



Making Your Code Readable and Maintainable — With XML & XSLT

Anthony B. Coates
Chief Technology Office, Reuters

`tony.coates@reuters.com`

XML DevCon Europe 2001

Overview

- » Why document?
- » What we have documented
- » What we will document
- » Types of documentation
- » Documentation tools
- » **xmLP** — a prototype Literate Programming tool for XML

Why Document?

- » This is the question continually repeated by first-year students (and Forth/APL programmers)
- » After all, everybody's code is **crystal clear** in its intent & workings
- » Plus, it's all **bug-free** — no need for future maintenance

Why Document?

- » OK, we all know that this isn't true
- » Code is crystal clear in that **fleeting moment** when you write it
- » Even authors struggle with their own code when they come back to it a month later
- » The problem is an **order of magnitude** worse for a non-author tasked to maintain that code

Why Document?: Why We Don't

- » There is no single reason, but many contribute:
- » *It takes too long*
- » *It's boring to go back and document the code after I have written it*
- » *The project manager didn't allow time for it*
- » *My yearly review depends on how many bugs were found in my code, not on how well I documented it*

Why Document?: Why We Don't

- » Plus there is an unspoken reason:
- » It's hard to explain “how it works” when your process is to **start writing without a plan of attack**, and then **just hack around** until it appears to work
- » The documentation you get is the documentation that your software process encourages

Why Document?: Why We Should

- » Less documentation means more time continually lost on “**reverse engineering of intent**”
- » My experience is that when I seriously document how a piece of code works, the first draft is not too convincing
- » So I rewrite it once or twice until it sounds plausible
- » In the process, I end up understanding what I **actually** have to do
- » That is often differs from my first impressions of what I had to do

Code To Document: Past

- » Source Code (text)
- » Makefiles
- » HTML (as text or as tagged markup)
- » Other text files: e.g. Windows INI files, Unix text DBs
- » Why are **text files** so common?
- » Because **text editors** are so common

Code To Document: Future

- » In addition, there will be:
- » XML Schemas
- » XSL (-T & -FO)
- » XML Queries (ABML, hopefully)
- » XMI (UML)

Code To Document: Future

- » Java, Eiffel, etc. **in XML**
- » 'Ant' build files
- » SVG?
- » Any other XML files
- » + anything else **not in XML** ...

Types of Documentation

- » Source code comments

```
macro: Sample Code Comments [1]
/*
 * Calculates the rounded result of "num / den".
 * As this is integer division, half the denominator
 * must be added to the numerator before dividing.
 */
int result = (num + den/2) / den;
```

- » The compiler (via the syntax) controls the order of such comments, not the author

Types of Documentation: Types of Documentation

- » Inverse comments is a system where only lines which start with a certain character sequence are used as program code (e.g. "% ")
- » All other lines are assumed to be comment text

macro: Sample Inverse Comment [2]

Calculates the rounded result of "num / den".
As this is integer division, half the denominator
must be added to the numerator before dividing.
`% int result = (num + den/2) / den;`

- » Can generate different types of documentation (text, HTML, (La)TeX), but difficult to support multiple output files

Types of Documentation: Types of Documentation

- » API documentation (e.g. Javadoc) is a different level of documentation
- » Only the **signatures** of methods and functions are documented, not their workings
- » What made Javadoc work is that the hyperlinking between classes, methods, and fields makes it relatively easy to navigate a large API to find what you need
- » Similar tools now available for XSL-T, XML Schema, C++, NetRexx, etc.

Types of Documentation: Types of Documentation

- » API documentation is a great complement to statement level documentation, but definitely not a replacement
- » Why does Javadoc only go down to the API level?
- » Because Javadoc implements a Java parser that **only** returns information down to the API level
- » With a fuller parser API, you could generate documentation with lower level detail
- » If Java was **expressed in XML**, rather than plain text, you would not have to be limited by the depth to which the Javadoc parser works

Types of Documentation: Types of Documentation

- » **Literate Programming** is a concept developed by Donald Knuth
- » He asked why people do not take computer programs to bed as suitable night-time reading
- » The answer was because computer programs are unreadable
- » Typical programs are 90% code, 10% comments
- » Knuth argued that to make a program widely understandable, you have to reverse this — 90% **comments**, 10% **code**

