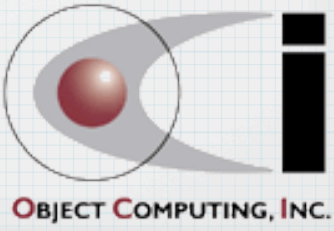
Writing API for XML (WAX)

http://ociweb.com/wax/

R. Mark Volkmann
mark@ociweb.com
March 2009



Why?

- Existing approaches to writing XML suffer from one or both of these **issues**
  - require **too much code**
  - use **too much memory**
- WAX addresses both!
- Wanted an application to convert "iTunes Music Library.xml" to more usable XML
  - mine is 8.7 MB
  - dict elements with key and "type" child elements

WAX Characteristics ...

* Requires less code than other approaches
* Uses less memory than other approaches
  * because it outputs XML as each method is called rather than storing it in a DOM-like structure and outputting it later
* Doesn't depend on any Java classes other than standard JDK classes
* A small library (around 27K)

... WAX Characteristics ...

* Writes all XML node types
* Always outputs well-formed XML or throws an exception
  * unless running in "trust me" mode
* Provides extensive error checking
* Automatically escapes special characters in text and attribute values
  * unless "unescaped" methods are used

... WAX Characteristics

* Allows most error checking to be turned off for performance
* Knows how to associate DTDs, XML Schemas and XSLT stylesheets with the XML it outputs
* Well-suited for writing XML request and response messages for REST-based and SOAP-based services

Just a Root Element

* To create this

```
<car/>
```

* Do this

```
WAX wax = new WAX();
wax.start("car").end().close();
```

* WAX constructor takes a String file path, an OutputStream, a Writer, or nothing to write to System.out
* end method terminates most recent start
* close method
  * terminates all unterminated elements and closes output destination
  * makes explicit end call here unnecessary

Root Element with Text

* To create this

```
<car>Prius</car>
```

* Do this

```
WAX wax = new WAX();
wax.start("car").text("Prius").close();
```


Child Element with Text

* To create this

```
<car>
  <model>Prius</model>
</car>
```

* Do this

```
WAX wax = new WAX();
wax.start("car").start("model").text("Prius").close();
```


Same with child Method

* To create this

```
<car>
  <model>Prius</model>
</car>
```

* Do this

```
WAX wax = new WAX();
wax.start("car").child("model", "Prius").close();
```

* child is a convenience method that is equivalent to calling start, text and end

Text in a CDATA Section

* To create this

```
<car>
  <model>
    <![CDATA[1<2>3&4'5"6]]>
  </model>
</car>
```

special characters in XML are
< > ' & "

* Do this

```
WAX wax = new WAX();
wax.start("car")
   .start("model").cdata("1<2>3&4'5\"6").close();
```


XML Without Indenting

* To create this

```
<car><model>Prius</model></car>
```

* Do this

```
WAX wax = new WAX();
wax.noIndentsOrLineSeparators(); // same as setIndent(null)
wax.start("car").child("model", "Prius").close();
```


Indent With 4 Spaces

* To create this

```
<car>
    <model>Prius</model>
</car>
```

* Do this

```
WAX wax = new WAX();
wax.setIndent("    "); // can also call setIndent(4)
wax.start("car").child("model", "Prius").close();
```

* Can pass to **setIndent**
  null, a single tab, or 0-4 spaces
  * default is 2 spaces

Add Attributes

* To create this

```
<car year="2008">
  <model>Prius</model>
</car>
```

* Do this

```
WAX wax = new WAX();
wax.start("car").attr("year", 2008)
   .child("model", "Prius").close();
```


XML Declaration

* To create this

```
<?xml version="1.0" encoding="UTF-8"?>
<car year="2008">
  <model>Prius</model>
</car>
```

* Do this

```
WAX wax = new WAX(Version.V1_0); // Version is an enum
wax.start("car").attr("year", 2008)
   .child("model", "Prius").close();
```


Comments

* To create this

```
<!-- This is a hybrid car. -->
<car year="2008">
  <model>Prius</model>
</car>
```

* Do this

```
WAX wax = new WAX();
wax.comment("This is a hybrid car.")
   .start("car").attr("year", 2008)
   .child("model", "Prius").close();
```


Processing Instructions

* To create this

```
<?target data?>
<car year="2008">
  <model>Prius</model>
</car>
```

* Do this

```
WAX wax = new WAX();
wax.processingInstruction("target", "data")
   .start("car").attr("year", 2008)
   .child("model", "Prius").close();
```


XSLT Stylesheet Ref.

