Java^{TM} Architecture for XML Binding

Scott Fordin
Sun Microsystems
Java and XML Technologies
July, 2003

NOW IN THE JAVA^{TM} WEB SERVICES DEVELOPER PACK!

RELATED LINKS

- Java Web Services Developer Pack
- JAXB Home Page
- Java Web Services and XML

The Java^{TM} Architecture for XML Binding (JAXB) is a new Java technology in the Java Web Services Developer Pack (JWSDP) that enables you to generate Java classes from XML schemas.

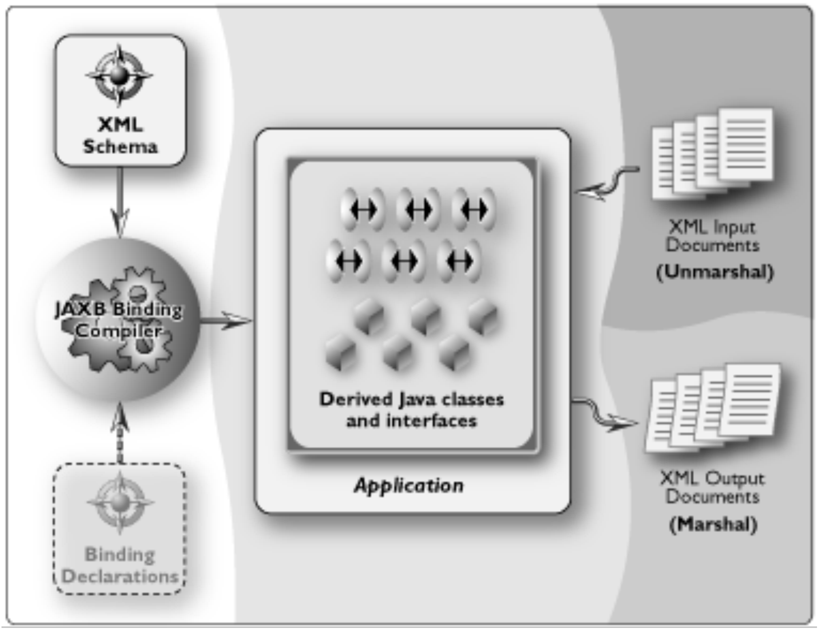
JAXB provides a free, fast, and convenient way to bind XML schemas to Java representations, making it easy for Java developers to incorporate XML data and processing functions in Java applications without having to know much about XML itself.

What is JAXB?

JAXB is a Java technology that enables you to generate Java classes from XML schemas by means of a *JAXB binding compiler*. The JAXB binding compiler takes XML schemas as input, and then generates a package of Java classes and interfaces that reflect the rules defined in the source schema. These generated classes and interfaces are in turn compiled and combined with a set of common JAXB utility packages to provide a *JAXB binding framework*.

The JAXB binding framework provides methods for unmarshalling XML instance documents into *Java content trees* -- a hierarchy of Java data objects that represent the source XML data -- and for marshalling Java content trees back into XML instance documents. The JAXB binding framework also provides methods for validating XML content as it is unmarshalled and marshalled.

What this means is that your Java applications can work with schema-compliant XML data as Java objects, leveraging native Java security models and programming methods. Moreover, Java developers do not need to be well-versed in the intricacies of SAX or DOM processing models, or even in the arcane language of XML schema, to take advantage of ubiquitous, platform-neutral XML technologies.


XML Schema
JAXB Binding Compiler
Binding Declarations
Derived Java classes and interfaces
Application
XML Input Documents (Unmarshal)
XML Output Documents (Marshal)

General JAXB Overview

Implementing JAXB in your Java applications is typically a two-step process:

1. One or more schemas are defined as needed, and then the JAXB binding compiler is run against these schemas to generate JAXB packages, classes, and interfaces.

2. Java application developers use the generated JAXB packages and the JAXB utility packages in a binding framework to unmarshal, marshal, and validate XML content.

