



## **FastXML White Paper**

**Version 1.2**

FastXML – Intelligent Model Driven XML Parsing	Version: 1.2
White Paper	

## Table of Contents

<b>1</b>	<b>Introduction to FastXML .....</b>	<b>2</b>
<b>2</b>	<b>Traditional XML Parsing and Processing .....</b>	<b>3</b>
<b>3</b>	<b>What’s Different About FastXML? .....</b>	<b>3</b>
<b>4</b>	<b>Using FastXML .....</b>	<b>4</b>
<b>5</b>	<b>Fast XML in Practice .....</b>	<b>6</b>
<b>6</b>	<b>Conclusion .....</b>	<b>7</b>
<b>7</b>	<b>Appendix .....</b>	<b>9</b>

FastXML – Intelligent Model Driven XML Parsing	Version: 1.2
White Paper	

# 1 Introduction to FastXML

FastXML is a new patent pending software technology for model-driven generating performance tuned xml parsers which parse and process XML documents substantially faster than other known methods such as DOM, XMLBeans, SAX and StAX.

## Performance Tuned

Performance tuning refers to an ability to remove overhead of layering and unnecessary intermediate processing by generating XML parsers specifically for a given XML Schema (xsd) and processing API. XML parsers are generated with the foreknowledge of which XML elements are needed at runtime thus only parsing the requisite elements, converting them into required types, and invoking processing API. FastXML's ability to tune each parser comes from its decorative configuration model in which XML processing instructions are written in Java against the XML Schema (xsd). This direct "inlining" of Java processing instructions against the xml schema and ability to manage and automatically combine multiple decorations into a single processing unit of work is what separates FastXML from most other layered approaches such as DOM which first parse the XML and then apply processing instructions. Some other technologies such as SAX are similar to FastXML in that processing instructions are invoked at parse time. However with SAX there is no simple method for combining and managing multiple processing logics into a single parser. Thus FastXML retains a 2.5x performance edge over SAX and a 5x advantage over DOM.

## Model Driven

Configuring and generating FastXML parsers is quick and easy due its simple decorative configuration model. Configuration is done by loading an XML Schema (xsd) into the configuration tool. The schema is displayed as a hierarchy and then Java processing instructions are added to desired elements. The processing instructions are invoked at parse time when the associated element(s) are encountered. This provides an intuitive and quick approach for configuring the FastXML parser.

## Multiple Decorations

FastXML allows multi-decoration of the XML schema. When Java code is added to the schema this is termed a decoration. It is possible to have more than one decoration against a schema. When the FastXML parser is generated, these decorations will be merged. At parse time the decorations are effectively invoked in parallel to each other. This allows for clean separation of concern and efficient management of multiple processing modules.

FastXML – Intelligent Model Driven XML Parsing	Version: 1.2
White Paper	

## 2 Traditional XML Parsing and Processing

XML has been around for 10 years. Yet, enterprise applications still process XML documents in two main ways: event-based based models such as the Simple API for XML (SAX) parse an XML document and generate an event for each XML token parsed; tree-based models such as the Document Object Model (DOM) parse an entire document into memory; then, applications traverse a generic tree.

These two approaches to processing XML have significant shortcomings: event-based models impose complex event-handling requirements—making it expensive to modify and test application source code if the schema changes; and tree-based models are much slower and use extensive amounts of memory. Both approaches tend to entangle the structure of the document with the application's source code (e.g. use of XPath in the application). Application architects avoid compromising their application designs by introducing an additional layer of indirection, which often introduces additional latency.

Before XML, enterprise applications exchanged data using hierarchical Electronic Data Interchange (EDI) documents. Businesses often found that they needed to introduce new data elements into these documents, which often meant that developers would have to modify the parsers themselves to accommodate new elements and attributes. EDI-based systems provided separate validation/compliance modules, because EDI didn't have a simple method of validating document structures for compliance with a schema.

XML's metadata and schema validation eliminated the need to modify parsers, change document syntax, and develop separate validation/compliance modules, which made XML a natural successor to EDI. However, XML left behind a material benefit of custom parsers: performance.