* To create this

a special processing instruction

```
<?xml-stylesheet type="text/xsl" href="car.xslt"?>
<car year="2008">
  <model>Prius</model>
</car>
```

* Do this

```
WAX wax = new WAX();
wax.xslt("car.xslt")
   .start("car").attr("year", 2008)
   .child("model", "Prius").close();
```


Default Namespace

* To create this

```
<car year="2008"
  xmlns="http://www.ociweb.com/cars">
  <model>Prius</model>
</car>
```

* Do this

```
WAX wax = new WAX();
wax.start("car").attr("year", 2008)
   .defaultNamespace("http://www.ociweb.com/cars")
   .child("model", "Prius").close();
```

can't specify namespaces until a start tag has begun

Non-Default Namespaces

* To create this

```
<c:car year="2008"
  xmlns:c="http://www.ociweb.com/cars">
  <c:model>Prius</c:model>
</c:car>
```

* Do this

```
WAX wax = new WAX();
String prefix = "c";
wax.start(prefix, "car").attr("year", 2008)
   .namespace(prefix, "http://www.ociweb.com/cars")
   .child(prefix, "model", "Prius").close();
```


Associate an XML Schema

* To create this

```
<car year="2008"
  xmlns="http://www.ociweb.com/cars"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xsi:schemaLocation="http://www.ociweb.com/cars car.xsd">
  <model>Prius</model>
</car>
```

* Do this

```
WAX wax = new WAX();
wax.start("car").attr("year", 2008)
   .defaultNamespace("http://www.ociweb.com/cars", "car.xsd")
   .child("model", "Prius").close();
```


Multiple XML Schemas

* To create this

```
<car year="2008"
  xmlns="http://www.ociweb.com/cars"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:m="http://www.ociweb.com/model"
  xsi:schemaLocation="http://www.ociweb.com/cars car.xsd
    http://www.ociweb.com/model model.xsd">
  <m:model>Prius</m:model>
</car>
```

* Do this

```
WAX wax = new WAX();
wax.start("car").attr("year", 2008)
   .defaultNamespace("http://www.ociweb.com/cars", "car.xsd")
   .namespace("m","http://www.ociweb.com/model", "model.xsd")
   .child("m","model", "Prius").close();
```


Associate a DTD

* To create this

```
<!DOCTYPE car SYSTEM "car.dtd">
<car year="2008"
  <model>Prius</model>
</car>
```

* Do this

```
WAX wax = new WAX();
wax.dtd("car.dtd")
   .start("car").attr("year", 2008)
   .child("model", "Prius").close();
```


Entity Definitions & Use

* To create this

```
<!DOCTYPE root [
  <!ENTITY oci "Object Computing, Inc.">
  <!ENTITY moreData SYSTEM "http://www.ociweb.com/xml/moreData.xml">
]>
<root>
  The author works at &oci; in St. Louis, Missouri.
  &moreData;
</root>
```

* Do this

```
String url = "http://www.ociweb.com/xml/";
WAX wax = new WAX();
wax.entityDef("oci", "Object Computing, Inc.")
   .externalEntityDef("moreData", url + "moreData.xml")
   .start("root")
   .unescapedText(
       "The author works at &oci; in St. Louis, Missouri.", true)
   .unescapedText("&moreData;", true)
   .close();
```

text on a new line

Common Usage Pattern

* Method in Car class

```
public void toXML(WAX wax) {
    wax.start("car")
       .attr("year", year)
       .child("make", make)
       .child("model", model)
       .end();
}
```

* Sample output from `car.toXML(wax)`

```
<car year="2008">
  <make>Toyota</make>
  <model>Prius</model>
</car>
```


With Object Associations

* Method in **Address** class

```
public void toXML(WAX wax) {
    wax.start("address")
       .child("street", street)
       .child("city", city)
       .child("state", state)
       .child("zip", zip)
       .end();
}
```

* Method in **Person** class

```
public void toXML(WAX wax) {
    wax.start("person")
       .attr("birthdate", birthdate)
       .child("name", name);
    address.toXML(wax);
    wax.end();
}
```

* Sample output from `person.toXML(wax)`

```
<person birthdate="4/16/1961">
  <name>R. Mark Volkmann</name>
  <address>
    <street>123 Some Street</street>
    <city>Some City</city>
    <state>MO</state>
    <zip>12345</zip>
  </address>
</person>
```

Alternatively, methods that take a model object and a WAX object can be added to another class to avoid modifying model classes.

