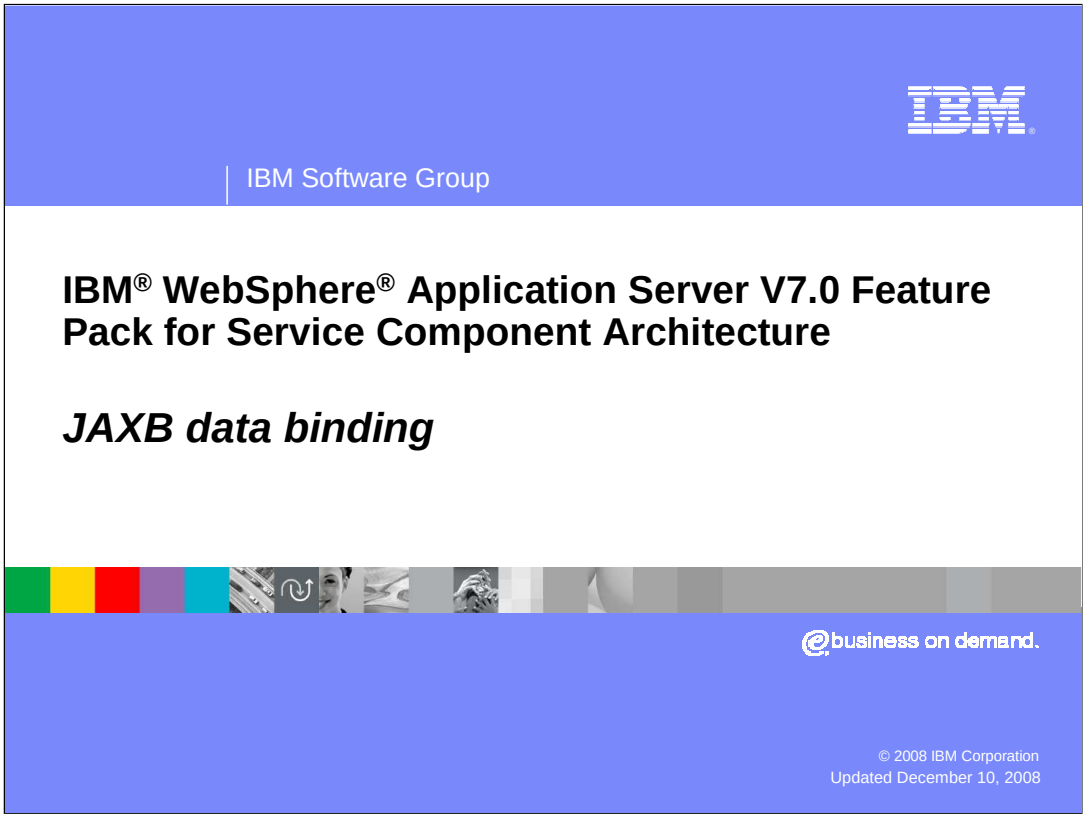


This presentation will cover the data binding piece of SCA; specifically JAXB, which is what is implemented in the SCA feature pack.

Agenda

- Overview
- JAXB in SCA feature pack
- Summary and references

This presentation starts with a small overview of the data bindings in general, followed by the discussion of the implementation of JAXB in the SCA feature pack.

Section

Overview

This first section is the overview.

Data bindings: Overview

- **Q. In a "data binding" what is being bound?**
- A. Application data format (that is the Java™ programming model) is being bound to the wire data format used by the runtime.
- **Q. Why?**
- A. More Java-friendly programming model to the application programmer using a data binding.

In a data binding, what is being bound? The answer to this question is that Application data format (that is the Java programming model) is being bound to the wire data format used by the runtime. And why is this done? Although XML is useful in SOA for bridging a variety of technologies, platforms, and implementation types, it can be a burden to program to. And although the SCA runtime in some cases uses an XML wire data format, data binding provides a more Java-friendly programming model to the application programmer.