Types of Documentation: Types of Documentation

- » So, a literate document (or program) aims to be a human-readable document **first**, and a program **second**
- » Programs are built up from chunks/macros/fragments which can appear anywhere within the documentation, and in any order
- » The compiler no longer dictates the order in which the code appears
- » The code appears in the order which makes it easiest to **explain to humans**
- » Examples of literate programming will follow later, but you may have noticed that this presentation is itself a **literate document**

Types of Documentation: Types of Documentation

- » Separate documentation — there will always be some documentation in files separate to the program code itself
- » Such documentation always has **synchronisation** issues — who guarantees that this external documentation is properly updated whenever the code is?
- » However, even with literate documents, it is non-trivial to generate both technical and end-user documentation from a common source, because of the differing scope and goals of such documentation

Types of Documentation: Types of Documentation

- » Also, it may not be possible for **diagrams** or other illustrations to be updated automatically
- » Do the best you can — **minimise** the amount of documentation that is external to the file(s) with the code, but do not expect to eliminate it all

Documentation Tools: Javadoc

- » With Java 2, Sun made Javadoc pluggable by including a **Doclet** interface
- » Javadoc parses Java source files and reports the information (down to the API level) to the Doclet
- » The standard (default) Doclet produces the HTML Javadoc that has become so familiar (to Java developers, anyway)

Documentation Tools: Javadoc

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#) [NewsML Toolkit by Reuters & WAVO](#)
[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

org.xmlnews.newsml

Class NewsML

```
java.lang.Object
|
+--org.xmlnews.newsml.support.BaseObject
   |
   +--org.xmlnews.newsml.NewsMLNode
      |
      +--org.xmlnews.newsml.NewsML
```

```
public class NewsML
    extends NewsMLNode
```

Documentation Tools: Javadoc: JDocx

- » The standard Doclet produces HTML, but Doclets can just as easily produce XML
- » The first such Doclet was **JDocx**
(<http://www.componentregistry.com/entrypage.jsp>)
- » JDocx produces a **single XML file** and several XSL-T files (views)
- » This is far fewer files than the 1 Web page per class that Javadoc normally produces

Documentation Tools: Javadoc: JDocx

- » Having all of the information in a single XML file makes searching and manipulation easier
- » It also allows you to use XSL-T to create custom HTML
- » For most people, probably easier to write XSL-T than a Doclet
- » **Caveat** — in my testing, JDocx embedded a null character (�) in its output, which made it invalid XML, so I had to do a quick edit
- » **Note:** I have used Extensibility's **XML Instance** (<http://www.extensibility.com/>) for screenshots of XML output

Documentation Tools: Javadoc: JDocx

<model>	<package>	<classes>	<class> classType = class
[-]	[-]	[-]	[-] class classType = class accessSpecifier = public
			[-] className NewsML
			[-] classQualifiedName org.xmlnews.newsml.NewsML
		[-]	[-] description
			[-] briefDescription <![CDATA[The top-level NewsML package.]]>
			[-] fullDescription <![CDATA[The top-level NewsML package. <p>This class represents the root of the NewsML content tree. The NewsML specification defines packaging and metadata, but not content: in other words, a NewsML package shows how news objects go together, and provides information (metadata) about the news objects, but the objects themselves will appear in other formats (either XML-based or non XML-based).</p>

Documentation Tools: Javadoc: JDocx

<model>	<package>	<classes>	<class>	<method>
[-]				◆ method
				◆ accessSpecifier = public
[-]				◆ methodName
				▸ #text getType
[-]				◆ signature
				▸ #text ()
[-]				◆ return
[-]				◆ returnType
				▸ #text int
				◆ returnDescription <![CDATA[Always returns {@link NewsMLNode#NEWSML}.]]>
[-]				◆ description
				◆ briefDescription <![CDATA[Get the type of node in the NewsML content tree.]]>

Documentation Tools: Javadoc: Sun Java Doclet

- » JavaSoft have proposed an XML Doclet in a JDK release after 1.3, but meanwhile Sun has released a **Java Doclet** (<http://www.sun.com/xml/developers/doclet/>)
- » Produces **1 XML file per package + 1 XML file per class**, following existing Javadoc practice for HTML

Documentation Tools: Javadoc: Sun Java Doclet

<package> <classDesc> name = NewsMLNode	
[-] package	
• name =	org.xmlnews.newsml
[+] classDesc	• name... NewsMLNode
[-] classDesc	• name... NewsItem
[-] classDesc	• name... NewsML
[-] classDesc	• name... ContentItem
[-] classDesc	• name... NewsComponent
[-] classDesc	• name... NewsItemRef
[-] description	
[-] lead	
▸ #text	A reference to a news item located elsewhere.
[-] detail	<p>This class represents a cross-link in the NewsML content tree.