WAX Limitations

* Only writes XML, **doesn't read** it
  * for reading large XML documents a pull-parser API such as StAX is recommended
* **Doesn't validate** that the XML it outputs is valid according to some schema
* **Doesn't automatically serialize/deserialize** Java beans to/from XML like XStream
  * XStream is great when a direct mapping is correct

Interface Chaining Pattern ...

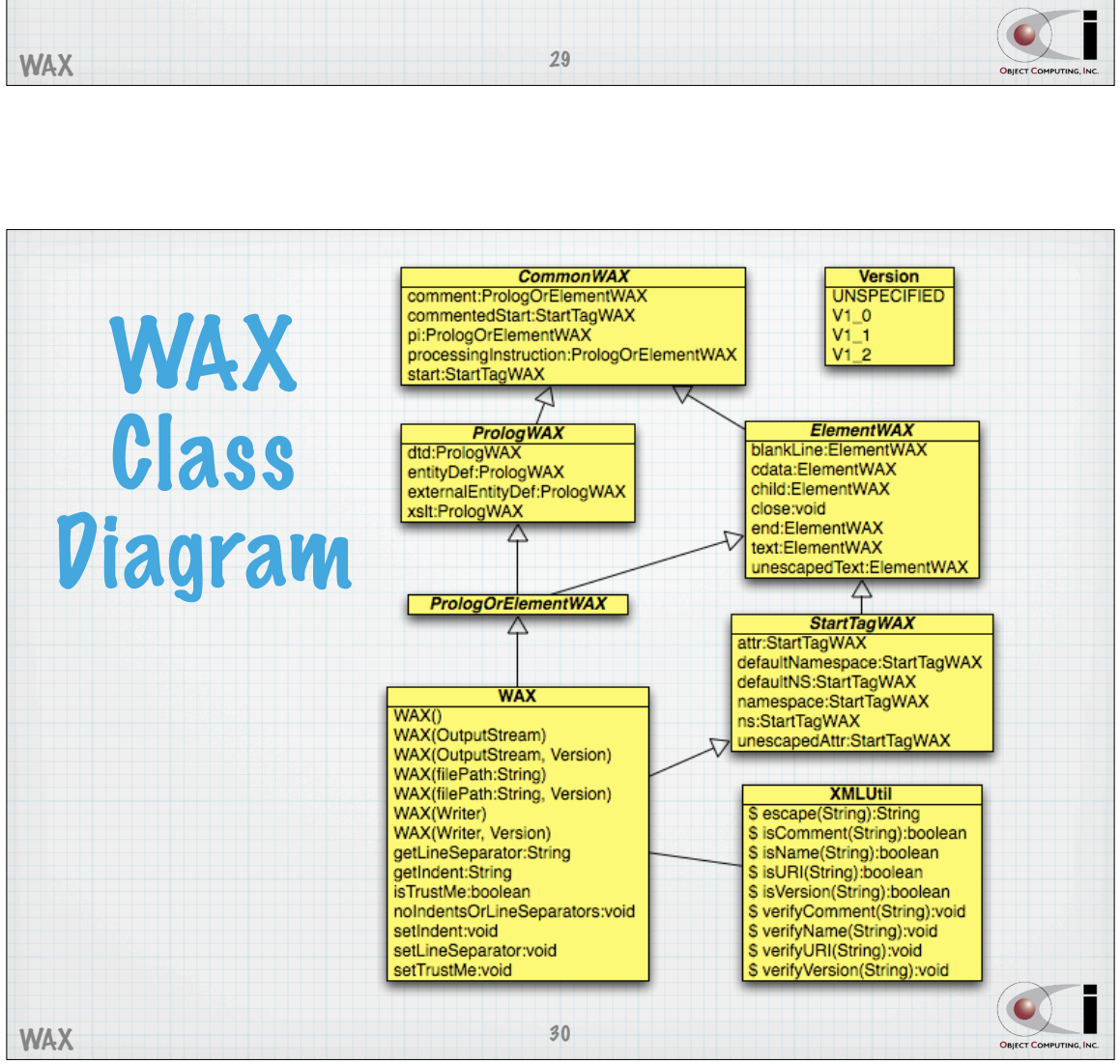
* Methods that write part of the XML output return the WAX object on which they are invoked to support method chaining
* Methods that configure WAX, including setIndent and setTrustMe, do not
* When method chaining is used, compile-time type checking verifies that each successive call is valid in the context of the previous call
  * for example, it's not valid to call attr immediately after calling text

... Interface Chaining Pattern ...