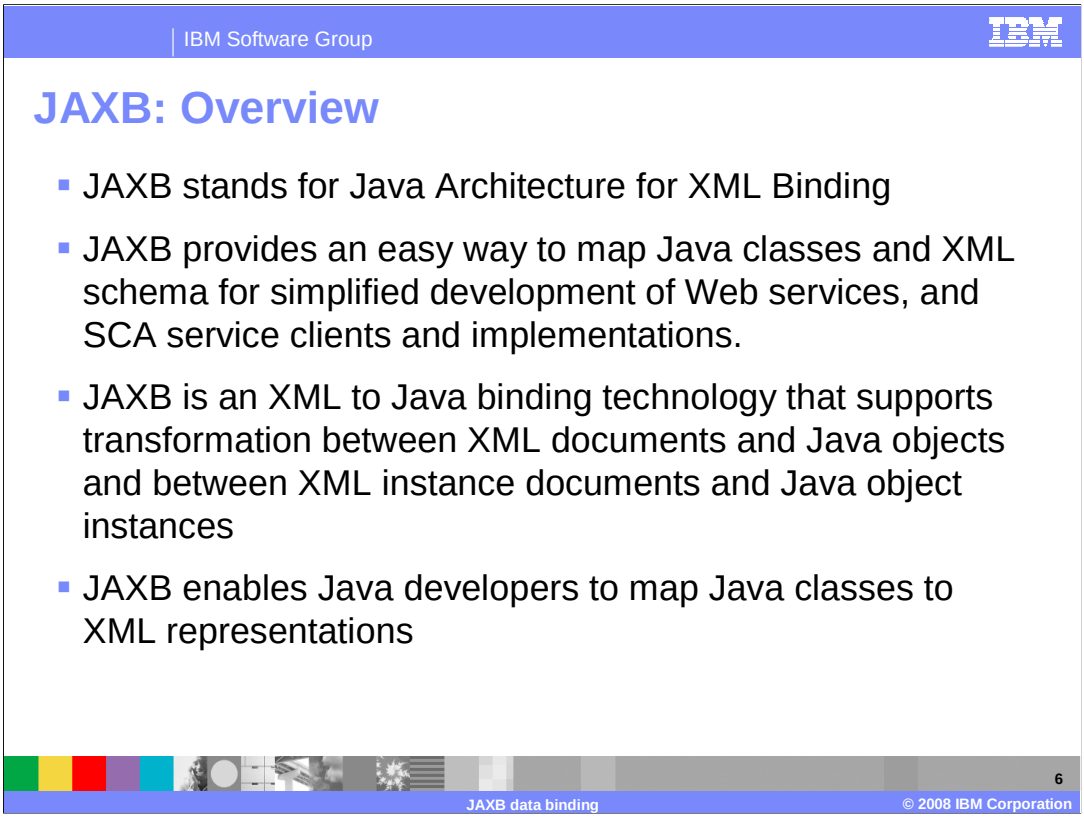
Data bindings: Overview

- Two data bindings relevant to SCA feature pack:
  - SDO – Dynamic or static (generated) API
  - JAXB – Static API
- For SCA feature pack, JAXB with its static API is the only one supported
- ** The JAX-WS programming model in WebSphere Application Server also uses JAXB

There are two data bindings relevant in this discussion of the SCA Feature Pack. JAXB and service data objects, or SDOs.

SDO - the Dynamic or Static (generated) API mentioned in the OSOA SCA specifications - is not supported in the SCA feature pack. It was supported in SCA Feature Pack alpha on WebSphere Application Server 6.1, but it is not supported in the current release of the SCA feature pack.

For SCA feature pack, JAXB with its static API is the only one supported. Something to note is that the JAX-WS programming model in WebSphere Application Server also uses JAXB and the SCA feature pack is comparable.

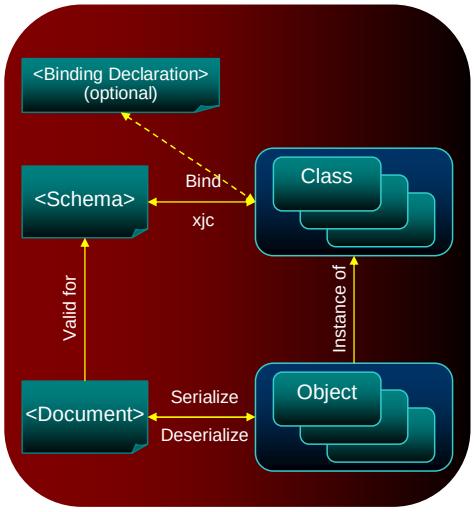


JAXB stands for Java Architecture for XML Binding

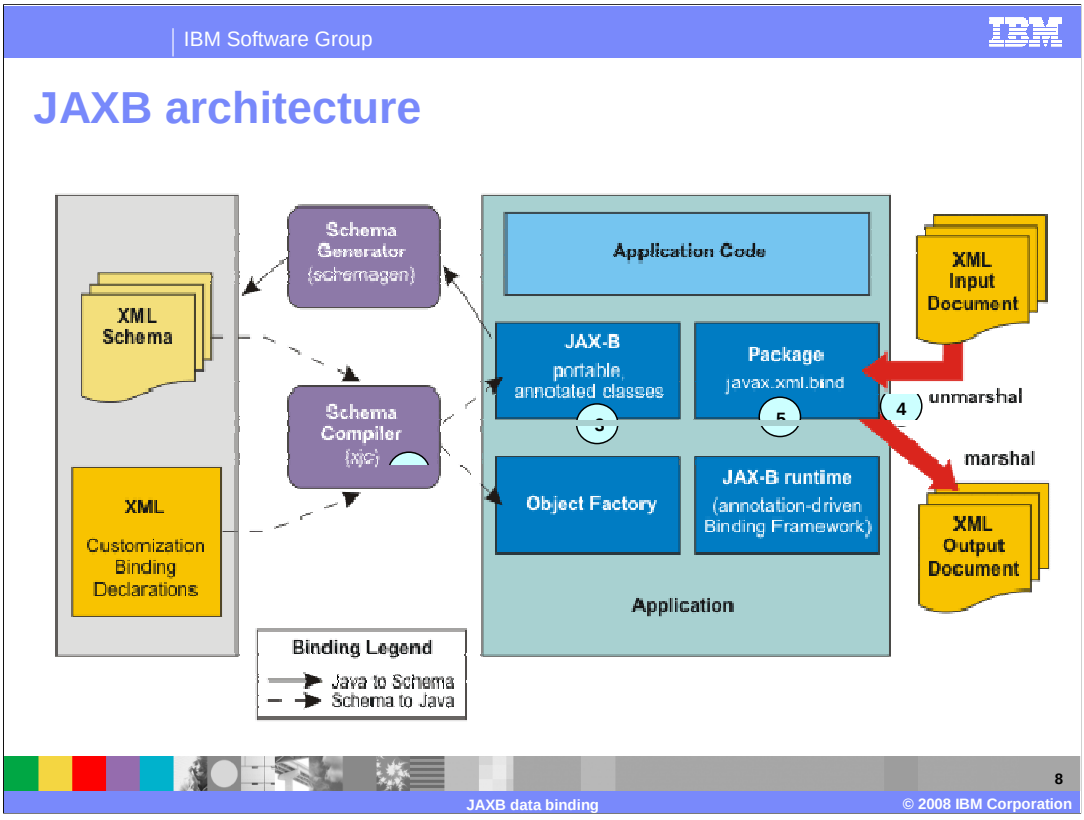
It is a Java technology that lets you generate Java classes from XML schemas by means of a JAXB binding compiler. The JAXB binding compiler takes XML schemas as input, and then generates a package of Java classes and interfaces that reflect the rules defined in the source schema. These generated classes and interfaces are in turn compiled and combined with a set of common JAXB utility packages to provide a JAXB binding framework. JAXB is a Java technology that provides an easy and convenient way to map Java classes and XML schema for simplified development of Web services and SCA service clients and implementations. JAXB uses the flexibility of platform-neutral XML data in Java applications to bind XML schema to Java applications without requiring extensive knowledge of XML programming. JAXB is an XML to Java binding technology that supports transformation between XML documents and Java objects and between XML instance documents and Java object instances. JAXB enables Java developers to map Java classes to XML representations. JAXB allows storing and retrieving data in memory in any XML format, without the need to implement a specific set of XML loading and saving routines for the program's class structure. So application developers can focus on business need to define the XML schema for the data that flows between the services and the client, and interact with the business data in the JAVA POJO form.