Documentation Tools: Javadoc: Sun Java Doclet

<class>	
[-] class	
• name =	NewsML
• inPackage =	org.xmlnews.newsml
• mod =	public
[-] superclass	• name = NewsMLNode • inPackage = org.xmlnews.newsml
[-] description	
[-] lead	
▸ #text	The top-level NewsML package.
[-] detail	<p>This class represents the root of the NewsML content tree. The NewsML specification defines packaging and metadata, but not content. In other words, a NewsML package shows how news objects go together, and provides information

Documentation Tools: Javadoc: Sun Java Doclet

<class>	<methods>
[-] ◆ methods	
[-] ◆ method	
◆ name =	getType
◆ mod =	public
◆ signature =	getType()
[-] ◆ returns	
◆ stype =	int
▼ #text	Always returns {@link NewsMLNode#NEWSML}.
[-] ◆ description	
[-] ◆ lead	
▼ #text	Get the type of node in the NewsML content tree.
[-] ◆ overrides	
◆ name =	NewsMLNode
◆ inPackage =	org.xmlnews.newsml

Documentation Tools: Javadoc: Cocoon JavaXMLDoclet

- » The **Apache Cocoon** (<http://xml.apache.org/cocoon/>) project has an XML Doclet under development
- » It is only at an “alpha” stage of development
- » Not reviewed here, because it is only sporadically available
- » Something to keep in mind for the future

Documentation Tools: XSL-T: Xsldoc

- » **Xsldoc** (<http://www.xsldoc.org/>) is an API documentation tool for XSL-T
- » Defines its own pluggable **Doclet API** with a similar feel to the Javadoc Doclet API
- » The standard Xsldoc Doclet produces HTML
- » Similar to Javadoc at command-line level, and hence familiar
- » No screenshot necessary — it really looks like Javadoc
- » Just replace **classes with stylesheets** and **methods with templates**

Documentation Tools: XSL-T: Xsldoc

- » Requires a certain style of commenting, just as Javadoc does
- » If your project uses both Java & XSL-T significantly, you will appreciate having the same output format for both Java & XSL-T documentation

```
macro: Sample Xsldoc Comment [3]
<!--
  This is a <b>doc</b> comment.
  @see app\user\UserDetail.xml
-->
```

Documentation Tools: XSL-T: XSLTDoc

- » **XSLTDoc** (<http://www.jenitennison.com/xslt/utilities/>) is an XSL-T formatter that runs inside IE5 (using the XSL-T implementation of MSXML)
- » Unlike the other tools, XSLTDoc is designed to give a **quick overview** of a script, not for generating static documentation
- » Start by opening **xslt-doc.xsl** in IE5
- » A dialog prompts you for the XSL-T file to document

Documentation Tools: XSL-T: XSLTDoc

T:\Presentations\Making-Your-Code-Readable-and-Maintainable\xmlPtangle.xsl

Output Control	Method	Public	System
xsl:output	text		

Variables & Parameters	Name	Select	Content
xsl:param	XmlIndent	substring(\$tempIndent,2,1)	
xsl:variable	tempIndent		x x

Templates	Match	Mode	Priority
xsl:template	*@* text() comment()		
xsl:template	/		

Named Templates	Name
xsl:template	invoke-macro
xsl:template	tangle-text
xsl:template	tangle-xml

Documentation Tools: XSL-T: XSLTDoc

Named Templates	Name
xsl:template	invoke-macro
xsl:template	tangle-text
xsl:template	tangle-xml

```
<!-- Expands an invoked macro. -->
<xsl:template name="invoke-macro">
  <xsl:param name="macroName" />
  <xsl:for-each select="//lp:macro
[normalize-space(lp:name) = $macroName]">
    <xsl:for-each select="lp:text |
lp:xml">
      <xsl:if test="self::lp:text">
        <xsl:call-template
name="tangle-text" />
      </xsl:if>
      <xsl:if test="self::lp:xml">
        <xsl:call-template
name="tangle-xml" />
      </xsl:if>
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>
```

[XSLT Recommendation definition](#)

The xsl:template element is used to define a piece of XSLT code that will be run either when certain nodes are found within the source XML, or when the template is called by name.