* A novel approach suggested by Brian Gilstrap at OCI
* WAX methods that return the WAX object return it as one of many interface types that are implemented by the WAX class rather than the WAX class type
* The interface returned describes only the WAX methods that are valid to invoke next
* Allows IDEs to flag invalid method chaining call sequences as code is entered

... Interface Chaining Pattern

* Downside to method chaining
  * if a method in the chain throws an exception, it may not be apparent which one threw it since the chain could invoke the same method multiple times
* The following UML diagram conveys all the details
  * note interface types implemented by WAX class
  * most WAX methods specify one of these interfaces as their return type



Bigger Example: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="artist.xslt"?>
<!DOCTYPE artist SYSTEM "http://www.ociweb.com/xml/music.dtd">
<artist name="Gardot, Melody"
  xmlns="http://www.ociweb.com/music"
  xmlns:date="http://www.ociweb.com/date"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xsi:schemaLocation="http://www.ociweb.com/music http://www.ociweb.com/xml/music.xsd
    http://www.ociweb.com/date http://www.ociweb.com/xml/date.xsd">
  <!-- This is one of my favorite CDs! -->
  <cd year="2008"/>
    <title>Worrisome Heart</title>
    <date:purchaseDate>4/3/2008</date:purchaseDate>
  </cd>
</artist>
```

associates with an XSLT stylesheet

associates with a DTD

associates with two XML Schemas

It's not normal to associate an XML document with both a DTD and an XML Schema. We're just doing it to show how both work.

Bigger Example: Code

```
import com.ociweb.xml.WAX;

public class CDDemo {

    public static void main(String[] args) {
        WAX wax = new WAX(WAX.Version.V1.0); // writing to stdout

        wax.xslt("artist.xslt")
           .dtd("http://www.ociweb.com/xml/music.dtd")

           .start("artist")
           .attr("name", "Gardot, Melody")

           // "" signifies the default namespace
           .defaultNamespace("http://www.ociweb.com/music",
               "http://www.ociweb.com/xml/music.xsd")
           .namespace("date", "http://www.ociweb.com/date",
               "http://www.ociweb.com/xml/date.xsd")

           .comment("This is one of my favorite CDs!")
           .start("cd").attr("year", "2008")
           .child("title", "Worrisome Heart")
           .child("date", "purchaseDate", "4/3/2008")