JAXB

- Maps XML Schema to Java:
  - Schema Compiler
  - XML Binding Declarations
- Maps Java to XML Schema:
  - Schema Generator
  - J2SE 5.0 Annotations
- Runtime:
  - Serialization / deserialization
  - Binding



The JAXB binding framework provides methods for unmarshalling XML instance documents into Java content trees. This is a hierarchy of Java data objects that represent the source XML data and for marshalling Java content trees back into XML instance documents.



Here is a picture of JAXB architecture.

JAXB provides the **xjc** schema compiler tool shown in 1, it also provides 2) the **schemagen** schema generator tool, and a runtime framework.

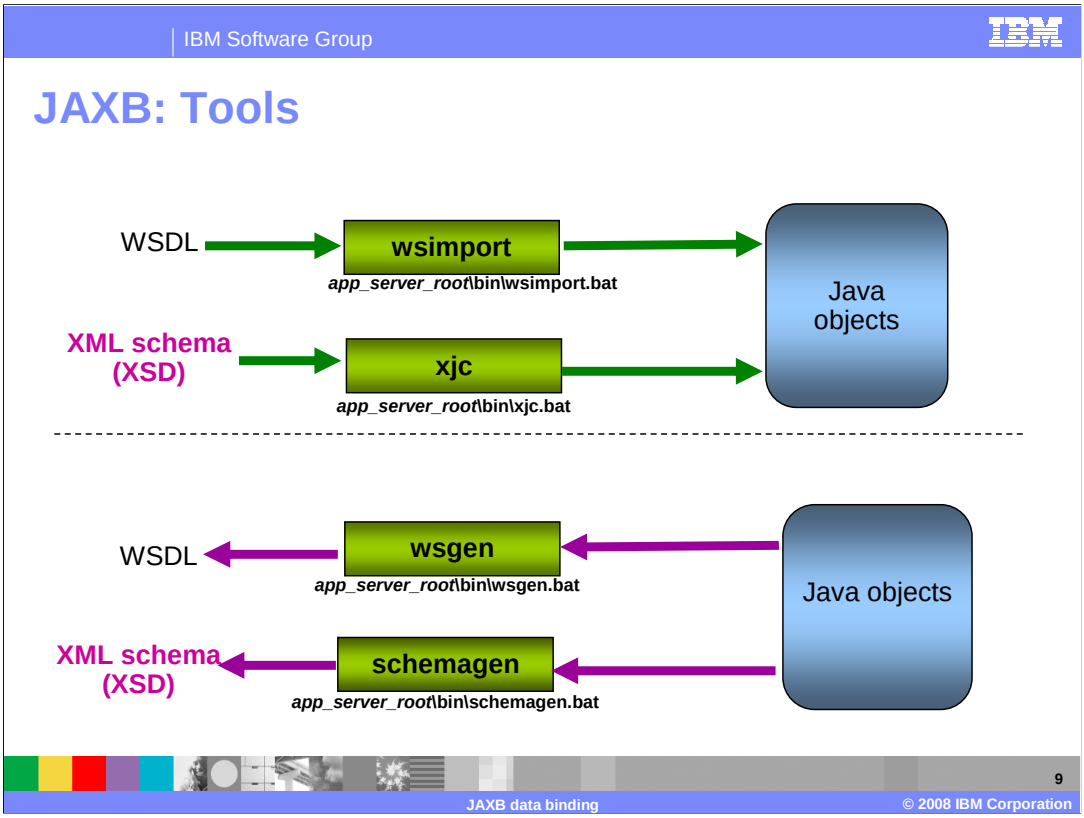
Number 3 denotes JAXB annotated classes and artifacts. These contain all the information needed by the JAXB runtime API to process XML instance documents.

At 4, the JAXB runtime API supports marshaling of JAXB objects to XML and unmarshalling the XML document back to JAXB class instances. Optionally, you can use JAXB to provide XML validation to enforce both incoming and outgoing XML documents to conform to the XML constraints defined within the XML schema.

5 is the JAXB binding package, **javax.xml.bind**, which defines the abstract classes and interfaces that are used directly with content classes. In addition the package defines the marshal and unmarshal APIs.