Separating the binding process from the implementation process provides three significant benefits:

- A JAXB binding implementation is typically much smaller and more efficient than an analogous SAX or DOM implementation because the generated JAXB packages contain representations of only the XML schema directives actually used in the source schemas, rather than the entire XML schema language.

- A small team of XML schema developers can focus on writing schemas and generating JAXB packages, which can in turn be used by any number of Java developers for writing any number of Java applications that implement the JAXB binding framework. Not everyone needs to know XML, and the code for a single binding implementation can be shared among different applications.

- JAXB bindings can be customized to suit your particular application needs -- for example, you can customize package, interface, or property names to resolve naming conflicts, among many other customization options. This makes it possible to use one schema as the basis for any number of JAXB binding framework implementations, or to accommodate updates to a source schema without having to significantly alter the Java applications based on that schema.

What Problems Does JAXB Solve?

XML and Java technologies are ideally suited to the rapid creation and deployment of robust, platform-neutral, standards-based Web services and applications. That said, the primary goal of XML data binding is to make it easier to use Java applications to read, process, and output XML data.

"Easier" in this case means not having to worry about XML syntax -- or more, about things like tracking down errors caused by irregular use of white space, brackets, quotes and suchlike in source XML documents -- and not having to be versed in the complexities of the XML language.

XML provides flexible, extensible, platform-neutral formats and protocols for structuring and exchanging information. Its flexibility and extensibility are both its strength and weakness, because beyond general rules for well-formedness, XML documents can be structured in widely different ways. The problems come when different applications read and write XML documents using incompatible structures, and the data within those documents adhere to particular structures more or less stringently.

Part of the solution comes in the form of XML DTDs and W3C XML schemas; the former describing the elements, entities, and relationships that can be used in an XML document, and the latter providing even more rigorous constraint rules along similar lines. These constraints alone, however, are often not enough for Java developers trying to come up with robust applications in which datatyping, validation, and consistency are critical. By enforcing the use of schemas, and by enhancing XML schema's support for datatyping and validation, JAXB data binding enhances the inherent strength of the XML data model.

In addition to the general goals of XML-to-Java data binding, JAXB provides features to meet several important additional goals:

- ***Be easy to use***

  JAXB makes it easy to access and modify XML documents within Java programs without having to deal with the complexities of SAX or DOM. Even if you don't know much about XML, you can compile an XML schema and immediately compile and start using the classes generated by JAXB.

- ***Be customizable***

  JAXB provides a standard way to customize the binding of existing schema components to Java representations. Sophisticated applications sometimes require fine control over the structure and content of schema-derived classes, both for their own purposes and for keeping pace with schema evolution.

- ***Be portable***

  You can write applications implementing JAXB in such a way that the JAXB components can be replaced without having to make significant changes to the rest of the source code.

- ***Support validation on demand***

  When working with a content tree corresponding to an XML document, it is often necessary to validate the tree against the constraints in the source schema. With JAXB, it is possible to do this at any time, without the user having to first marshal the tree into XML.

- ***Provide clean "round-tripping"***

  Transforming a Java content tree to XML content and back to Java content again results in equivalent Java content trees before and after the transformation.