           .close();
    }
}
```


Namespace Handling

* WAX remembers
  the namespace declarations that are in-scope
  and verifies that
  only in-scope namespace prefixes
  are used on elements and attributes

Minimal Buffering

* WAX writes out bits of XML as calls are made
  * it doesn't buffer up the data in a data structure
    to be written out later,
    as is done in the DOM approach
  * actually it does buffer data for five cases,
    none of which involve a large amount of data

Five Cases of Buffering ...

1. **Entity definitions**, specified before the root element, are held in a list and written out in a DOCTYPE just before the root element start tag is output.
   This can't be done until the root element name is known.
   Once this is done, the list is cleared.

2. **Associations between namespace URIs and XML Schema paths**, specified using the defaultNamespace and namespace methods, are held in a map.
   This information is needed to construct the value of the xsi:schemaLocation attribute.
   After each start tag is completed, the map is cleared.

3. The names of **unterminated ancestor elements** are held in a stack.
   This is needed so they can be properly terminated when the end method is invoked, which pops one name off the stack.
   The close method calls end for each name remaining on this stack in order to terminate all unterminated elements.

... Five Cases of Buffering

4. All **namespace prefixes used on the current element or its attributes** are held in a list.
   When the start tag for the current element is closed, all the prefixes in this list are checked to verify that a matching namespace declaration is in scope.
   This is necessary because a namespace can be defined on the same element that uses the prefix for itself and/or its attributes.
   After the prefixes are verified, the list is cleared.

5. The **namespace prefixes** that are defined for each element are held in a stack.
   As each element is terminated, an entry is popped off this stack.
   This is used to verify that all namespace prefixes used on elements and attributes are in scope.

Well-formed Guarantee

- WAX always outputs well-formed XML or throws an exception
  - unless you are running in "trust me" mode or forget to call the close method when finished
  - if an exception is thrown then the tags already output may not be terminated
  - all exceptions thrown by WAX are runtime exceptions
    - IOExceptions are wrapped by RuntimeException

State Tracking ...

- WAX keeps track of the current state of the document in order to provide extensive error checking
- There are four states:
  1. IN_PROLOG - start tag for root element hasn't been output yet
  2. IN_START_TAG - start tag of current element has been written, but the > or /> at the end hasn't so attributes and namespace declarations can still be added
  3. AFTER_START_TAG - A > has been written at the end the start tag for the current element so it's ready for content such as text and child elements
  4. AFTER_ROOT - root element has been terminated; only comments and processing instructions can be output now

... State Tracking ...

- WAX uses the current state to determine whether specific method calls are valid
  - for example, if the state is IN_PROLOG, it doesn't make sense to call the attr method
  - attr adds an attribute to an element, but you haven't written any elements yet if you're still in the prolog section of the XML

... State Tracking

- When the state is IN_START_TAG, many methods trigger termination of the start tag
  - these include: cdata, child, close, comment, end, processingInstruction, start and text
  - this happens because none of these things can be written inside a start tag
  - methods that do not cause a start tag to be terminated include: attr and namespace because these are things that belong in a start tag

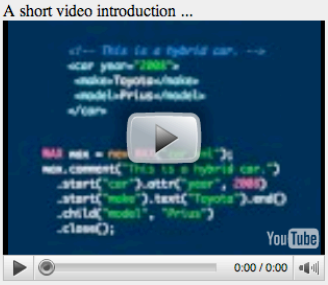
http://ociweb.com/wax/

Writing API for XML (WAX)

Download API (javadoc) WAX for Ruby Interface Chaining pattern

Contents

A short video introduction ...



Inspiration ...

Introduction

What's the best way to read a large XML document? Of course you'd use a SAX parser or a pull parser. What's the best way to write a large XML document? Building a DOM structure to describe a large XML document won't work because it won't fit in memory. Even if it did, it's not a simple API to use. There hasn't been a solution that is simple and memory efficient until now.

Google Code

Google Code **waxy**
Writing API for XML (ignore the "y")

Search Projects | Search the Web

Project Home | Downloads | Wiki | Issues | Source | Administer

WAX is a new approach to writing XML that

- focuses on writing XML, not reading it
- requires less code than other approaches
- uses less memory than other approaches (because it outputs XML as each method is called rather than storing it in a DOM-like structure and outputting it later)
- doesn't depend on any Java classes other than standard JDK classes
- is a small library (around 12K)
- writes all XML node types
- always outputs well-formed XML or throws an exception
- provides extensive error checking
- automatically escapes special characters in text and attribute values when error checking is turned on
- allows all error checking to be turned off for performance
- knows how to associate DTDs, XML Schemas and XSLT stylesheets with the XML it outputs
- is well-suited for writing XML request and response messages for REST-based and SOAP-based services

Code License: GNU Lesser General Public License
Labels: XML, Java

Links: home page
API (javadoc)

Project owners:
r.mark.volkmann

For more information, see the home page which includes usage examples.

The version of WAX for Java 1.4 requires retroweaver-rt-{version}.jar which is available under BSD license from http://retroweaver.sourceforge.net/.

Google Code **waxy**
Writing API for XML (ignore the "y")

Project Home | Downloads | Wiki | Issues | Source

New Download | Search Current Downloads for

Filename	Summary + Labels
wax14_1.0.1.jar	binary for Java 1.4
wax_1.0.1.jar	binary for Java 5 and above
wax_src_1.0.1.zip	source

From http://ociweb.com/wax/, For information about the Ruby version, click here. Ports for other programming languages will follow.

Ruby Version

There is also a C# version.

http://www.ociweb.com/mark/programming/wax_ruby.html

* To install
  * gem install wax
* Example - compare to Java code on slide 32

```
require 'wax'
url = "http://www.ociweb.com"
WAX.write($stdout, "1.0") do
  xslt "artist.xslt"
  dtd "#{url}/xml/music.dtd"
  start "artist"
  attr "name", "Gardot, Melody"
  namespace "", "#{url}/music", "#{url}/xml/music.xsd"
  namespace "date", "#{url}/date", "#{url}/xml/date.xsd"
  comment "This is one of my favorite CDs!"
  start "cd"
  attr "year", 2008
  child "title", "Worrisome Heart"
  child "date", "purchaseDate", "4/3/2008"
end
```


WAX On!

* LGPL licensed
* http://ociweb.com/wax/ contains
  * download link to Google Code
  * link to API documentation (javadoc)
  * video introduction (less than four minutes)
  * Java and Ruby WAX tutorials
  * inner details
  * comparisons to other approaches