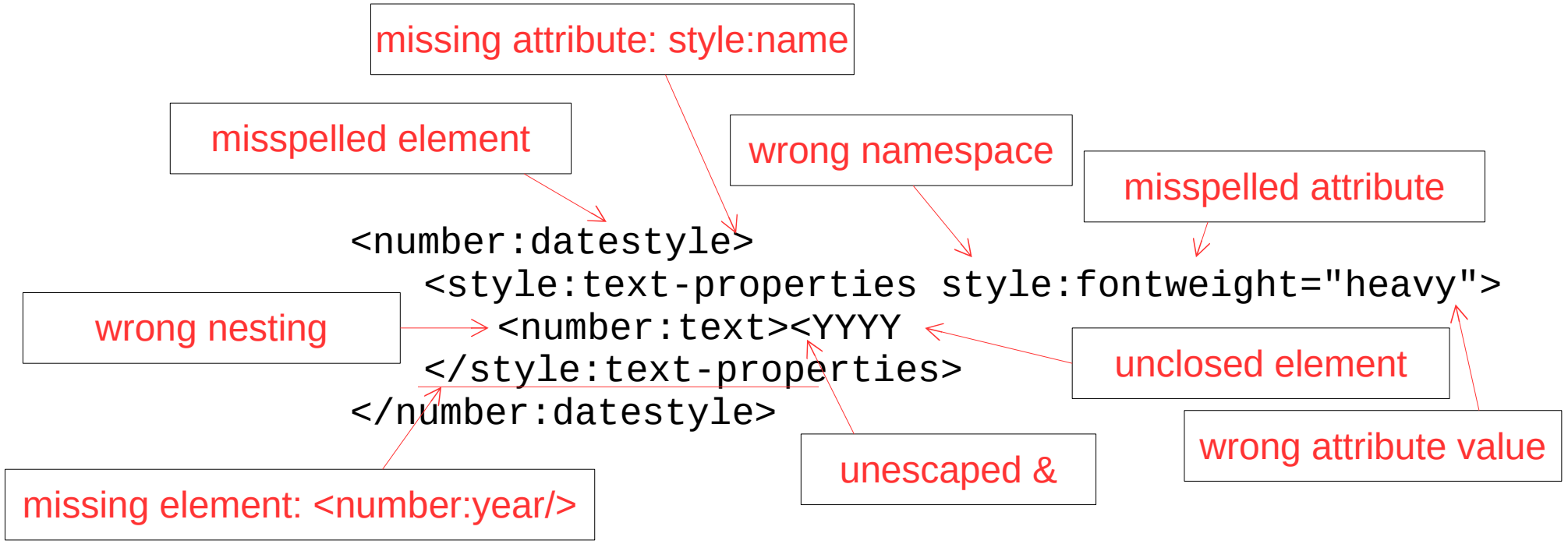
- element names `<office:text/>`
- attribute names `style:name="..."`
- element nesting `<html><head>`
- element order `</head><body>`
- element data type `<dc:date>2015-08-...`
- attribute data type `show="true"`

Naive example: 9 errors in 7 lines

```
out.writeStartElement("number:datestyle");
out.writeStartElement("style:text-properties");
out.writeAttribute("style:fontweight", "heavy");
out.writeStartElement("number:text");
out.write("<YYYY");
out.writeEndElement();
out.writeEndElement();
```



Naive example: 9 errors in 7 lines

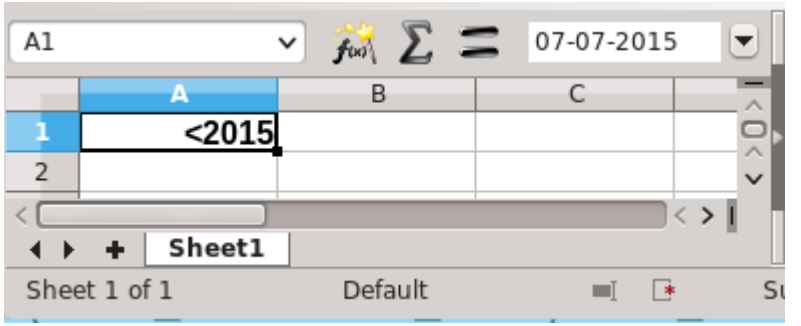


LibreOffice FastSerializer

6 errors in 7 lines

```
GetExport().StartElement( XML_NAMESPACE_TEXT, XML_DATE_STYLE, false);
GetExport().StartElement( XML_NAMESPACE_TEXT, XML_TEXT_PROPERTIES, false);
GetExport().AddAttribute( XML_NAMESPACE_STYLE, XML_FONT_WEIGHT, sHeavy);
GetExport().StartElement( XML_NAMESPACE_NUMBER, XML_TEXT, false);
GetExport().Characters("<YYYY");
GetExport().EndElement( XML_NAMESPACE_TEXT, XML_TEXT_PROPERTIES, false);
GetExport().EndElement( XML_NAMESPACE_TEXT, XML_DATE_STYLE, false);
```

similar to LO file *txtparae.cpp*



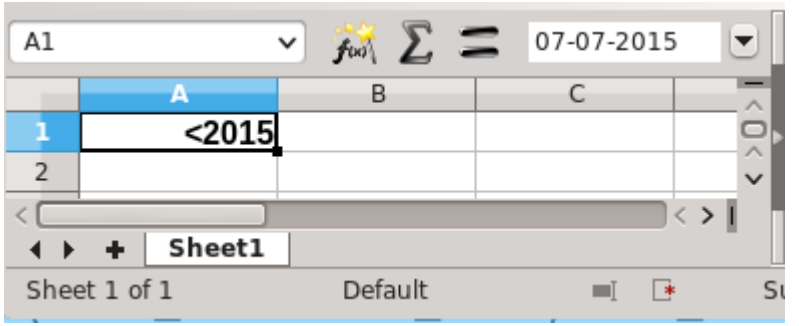
Blasien

ODF

```
<number:date-style style:name="maximum_year">
    <style:text-properties fo:font-weight="bold"/>
    <number:text>&lt;</number:text>
    <number:year/>
</number:date-style>
```