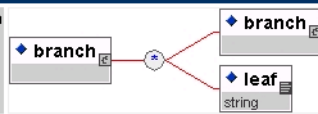
In this case, the template can be called with the name invoke-macro.

The template will use the parameters passed to it that are defined by the contained xsl:param elements.


Documentation Tools: XML Schemas: SchemaDOC

- » **SchemaDOC** is an XML Schema documentor that is built into Extensibility's **XML Console** (<http://www.extensibility.com/>)
- » Documents Schema constructs with diagrams (similar format to XML Authority)
- » Not only documents Schemas, but **analyses** them for items such as unused elements

Documentation Tools: XML Schemas: SchemaDOC

Element branch	
diagram	 A diagram showing a choice between two elements: 'branch' and 'leaf'. The 'branch' element is shown with a diamond icon and a box containing 'branch'. The 'leaf' element is shown with a diamond icon and a box containing 'leaf' and 'string'. A circle with a plus sign is connected to both elements, indicating a choice.
description	There can be one or more branches between the root and any leaf.
attributes	
uses	branch , leaf
used by	root branch
content	choice
source	<pre>-<xsd:element name="branch"> -<xsd:annotation> <xsd:documentation>There can be one or more branches between the root and any leaf.</xsd:documentation> </xsd:annotation> -<xsd:complexType> -<xsd:choice minOccurs="0" maxOccurs="unbounded"> <xsd:element ref="branch" /> <xsd:element ref="leaf" /> </xsd:choice> </xsd:complexType> </xsd:element></pre>

Documentation Tools: XML Schemas: SchemaDOC

Element island	
diagram	 A diagram showing an unused element 'island' with a diamond icon and a box containing 'island' and 'string'.
description	Island is an unused element in this Schema.
attributes	
uses	Does not reference other elements.
used by	Not used by other elements. Probably an orphan.
content	sequence (default)
type	xsd:string
source	<pre>-<xsd:element name="island" type="xsd:string"> -<xsd:annotation> <xsd:documentation>Island is an unused element in this Schema.</xsd:documentation> </xsd:annotation> </xsd:element></pre>

Literate Programming Tools

- » **Literate programming** (<http://www.literateprogramming.com/>) tools allow code fragments to be embedded in a human readable document
- » For historical reasons, the human readable document have generally been rendered using (La)TeX
- » Sometimes the (La)TeX markup was hidden, sometimes it was exposed

Literate Programming Tools

- » The code fragments have been **plain text only**
- » The markup for code fragments has been **tool specific**
- » Hence no interoperability between LitProg tools

Literate Programming Tools

- » LitProg tools to date operate in **batch mode**
- » You write your literate document in an editor first
- » The tool **tangles** your document to extract the code fragments and assemble them in order
- » It also **weaves** the source document to produce the output documentation, often with additions like cross-references

Literate Programming Tools

- » LitProg tools can be broken into two broad classes
- » **Syntactic tools** provide markup for expressing some of the syntax of a language
- » Allows some sanity checking before compilation, but limits the variety of languages or document types that you can work with
- » For example, Makefiles are not a supported language type, but can be convenient to include in your literate document

Literate Programming Tools

- » **Non-syntactic tools** treat all code as meaningless text
- » This is more flexible, but you lose the sanity checking
- » With a non-syntactic tool, any text file can be written literately, so you are not confined to languages
- » You can document Unix text databases, or even a list of names
- » I have always used non-syntactic tools, because I have always needed unsupported language types — Makefiles, Maple, Mathematica

xmlLP = XML LitProg

- » **xmlLP** (<http://about.reuters.com/researchandstandards/firstcontact/xmlp/>) is my demonstration of how non-syntactic LitProg can be done in XML using XSL-T
- » It is a “work in progress”, most recently updated to accommodate my XML DevCon talks
- » xmlLP is implemented in about **500 lines** of XSL-T code
- » Compare that to thousands or tens of thousands of lines of code for a traditional LitProg tool

xmlLP = XML LitProg

- » The code samples in this presentation are embedded as xmlLP macros (i.e. code fragments) in the XML source

macro: Unused Macro [4]

The text in this macro
is a display sample only
and is not used otherwise

- » To embed this in the presentation, I added the following to my presentation. **Note:** Generally, text blocks will need to be encased in a CDATA section to preserve linefeeds, etc., but this will not be shown:

xmlLP = XML LitProg

macro: Markup for Unused Macro [5]

```
<lp:macro lp:usage="never" lp:final="true">  
  <lp:name>Unused Macro</lp:name>  
  <lp:text>The text in this macro  
is a display sample only  
and is not used otherwise</lp:text>  
</lp:macro>
```