Specialized systems such as hardware-based IBM's Datapower and software-based Intel's Sarvega accelerate only specific tasks of validation and transformation of XML documents – specifically XSLT. Since these systems physically separate validation and transformation from the consuming application they cannot directly integrate and optimize low level parsing with API integration. Although additional processing layer may result in improved overall throughput, but in comparison to FastXML the total computation performed by all layers for end-to-end processing is likely to be greater thus resulting in higher latency.

Clearly, applications will benefit substantially from a new, "layer-less" high-performance approach to processing XML documents.

## 3 What's Different About FastXML?

### Throughput

Increased transactional volume without exponential increase in cost. FastXML generated parsers exhibit linear performance characteristics proportionate to the size of the input xml document.

### Rapid Development

Configuration of FastXML parsers is done visually by decorating the XML Schema with processing instructions.

### Ease of Maintenance

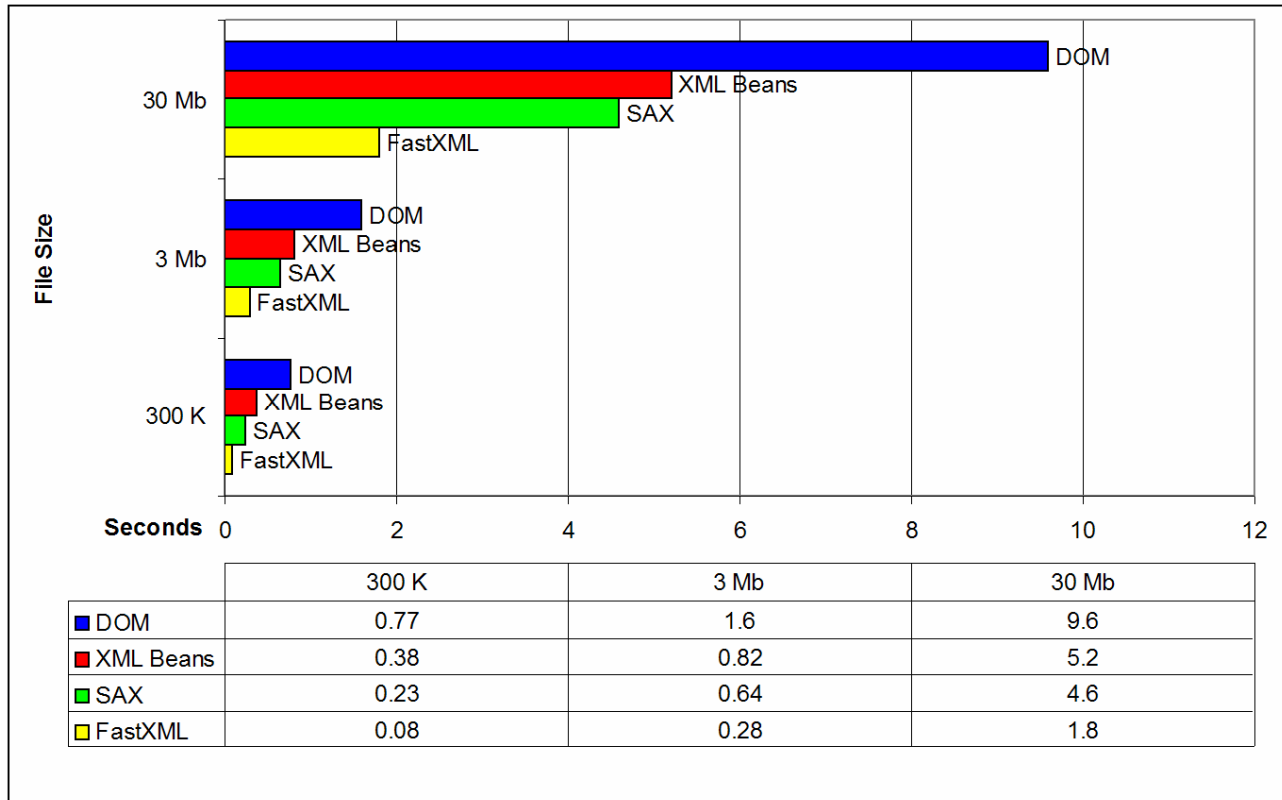
The visual nature of configuration provides for quick understanding of parser logic facilitating quick changes. Ability to independently manage multiple decorations further facilitates maintainability due to separation of concerns.