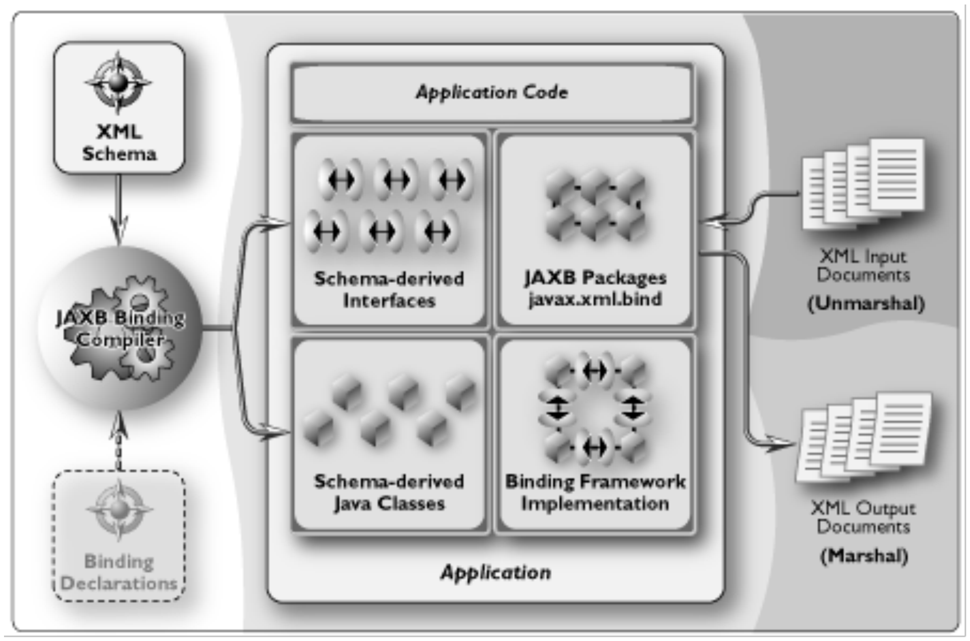
When Should I Use JAXB?

JAXB is particularly useful if you want to:

- Access configuration values from a properties file stored in XML format.
- Develop a tool that can create or modify a configuration properties file represented in XML format.
- Receive data in the form of an XML document, and then access and/or update the data without having to write SAX event handlers or traverse a DOM parse tree.
- Validate data input by a user; for example, from a form presented in a Web browser, where the form data is mapped to an XML document. In such cases, JAXB provides the capability to validate the accuracy of the data using the validation constraints of a schema that describes the data collected from the form.
- Bind an XML document into a Java representation, update the content via Java interfaces, validate these changes against the constraints within the original schema, and then write the updated Java representation back to an XML document.
- Unmarshal an XML document already known to be valid; validation usually performed by the application while unmarshalling the document can be disabled in this case, thereby greatly improving performance over SAX or DOM methods.

How Does JAXB Work?

Core Components


XML Schema
JAXB Binding Compiler
Binding Declarations
Application Code
Schema-derived Interfaces
JAXB Packages javax.xml.bind
Schema-derived Java Classes
Binding Framework Implementation
Application
XML Input Documents (Unmarshal)
XML Output Documents (Marshal)

Core JAXB Components

A JAXB implementation comprises eight core components:

1. *XML Schema*

   An XML schema uses XML syntax to describe the relationships among elements, attributes and entities in an XML document. The purpose of an XML schema is to define a class of XML documents that is, you may want to define separate schemas for chapter-oriented books, for an online purchase order system, or for a personnel database. An XML document that conforms to a particular schema is referred to as an instance document.

2. ***Binding Declarations***

   By default, the JAXB binding compiler binds Java classes and packages to a source XML schema based on rules defined in the JAXB Specification. In most cases, the default binding rules are sufficient to generate a robust set of schema-derived classes from a wide range of schemas. There may be times, however, when the default binding rules are not sufficient for your needs. JAXB supports customizations and overrides to the default binding rules by means of binding declarations made either inline as annotations in a source schema, or as statements in an external binding customization file that is passed to the JAXB binding compiler. Note that custom JAXB binding declarations also allow you to customize your generated JAXB classes beyond the XML-specific constraints in an XML schema to include Java-specific refinements such as class and package name mappings.

3. ***Binding Compiler***

   The JAXB binding compiler is the core of the JAXB processing model. Its function is to transform, or bind, a source XML schema to a set of JAXB content classes in the Java programming language. Basically, you run the JAXB binding compiler using an XML schema (optionally with custom binding declarations) as input, and the binding compiler generates Java classes that map to constraints in the source XML schema.