C++11

```
XmlWriter<style::StyleType>(stream)
<number::date_style( text::style_name="maximum_year" )
    <style::text_properties( fo::font_weight=bold )>style::text_properties
    <number::text
        <">"
    >number::text
    <number::year>number::year
>number:date_style;
```



Blasien

```
XmlWriter<style::StyleType>(stream)
<number::date_style( text::style_name="maxium_year" )
    <style::text_properties( fo::font_weight=bold )>style::text_properties
    <number::text
        <"<"
    >number::text
    <number::year>number::year
>number::date_style;
```

LibreOffice FastSerializer

```
GetExport().StartElement( XML_NAMESPACE_TEXT, XML_DATE_STYLE, false);
GetExport().AddAttribute( XML_NAMESPACE_TEXT, XML_STYLE_NAME,
                          GetExport().EncodeStyleName( "minimum_year") );
GetExport().StartElement( XML_NAMESPACE_TEXT, XML_TEXT_PROPERTIES, false);
GetExport().AddAttribute( XML_NAMESPACE_STYLE, XML_FONT_WEIGHT, sBold);
GetExport().EndElement( XML_NAMESPACE_TEXT, XML_TEXT_PROPERTIES, false);
GetExport().StartElement( XML_NAMESPACE_NUMBER, XML_TEXT, false);
GetExport().Characters("<");
GetExport().EndElement( XML_NAMESPACE_NUMBER, XML_TEXT, false);
GetExport().StartElement( XML_NAMESPACE_NUMBER, XML_YEAR, false);
GetExport().EndElement( XML_NAMESPACE_NUMBER, XML_YEAR, false);
GetExport().EndElement( XML_NAMESPACE_TEXT, XML_DATE_STYLE, false);
```

States and sinks

```
const HtmlTag html;
const BodyTag body;
const HtmlDocSink sink(stream)

<html
   <body
       <"hello"
   >body
>html;
```

```
<html>
  <body>hello</body>
</html>
```

```
const HtmlTag html;
const BodyTag body;

const HtmlDocSink sink(stream);
const HtmlSink sink2 = sink < html;
const BodySink sink3 = sink2 < body;
const BodySink sink4 = sink3 < "hello";
const HtmlSink sink5 = sink4 > body;
const HtmlDocSink sink6 = sink5 > html;
```

Operator overloading

```
sink < html < body < "hello" > body > html;
```

```
HtmlSink operator<(const HtmlDocSink& sink, const HtmlTag& tag) {
    sink.startElement(tag);
    return HtmlSink(sink);
}
BodySink operator<(const HtmlSink& sink, const BodyTag& tag) {
    sink.startElement(tag);
    return BodySink(sink);
}
BodySink operator<(const BodySink& sink, const char* text) {
    sink.writeCharacters(text);
    return sink;
}
HtmlSink operator>(const BodySink& sink, const BodyTag& tag) {
    sink.endElement();
    return sink.base;
}
HtmlDocSink operator>(const HtmlSink& sink, const HtmlTag& tag) {
    sink.endElement();
    return sink.base;
}
```

How to use it?

```
template <typename NodeType_>
class SafeSerializer {
public:
    static constexpr bool is_xmlsink = true;
    using NodeType = NodeType_;
    using StringType = OUString;
    SvXMLExport& serializer;
    explicit XmlWriter(SvXMLExport& s) :serializer(s) {}
    template <typename Tag>
    inline void startElement(const Tag &tag) const {
        serializer.StartElement(tag.ns(), tag.name());
    }
    inline void endElement() const {
        serializer.EndElement();
    }
    template <typename Tag>
    inline void writeAttribute(const Tag &tag, const OUString& value) const {
        serializer.AddAttribute(tag.ns(), tag.name(), value);
    }
    inline void writeCharacters(const OUString& value) const {
        serializer.Characters(value);
    }
};
```

```
#include <XHtml11.h>
using namespace xhtml;

struct create_paragraphs {
    const QList<QString> texts;
    template <typename Sink>
    Sink operator()(const Sink& sink) {
        for (const QString& text: texts) {
            sink <p<text>p;
        }
        return sink;
    }
};
QDomDocument
createDocument(const QString& docTitle, const QList<QString>& paragraphs) {
    QDomDocument dom("test");
    XmlBuilder<XHtmlDocument>(dom)
    <html
        <head
            <title
                <docTitle
            >title
        >head
        <body
            <create_paragraphs{{paragraphs}}
        >body
    >html;
    return dom;
}
```

Blasien

- tiny c++11 header library
- .rng → .h
- XML validation at compile time
- C++ looks like XML
- Prevention is better than cure
- Be strict in what you create...

```
http://vandenoever.info
https://github.com/vandenoever/blasien
```