### Embeddable

FastXML parsers are POJOs and can therefore be embedded into any Java application.

## Benchmarks

The chart below shows parsing time of FastXML vs. other mainstream parsing technologies. Three different file sizes are compared with FastXML begin over 2.5Xs faster than event SAX. Detailed description of benchmark tests is included in the Appendix.



## 4 Using FastXML

### Parser Lifecycle

Figure 1 below shows how FastXML configuration tool is used. Currently there are two ways to configure FastXML parsers. There is an Eclipse plug-in which provides for a seamless development experience. Also, if preferred there is a browser-based Web version of the configuration tool. Both tools have similar look and feel. Eclipse plug-in is a preferred method of using FastXML. A limited browser-based Web version should be used for evaluation purposes or when using Eclipse is not desirable. Both tools offer a similar look-and-feel.

Using FastXML can be described in four steps of Upload, Decorate, Generate, and Run.

#### Upload

Add the xsd to the Eclipse project or upload file to the fastxml.net website

#### Decorate

Once the xsd has been added to the Eclipse project (or uploaded to the fastxml.net website) it can now be decorated with processing instructions. Decoration is performed by first clicking on a schema element on the left and then adding the instructions that should execute when a matching xml element is

FastXML – Intelligent Model Driven XML Parsing	Version: 1.2
White Paper	

encountered at parse time. See the FastXML User’s Guide for more information on how to write processing instructions.

**Generate**

Once the desired processing instructions have been added to the schema you can now generate your custom intelligent parser.

**Run**

The generated Java file can either be included in another project or run as a standalone program with the built-in main() function.