4. ***Binding Framework Implementation***

   The JAXB binding framework implementation is a runtime API that provides interfaces for unmarshalling, marshalling, and validating XML content in a Java application. The binding framework comprises interfaces in the `javax.xml.bind` package.

5. ***Schema-Derived Classes***

   These are the schema-derived classes generated by the binding JAXB compiler. The specific classes will vary depending on the input schema.

6. ***Java Application***

   In the context of JAXB, a Java application is a client application that uses the JAXB binding framework to unmarshal XML data, validate and modify Java content objects, and marshal Java content back to XML data. Typically, the JAXB binding framework is wrapped in a larger Java application that may provide UI features, XML transformation functions, data processing, or whatever else is desired.

7. ***XML Input Documents***

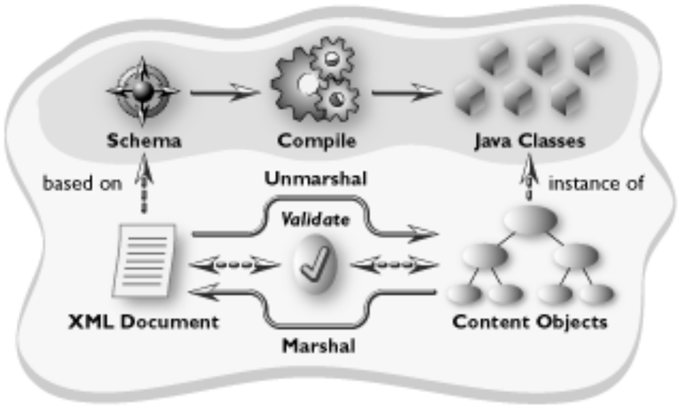
   XML content that is unmarshalled as input to the JAXB binding framework -- that is, an XML instance document, from which a Java representation in the form of a content tree is generated. In practice, the term *document* may not have the conventional meaning, as an XML instance document does not have to be a completely formed, selfstanding document file; it can instead take the form of streams of data passed between applications, or of sets of database fields, or of XML infosets, in which blocks of information contain just enough information to describe where they fit in the schema structure.

In JAXB, the unmarshalling process supports validation of the XML input document against the constraints defined in the source schema. This validation process is optional, however, and there may be cases in which you know by other means that an input document is valid and so you may choose for performance reasons to skip validation during unmarshalling. In any case, validation before (by means of a third-party application) or during unmarshalling is important, because it assures that an XML document generated during marshalling will also be valid with respect to the source schema.

8. ***XML Output Documents***

   XML content that is marshalled out to an XML document. In JAXB, marshalling involves parsing an XML content object tree and writing out an XML document that is an accurate representation of the original XML document, and is valid with respect the source schema. JAXB can marshal XML data to XML documents, SAX content handlers, and DOM nodes.

The JAXB Binding Process



Steps in the JAXB Binding Process

There are seven general steps in the JAXB data binding process:

1. ***Generate Classes***

   An XML schema is used as input to the JAXB binding compiler to generate source code for JAXB classes based on that schema.

2. ***Compile Classes***

   All of the generated classes, source files, and application code must be compiled.

3. ***Unmarshal***

   XML documents written according to the constraints in the source schema are unmarshalled by the JAXB binding framework. Note that JAXB also supports unmarshalling XML data from sources other than files/documents, such as DOM nodes, string buffers, SAX Sources, and so forth.

4. ***Generate Content Tree***

   The unmarshalling process generates a content tree of data objects instantiated from the generated JAXB classes; this content tree represents the structure and content of the source XML documents.

5. ***Validate (optional)***

The unmarshalling process optionally involves validation of the source XML documents before generating the content tree. Note that if you modify the content tree in Step 6, below, you can also use the JAXB Validate operation to validate the changes before marshalling the content back to an XML document.