** In general, JAXB is the default data binding technology used by the Java API for XML Web Services (JAX-WS) tools and implementation within this product. You can develop JAXB objects for use within JAX-WS and SCA applications.

You can also use JAXB independent of JAX-WS when you want to take advantage of the XML data binding technology to manipulate XML within your Java applications.



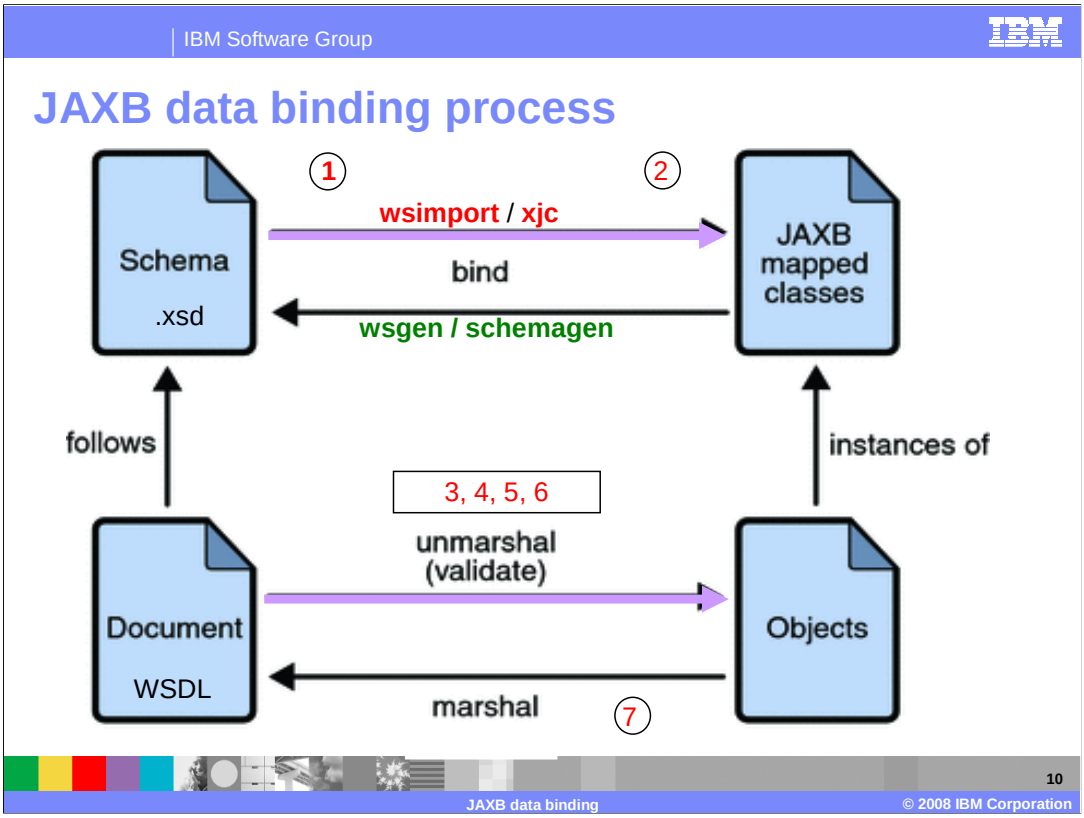
JAXB provides these four tools: wsimport, xjc, wsgen and schemagen. You can run them from root of WebSphere Application Server install\bin directory.

The **wsimport** tool reads an existing WSDL file and generates a Java interface and required data beans to model the service data.

The **xjc** tool is the JAXB schema compiler. It generates fully annotated Java classes from an XML schema.

The **wsgen** tool creates WSDL documents from Java beans.

The **schemagen** tool is the schema generator. **schemagen** creates the XML schema from Java beans. Note that data stored in XML documents can be accessed without the need of understanding the data structure.



Here is a look at the data binding process:

At label 1, the first step is to generate classes. Suppose you start with an XML schema (xsd file). In this scenario, the XML schema is used as an input to the JAXB binding compiler (xjc) to generate JAXB classes based on that schema. At label 2, all of the generated classes, source files, and application code are then compiled.

Next step, shown at 3, is to Unmarshal.

XML documents (WSDL) 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.

At 4, The next step is the generation of the 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.

At five and six, there is also the ability to process content at this point. This means that the client application can modify the XML data represented by the Java content tree by means of interfaces generated by the binding compiler. This completes the one way process of the binding.

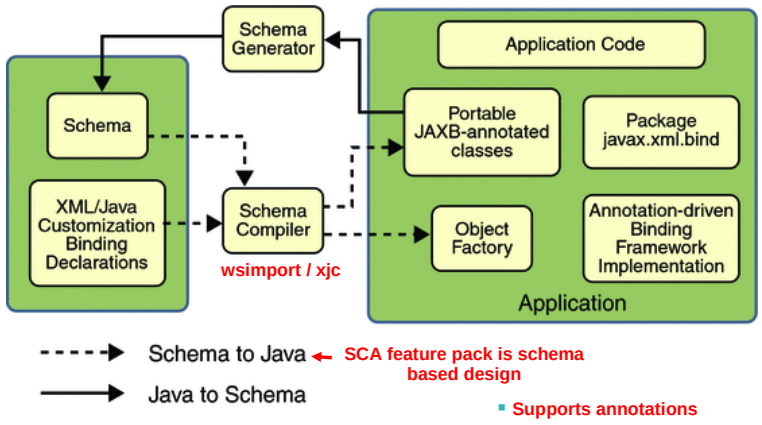
Note that as shown at 7, the processed content tree may be marshalled out to one or more XML output documents.