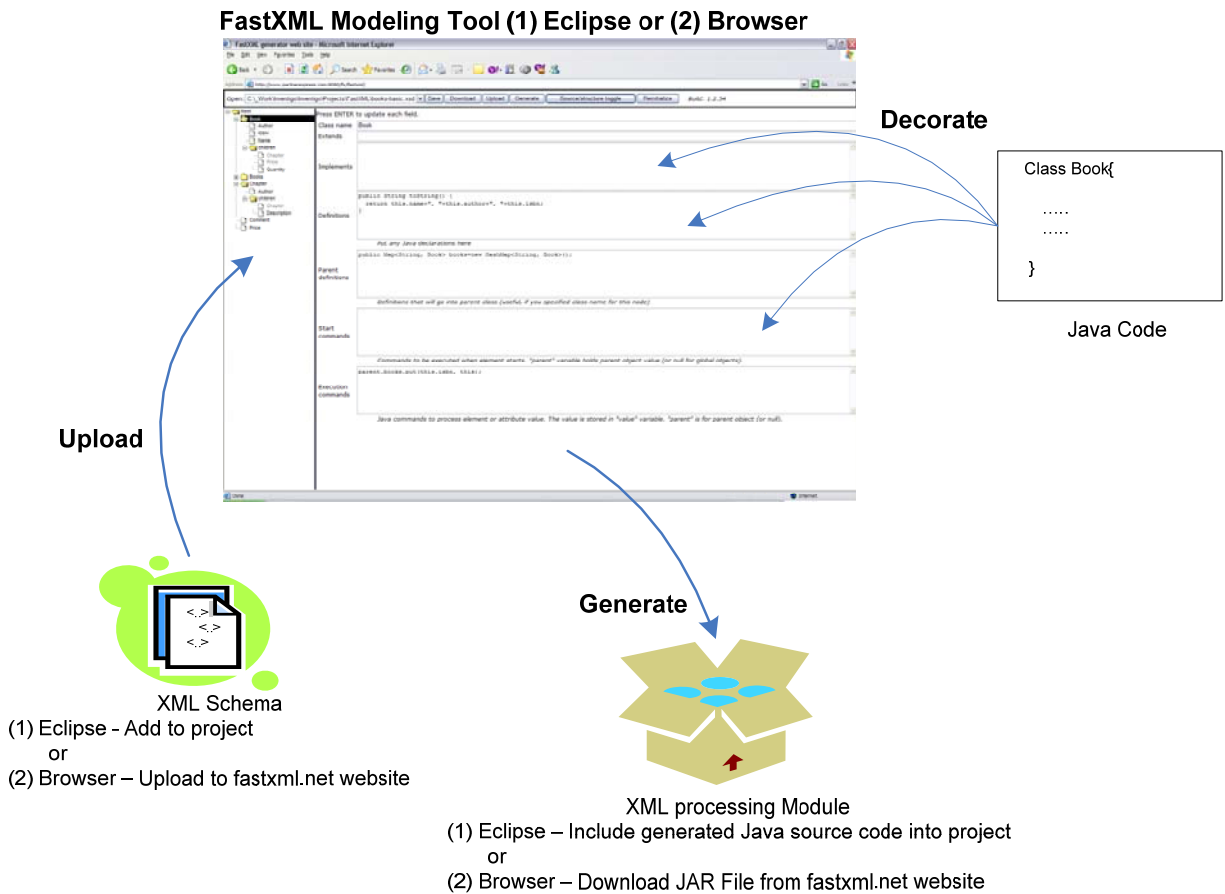


Figure 1 – FastXML Configuration and Code Generation

**Configuration Tools**

Figure 2 below shows the configuration GUI screen.