6. ***Process Content***

   The client application can modify the XML data represented by the Java content tree by means of interfaces generated by the binding compiler.

7. ***Marshal***

   The processed content tree is marshalled out to one or more XML output documents. The content may be validated before marshalling.

To synopsize, using JAXB involves two discrete sets of activities:

- Generate and compile JAXB classes from a source schema, and build an application that implements these classes
- Run the application to unmarshal, process, validate, and marshal XML content through the JAXB binding framework

These two steps are usually performed at separate times in two distinct phases. Typically, for example, there is an application development phase in which JAXB classes are generated and compiled, and a binding implementation is built, followed by a deployment phase in which the compiled JAXB classes are used to process XML content in an ongoing "live" production setting.

How Does JAXB Represent XML Content?

Binding XML Names to Java Identifiers

XML schema languages use XML names strings that match the Name production defined in XML 1.0 (Second Edition) (http://www.w3.org/XML/) to label schema components. This set of strings is much larger than the set of valid Java class, method, and constant identifiers. To resolve this discrepancy, JAXB uses several name-mapping algorithms.

The JAXB name-mapping algorithm maps XML names to Java identifiers in a way that adheres to standard Java API design guidelines, generates identifiers that retain obvious connections to the corresponding schema, and is unlikely to result in many collisions.

Java Representation of XML Schema

JAXB supports the grouping of generated classes and interfaces in Java packages. A package comprises:

- A name, which is either derived directly from the XML namespace URI, or specified by a binding customization of the XML namespace URI
- A set of Java content interfaces representing the content models declared within the schema

- A set of Java element interfaces representing element declarations occurring within the schema
- An `ObjectFactory` class containing an instance factory method for each Java content interface and Java element interface in the package, and a dynamic instance factory allocator, which creates an instance of the specified Java content interface
- A set of typesafe enum classes
- javadoc for the package

Can I Customize JAXB Bindings?

The default JAXB bindings can be overridden at a global scope or on a case-by-case basis as needed by using custom binding declarations. As described previously, JAXB uses default binding rules that can be customized by means of binding declarations made in either of two ways:

- As inline annotations in a source XML schema
- As declarations in an external binding customizations file that is passed to the JAXB binding compiler

Custom JAXB binding declarations also allow you to customize your generated JAXB classes beyond the XML-specific constraints in an XML schema to include Java-specific refinements such as class and package name mappings.

You do not need to provide a binding instruction for every declaration in your schema to generate Java classes. For example, the binding compiler uses a general name-mapping algorithm to bind XML names to names that are acceptable in the Java programming language. However, if you want to use a different naming scheme for your classes, you can specify custom binding declarations to make the binding compiler generate different names. There are many other customizations you can make with the binding declaration, including:

- Name the package, derived classes, and methods
- Assign types to the methods within the derived classes
- Choose which elements to bind to classes
- Decide how to bind each attribute and element declaration to a property in the appropriate content class
- Choose the type of each attribute-value or content specification

---

How Do I Get JAXB?

A JAXB Reference Implementation (JAXB RI) is included in the latest version (1.1) of the Java^{TM} Web Services Developer Pack (JAVA WSDP 1.1). The Java WSDP is a free, integrated toolkit that allows Java developers to build, test and deploy XML applications, Web services, and Web applications.

The JAXB RI includes binary code, a user's guide, the JAXB specification, and a rich set of sample applications and sample source code. In addition to the JAXB RI, you may want to download the Java Web Service Tutorial, which contains detailed instructions for using JAXB and all the other technologies in the Java WSDP. Finally, please visit the JAXB home page for the latest JAXB project information and specifications.

JAXB fills an important need for Java developers wishing to manipulate XML data in their applications without having to wrestle with XML itself. By merging ubiquitous, platform-neutral XML technologies with the security, portability, and convenience of write-once-run-anywhere Java technologies, JAXB provides a key tool for the rapid and efficient development of Web services and network-based applications.