- » A macro's usage can be **never** (*not used in code*), **once** (*default; must be used, but only once*), or **multiple** (*used any number of times*)

xmlLP = XML LitProg

- » A **final** macro can be defined in only one place. Otherwise, the macro can be defined in multiple places, and those definitions are appended in order
- » A macro's name is defined in an **lp:name** element
- » This means that macros can have rich names to suit specialist formatting conventions
- » For example, you could use MathML in macro names
- » Two macro names are considered identical if **xsl:value-of** returns the same text string for the two names

xmlLP = XML LitProg

- » A macro can contain a sequence of **lp:text** or **lp:xml** elements
- » **lp:text** is for plain text, **lp:xml** is for balanced XML

```
macro: Unused XML Macro [6]
<root>
  <branch>
    <leaf name = "c1">Red</leaf>
    <leaf name = "c2">Green</leaf>
  </branch>
</root>
```

xmlLP = XML LitProg

```
macro: Markup for Unused XML Macro [7]
<lp:macro lp:usage="never">
  <lp:name>Unused XML Macro</lp:name>
  <lp:xml>
    <root>
      <branch>
        <leaf name="c1">Red</leaf>
        <leaf name="c2">Green</leaf>
      </branch>
    </root>
  </lp:xml>
</lp:macro>
```

xmlLP = XML LitProg

- » Macros are not useful unless you can use them elsewhere
- » **lp:invoke** is used to invoke (call) a macro, and is replaced by the contents of the macro during **tangling**

```
macro: How to use lp:invoke [8]
...
  <lp:invoke>
    <lp:name>branch Template</lp:name>
  </lp:invoke>
...
```

- » As an example, the remaining major xmlLP element, **lp:file**, will be used in building a file from macros

xmlLP = XML LitProg

```
file #1: files/quickstyle.xsl
<xsl:stylesheet>
  branch Template [9]
  leaf Template [10]
</xsl:stylesheet>

macro: branch Template [9]
<xsl:template match = "branch">
  <bread>
    <xsl:apply-templates/>
  </bread>
</xsl:template>
Invoked in files #: 1
```

xmlLP = XML LitProg

```
macro: leaf Template [10]
<xsl:template match = "leaf">
  <loaf>
    <xsl:apply-templates/>
  </loaf>
</xsl:template>
Invoked in files #: 1
```

» The resultant product file **quickstyle.xsl** follows:

xmLP = XML LitProg

```
macro: quickstyle.xsl [11]
<xsl:stylesheet>
  <xsl:template match="branch">
    <bread>
      <xsl:apply-templates/>
    </bread>
  </xsl:template>
  <xsl:template match="leaf">
    <loaf>
      <xsl:apply-templates/>
    </loaf>
  </xsl:template>
</xsl:stylesheet>
```

xmLP = XML LitProg

- » OK, this is a trivial example, but keep in mind the code was **embedded in this presentation itself**
- » In practice, you would embed **xmLP** tags in XHTML (if shorter) or DocBook (if longer) or anything else that suits
- » **xmLP** is not production quality as it stands, but I would welcome anyone to contribute suggestions or write their own equivalent (ideally using the same or similar tags)
- » What **xmLP** really needs is test cases — something for the future

Conclusion

- » I hope this has given you a picture of what is done and can be done with documentation
- » I hope this has given you some ideas as to what you could do with Literate Programming
- » Any & all feedback is welcome
- » If you want to see a more significant example of LitProg in action, come see my other DevCon talk: **Publishing Your Stuff With NewsML**

Conclusion

- » This presentation was written in XML and converted to PDF using XSL (XSL-T & XSL-FO)
- » Thanks to **RenderX** (<http://www.renderx.com/>) for working on their **XEP** XSL-FO engine over a weekend so that I could do this presentation
- » The **xml-litprog-l mailing list** (<http://www.egroups.com/group/xml-litprog-l/>) is hosted at **eGroups** (<http://www.egroups.com/>); membership open to all
- » The **xml-doc mailing list** (<http://www.egroups.com/group/xml-doc/>) is hosted at **eGroups** (<http://www.egroups.com/>); membership open to all
- » The **latest version of this presentation** (<http://about.reuters.com/researchandstandards/events/2001/02/xml-devcon-europe-xml-doc/>) will be available on the Web