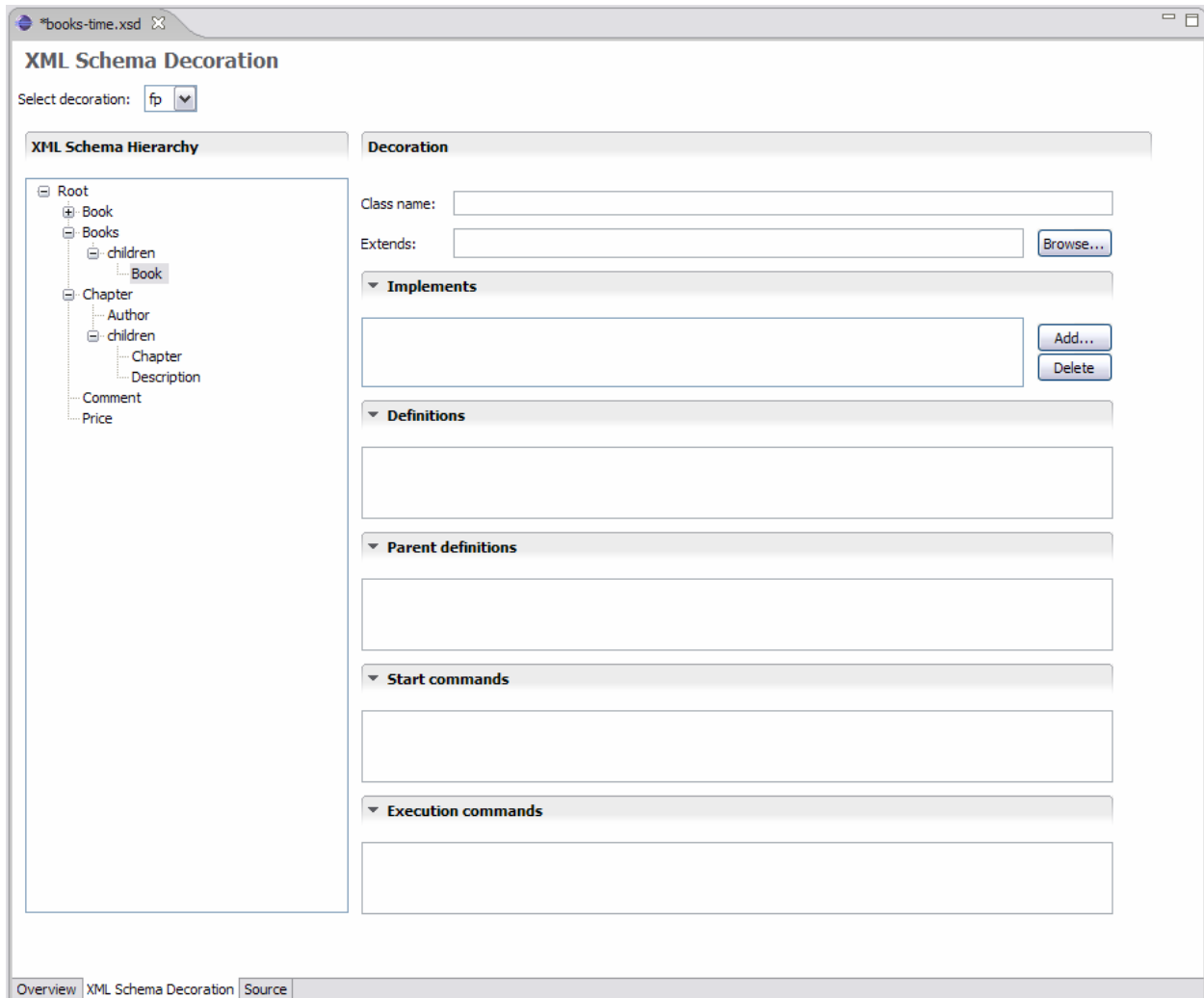


Figure 2 – Configuration GUI

### Developing with FastXML

When a FastXML parser is configured via an Eclipse Plug-in a Java source code file is produced. When parser is configured via a Web configuration tool a JAR file is generated. The JAR file contains the schema, generated processing classes and the Java source code.

## 5 Fast XML in Practice

FastXML can be used in any place where high volume time critical XML processing is required. It is particularly useful in situations where the entire XML payload is not needed rather only certain elements are needed and processed by multiple logical units.

### Enterprise Message Bus and Complex Event Processing

FastXML can be used as a pluggable component into an ESB in order to improve the ESB quality of service. One particular usage is in the implementation of a Complex Event Processor (CEP).

FastXML – Intelligent Model Driven XML Parsing	Version: 1.2
White Paper	

CEP is about capturing and analyzing high-volume streams of events, identifying sophisticated patterns, and applying time-aware event correlation and decision logic against those patterns. A CEP component should be capable of handling partial unpackaging of data for evaluation, parallel evaluations of multiple rules, and invocation of actions based upon the evaluations.

FastXML is a glove fit for a CEP operation as it possesses all of the qualities which allow it to excel in this role. FastXML as the core parser for a CEP component would allow fast execution, easy configuration, easy maintenance and plugability into any Java framework.

### Domain Specific Modeling

FastXML model-driven approach is complimentary to Domain-Specific Modeling (DSM). As Booch et al. say (Grady Booch, Alan Brown, Sridhar Iyengar, Jim Rumbaugh, Bran Selic, MDA Journal, May 2004) "the full value of MDA is only achieved when the modeling concepts map directly to domain concepts rather than computer technology concepts." This is consistent with the industrial experiences of DSM that consistently show it to be 5-10 times faster than current practices, including current UML-based implementations of MDA.

With DSM-enabled FastXML Configuration IDE your experts will define domain-specific processing modules that can be simply used during the document schema decoration process - the final processing module will be generated utilizing these high-level specifications - the resulting code is better than most developers write by hand. No "one size fits all" code, no stubs, no "round trip" problems. Instead, high-performance, top quality code.

The DSM-enabled FastXML would further compress time to market for efficient processing documents in XML or other formats and reduce risk of errors from misunderstandings.