Section

JAXB in SCA

Now that you know what JAXB is in general, this section looks at how it applies in SCA.

JAXB in SCA feature pack



For SCA feature pack, schema based interface design is the first class principle.

The implementation starts with schema definitions (WSDL/XSD) then Java interface is generated with JAXB then the implementation is written.

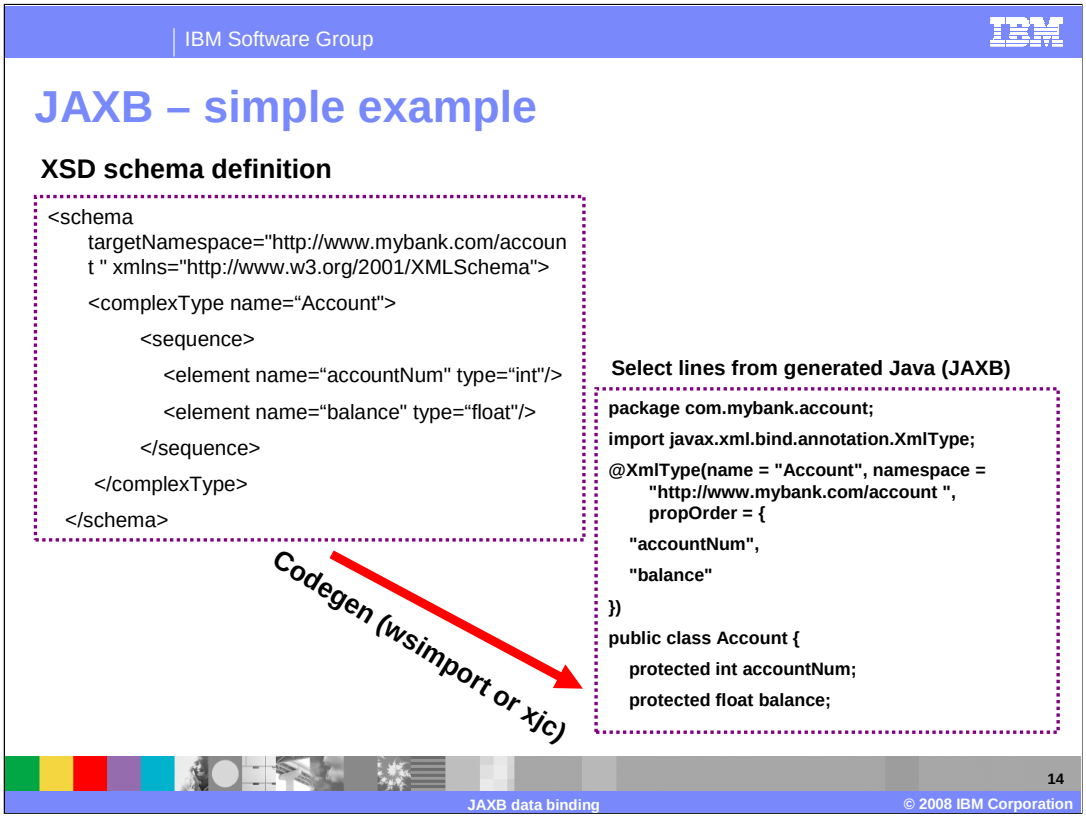
JAXB in SCA: Overview

- The WSDL/XSD used by the SCA runtime to define the wire format of the data and maps to:
  - The Java interface defined in terms of the Java classes
- XML document is the wire format of the data on a single service invocation and maps to:
  - Actual Java parameters (input/output arguments)

JAXB is used for conversion of WSDL/XSD to Java Classes.

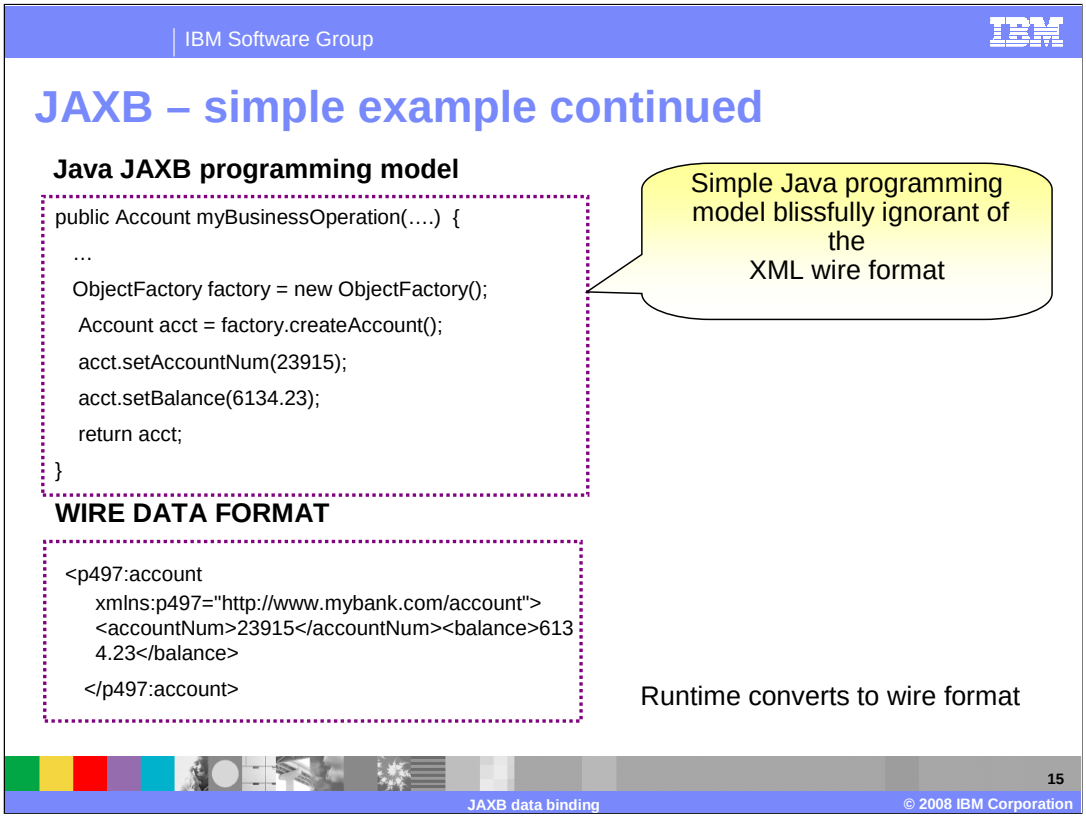
The WSDL/XSD service interface definition is used by the SCA runtime to define the wire format of the data and this maps to the Java interface defined in terms of the Java classes it uses for its methods' input, output, and exception types

The XML document is the wire format of the data on a single service invocation and this maps to the actual Java parameters (input and output arguments) passed across a single service invocation.



Here is a simple example of XSD schema to Java code using JAXB wsimport or xjc.

You basically start with an XSD schema, run it through the JAXB xjc tool and you end up with Java code.



The JAXB binding framework provides methods for unmarshalling XML instance documents into Java content trees. This is 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 (deserialized) and marshalled (serialized).

JAXB in SCA – when does it come into play?

- For clients and implementations of remotable interfaces when the default (SCA) and Web service bindings are used
  - Specifically bindings that use an XML wire format.
    - * EJB binding works differently since it uses Java serialization.
  - A client or implementation of a local interface does not use JAXB, since pass-by-reference is done over a local interface rather than a data copy.

JAXB comes into play for clients and implementations of remotable interfaces when the default (SCA) and Web service bindings are used. These are the bindings that use an XML wire format. The EJB binding works differently since it uses Java serialization.

A client or implementation of a local interface does not have to worry about JAXB, since pass-by-reference is done over a local interface. Any Java type may be used over a local interface.

As a caution for the EJB binding case, this is bad since the abstraction of a binding-neutral programming model has broken down. To the extent that the EJB binding is used only to access pre-existing EJBs with their types, this is not a huge problem.

JAXB – Top down

- Best practice is to start from WSDL/XSD definitions
- Code generation - wsimport
  - used to generate Java classes
- JAXB classes generated from schema definitions in WSDL <types> section
  - marked up using annotations from the javax.xml.bind.annotation.* namespace
- Java interface generated from WSDL port Type with @javax.jws.WebService annotation
- JAX-WS/JAXB customizations can be used to customize generated classes

The best practice is to start from WSDL/XSD definitions, which are the most interoperable - they are platform-neutral and language-neutral.

Code generation is through the wsimport tool that is used to generate Java classes to be used in your Java client or implementation from the WSDL/XSD definitions.

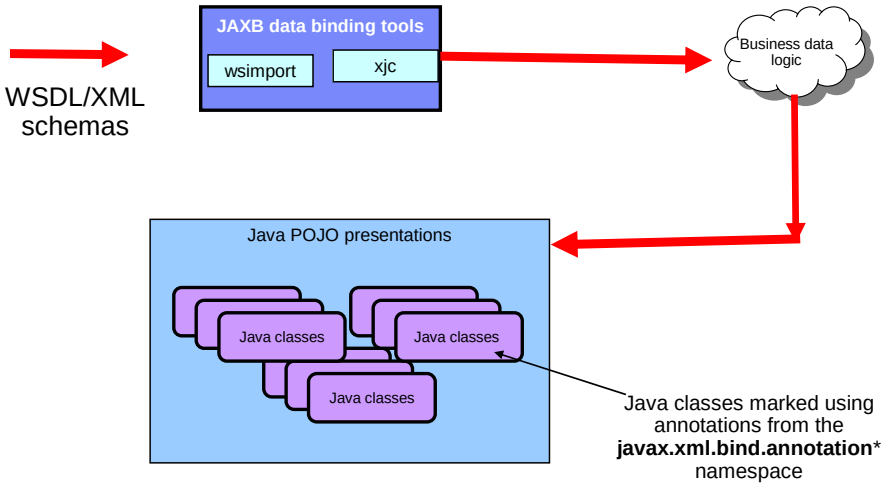
JAXB classes generated from schema definitions in the WSDL <types> section, and are marked up using annotations from the javax.xml.bind.annotation.* namespace

Java interface is generated from WSDL portType with @javax.jws.WebService annotation.

JAX-WS/JAXB customizations can be applied to generated classes by overriding or extending the default mappings and bindings.