## 6 Conclusion

In summary FastXML is a patent pending technology that is:

- Performance tuned. High speed and low memory consumption. Faster than other mainstream parsers such as DOM, SAX, StAX, and XMLBeans. Excels in situations where message peeking is preferred vs total message parsing
- Model-driven. Provides method for creating schema-independent applications. Generates custom parser that combines logic for type safe application integration with on-the-fly method invocation of each processing scenario
- Embeddable. Generated parsers are POJOs (source code included) and therefore pluggable into any Java framework

FastXML offers the following benefits:

Function/ Feature	Benefits	Know how
<b>Operations</b>		
1) Throughput	Increased transaction volume without exponential increase in cost.	Parallel invocation of services  Linear correlation between message size and processing time
2) Low Latency	Reduced per transaction cost	Overlapping of parsing and processing

FastXML – Intelligent Model Driven XML Parsing	Version: 1.2
White Paper	

3) Production Grade	Low level infrastructure is in place. Only user configuration has to be debugged.	Model driven code generation
4) Embeddable	Can be used as a plug in or run stand alone	Generates POJO source code
<b>Development</b>		
1) Rapid Development	No need to deal with low level parsing  Clear separation of concerns (allows building of applications with out concern with input formats)	IDE for model driven decoration  Supports multiple decorations
2) Light Weight	IDE is lightweight (browser-based and plug-in to Eclipse)  No constraints on the development environment	IDE is available as Software as a Service (SaaS) in browser or plug-in to Eclipse  IDE is available as a self contained installed plug-in to Eclipse
3) Open	Complementary to any infrastructure	Open to integration with any Java library  Generates standard Java code
4) Easy to Verticalize	Allows bridging the gap between technicians and domain experts	IDE can be easily verticalized into Domain Specific Modeling (DSM) with policies for specific domains (monitoring, archiving, domain specific message standards, etc).

FastXML – Intelligent Model Driven XML Parsing	Version: 1.2
White Paper	

## 7 Appendix

### FastXML Benchmarks Test Details

System	IBM ThinkPad Laptop	Test Description
CPU	Pentium M, 1,300Mhz	Test #1: Count number of books
Memory	768M	Test #2: Find max and min price

#### Input XML

File Name	Books10000.xml	Books1000.xml	Books100.xml
File Size	31,952,564 bytes	3,186,462 bytes	310,869 bytes
Element Count	10,000	1,000	100

#### Time Measurement Details

	FastXML	DOM		SAX	
	Time (ms)	Time (ms)	FastXML vs. DOM (times)	Time (ms)	FastXML vs. SAX (times)
Books10000.xml	1,873	9,654	5.15	4,637	2.48
Books1000.xml	280	1,622	5.79	641	2.29
Books100.xml	90	771	8.57	230	2.56

#### Output Details

	FastXML	DOM	SAX
Books10000.xml	Book Count: 10000 Max Price: 199.9946680224142 Min Price: 0.05024320880813171 Parsing time=1873 Executed 1 Executed 2	Book Count:10000 Prices Count: 10000 Max Price: 199.9946680224142 Min Price: 0.05024320880813171 Execution Time: 9654ms	Book Count:10000 Max Price: 199.9946680224142 Min Price: 0.05024320880813171 Execution Time: 4637ms
Books1000.xml	Book Count: 1000 Max Price: 199.91770300146942 Min Price: 0.28968621496794356 Parsing time=280 Executed 1 Executed 2	Book Count:1000 Prices Count: 1000 Max Price: 199.91770300146942 Min Price: 0.28968621496794356 Execution Time: 1622ms	Book Count:1000 Max Price: 199.91770300146942 Min Price: 0.28968621496794356 Execution Time: 641ms
Books100.xml	Book Count: 100 Max Price: 196.88429050438887 Min Price: 8.253809753568909 Parsing time=90 Executed 1 Executed 2	Book Count:100 Prices Count: 100 Max Price: 196.88429050438887 Min Price: 8.253809753568909 Execution Time: 771ms	Book Count:100 Max Price: 196.88429050438887 Min Price: 8.253809753568909 Execution Time: 230ms