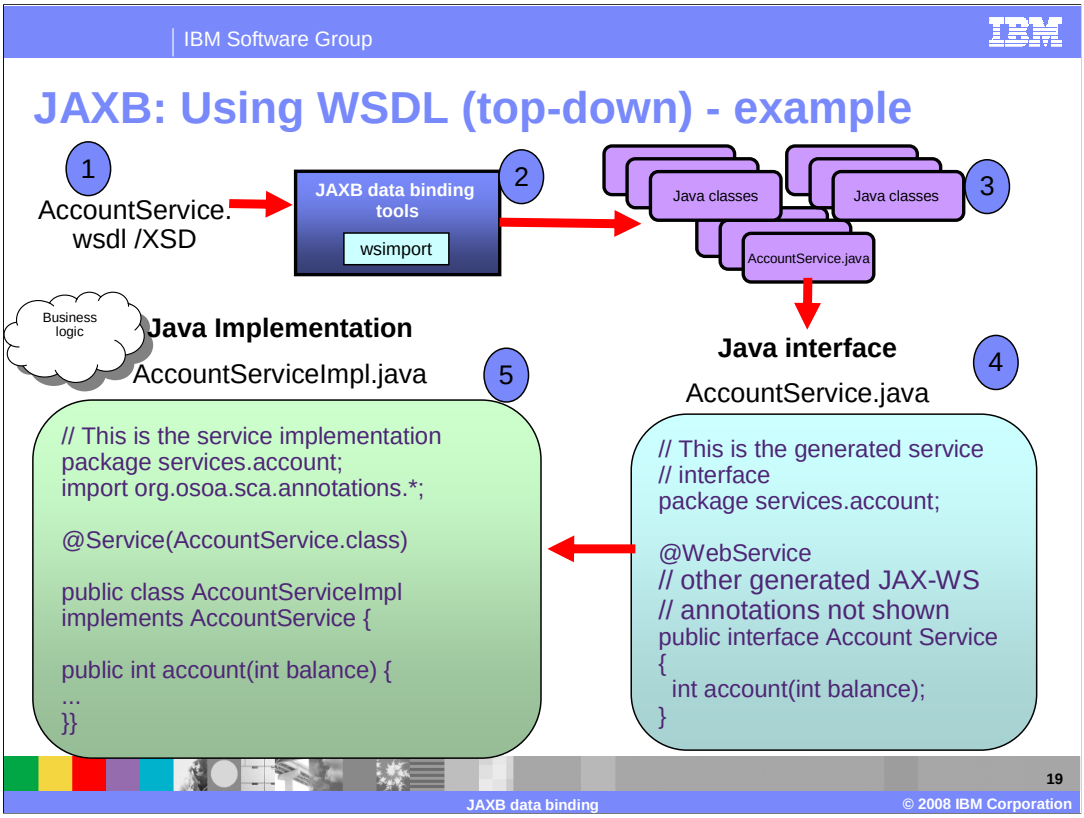
JAXB data binding tools: Top down scenario

- Top down scenario:



As mentioned in the previous slide, for the top-down scenario, wsimport and xjc are used to define the business data logic from the XML schema, then generate the Java representations. The classes generated are marked up using annotations from the **javax.xml.bind.annotation.* namespace**.

This is the implementation mostly used in the SCA feature pack.



This is a visual diagram example of a top-down scenario using JAXB mentioned in the previous slide.

First run the WSDL/XSD (in this example AccountService.wsdl) through **wsimport** to generate Java classes. These classes can then be used in your Java client or implementation from the WSDL/XSD definitions.

JAXB top-down marshalling/unmarshalling

- All schema-defined data is mapped to Java data
- JAXB annotations are generated to capture in Java the schema definitions that the Java is mapped from.

In the top-down case it is much simpler to understand what data gets marshalled.

All schema-defined data is mapped to Java data and JAXB annotations are generated to capture in Java the schema definitions that the Java is mapped from.

JAXB – Bottom-up

- Bottom-up is Java-centric
- JAXB marshalling used whether you generate JAXBs or not
  - The JAXB spec defines a mapping of unannotated POJO to XML.
  - JAXB mapping used even if no JAXB classes are generated, and even if no JAXB annotations are present.

The bottom-up, Java-centric view is designed to enable the Java programmer to just get started with a minimal effort. However, the SCA Java programming model is not the Java programming model – it is a JAXB-based programming model.

JAXB marshalling used whether you generate JAXBs or not. The JAXB specification defines a mapping of an annotated POJO to XML.

JAXB mapping is used even if no JAXB classes are generated, and even if no JAXB annotations are present.

JAXB bottom up scenario - No tools required

- The runtime will generate the needed XSD/WSDL definitions for you, and so there is no need to run schemagen or wsgen during development time.
- This is the simplest path to application development and deployment.

For a simpler path to application development and deployment, the runtime will generate the needed XSD/WSDL definitions for you, and so there is no need to run schemagen or wsgen during development time.

JAXB bottom up scenario - Dev tool

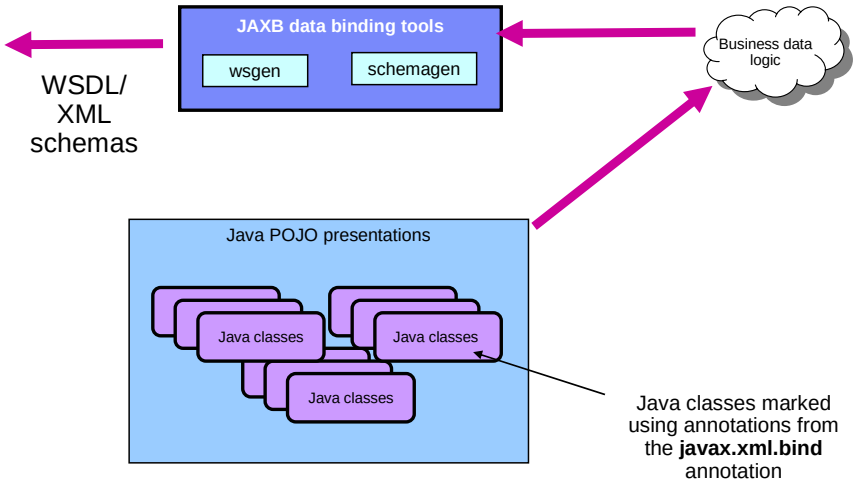
- You can choose to generate the XSD/WSDL yourself at development time (shown on next slide) for use within your application
  - Example: to refer to it in SCDL on an interface or binding definition
- This allows you to clearly see the WSDL/XSD used for your service interface
  - But it is no longer as simple
  - It might be better to restart with a top-down development approach after generating the WSDL

You can choose to generate the XSD/WSDL yourself at development time (shown on next slide) for use within your application. This allows you to clearly see the WSDL/XSD that is used for your service interface.

But it is no longer as simple; it might be better to restart with a top-down development approach after generating the WSDL.

JAXB data binding tools: Bottom up scenario

- Bottom up scenario:



It is possible to use tools to generate XSD/WSDL from your Java in the bottom-up case so that you can clearly capture the XSD/WSDL interface definition, which would otherwise be generated from the runtime.

The schemagen and wsgen tools allow you to generate XSD and WSDL definitions. This slide shows a visual example of how this process would happen.

JAXB bottom-up marshalling/unmarshalling

- JAXB marshalling/unmarshalling (that is serialization/deserialization) – understand what data gets preserved across your invocation
- For an unannotated class,
  - JavaBean properties are serialized
  - Private data is not

**** This is different from Java serialization which is used by other WebSphere Application Server programming models (example: RMI/EJB).**

You must understand JAXB marshalling and unmarshalling in order to understand what data gets preserved across your invocation. For an unannotated class, JavaBean properties are serialized. Private data is not.

This is different from Java serialization, which is used by other WebSphere Application Server programming models like RMI/EJB.

Bottom-up: marshalling/unmarshalling example

```
public class MyUnAnnotatedClass  {
     public MyUnAnnotatedClass()   // no-arg, default constructor required by JAXB
       private int  count;        // Java-serializable, BUT not marshalled by JAXB
       public String name;     // Java-serializable, ALSO marshalled by JAXB
       private String accountName;   // Java-serializable, ALSO marshalled by JAXB
       public String getAccountName()
       public void setAccountName(String)
      …
  }
```

*****

‘count’ not marshalled per JAXB since it is private data
‘name’ marshalled per JAXB since it is public data
‘accountName’ marshalled per JAXB since it has public getter/setter

Here is an example of JAXB Bottom-up approach on marshalling and unmarshalling.

Notice that:

- ‘count’ is not marshalled per JAXB since it is private data

- ‘name’ is marshalled per JAXB since it is public data

- ‘accountName’ is marshalled per JAXB since it has public getter/setter

The default marshalling could be described as using the JavaBeans view of the class.

JAXB in SCA – A note on validation

- Validation against schema can not currently be done against the payload of a service invocation
- Applications can use JAXB APIs to validate their own XML documents

Validation against schema cannot currently be done against the payload of a service invocation. Applications can use JAXB APIs to validate their own XML documents.

Benefits of JAXB

- Simplifies access to an XML document from a Java program
- JAXB allows you to access and process XML data without having to know XML or XML processing
- Allows for accessing data in non-sequential order
  - does not force navigation through a tree to access the data
- By unmarshalling XML data through JAXB, Java content objects that represent the content and organization of the data are directly available to your program
- Uses memory efficiently
- Flexible - you can unmarshal XML data from a variety of input sources

(1) JAXB simplifies access to an XML document from a Java program:.

(2) JAXB allows you to access and process XML data without having to know XML or XML processing. Unlike SAX-based processing, there is no need to create a SAX parser or write callback methods.

(3) JAXB allows you to access data in non-sequential order, but unlike DOM-based processing, it does not force you to navigate through a tree to access the data.

(4) By unmarshalling XML data through JAXB, Java content objects that represent the content and organization of the data are directly available to your program.

(5) JAXB uses memory efficiently: The tree of content objects produced through JAXB tends to be more efficient in terms of memory use than DOM-based trees.

(5) JAXB is flexible:

You can unmarshal XML data from a variety of input sources, including a file, an InputStream object, a URL, a DOM node, or a transformed source object.

You can marshal a content tree to a variety of output targets, including an XML file, an OutputStream object, a DOM node, or a transformed data object

You can unmarshal SAX events -- for example, you can do a SAX parse of a document and then pass the events to JAXB for unmarshalling.

JAXB allows you to access XML data without having to unmarshal it. Once a schema is bound you can use the ObjectFactory methods to create the objects and then use set methods in the generated objects to create content.

(6) JAXB's binding behavior can be customized in a variety of ways.

Section

Summary

And, in summary…

Summary

- JAXB:
  - Easy and convenient way to map Java classes and XML schema for simplified SCA Java development
  - Provides
    - a schema compiler
    - schema generator
    - a runtime framework to support two-way mapping between Java objects and XML documents
  - Supports transformation
    - between schema and Java objects
    - between XML instance documents and Java object instances

JAXB provides an easy and convenient way to map Java classes and XML schema for simplified SCA Java development.

JAXB provides a schema compiler, schema generator and a runtime framework to support two-way mapping between Java objects and XML documents.

JAXB supports transformation between schema and Java objects and between XML instance documents and Java object instances.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WASv7SCA_JAXBDatabinding.ppt

This module is also available in PDF format at: ../WASv7SCA_JAXBDatabinding.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers