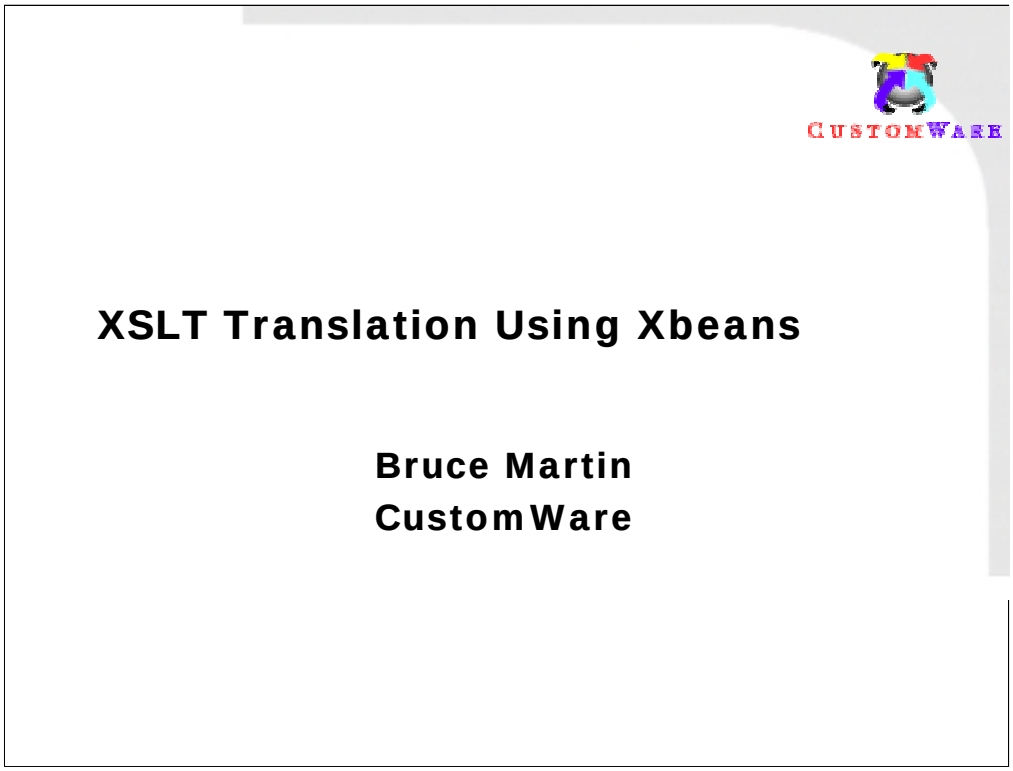


Welcome to my Xbeans presentation. I am Bruce Martin, from CustomWare.

Today I am going to be talking about the Xbean model for distributed, data flow applications.

I am going to focus on the translator Xbean that supports XSLT based translation.

Outline

- **XML and distributed applications**
- **Xbeans**
  - defined
  - Xbean channels
  - Xbeans as Java Beans
- **Example Xbeans**
- **The Translator Xbean**
- **Xbeans.org**

I'll begin by motivating Xbeans with XML and distributed applications.

I'll define Xbeans, discuss how they can be linked into channels and then talk about them as software components.

Then, I'll describe a number of example Xbeans and a sample application.

Finally, we will get into to the main topic of the talk: doing XSLT based translation with the translator Xbean.

I will end discussing the Xbeans.org open source project and talk about the upcoming second release of the code.

My favorite XML question:

From the jGuru XML FAQ:

Why would XML be a better solution than a typical delimited text file for B2B, or representing data in general? I can take a flat file, I can parse it and extract data, and then present that data in different ways using Java such as HTML or WML. So, why would I want to complicate things with DTDs, parsers, and APIs needed to extract and display data?

I manage the jGuru XML FAQ. This “devil’s advocate” question arrived from a jGuru member. It’s a very good question. Anybody who is promoting XML technology had better have a very good answer to this question.

XML

- **Portable, vendor neutral, readable data format.**
- **Lingua franca for data exchange**
  - Standards groups representing almost every human endeavor are agreeing upon DTDs for exchanging data.

The Extensible Markup Language, or XML, has emerged as the universal standard for exchanging and externalizing data. Software products of all kinds are being upgraded to "support XML." Typically this means they can import and export XML data.

At the same time, standards groups representing almost every human endeavor are agreeing upon XML Document Type Definitions (DTDs) for exchanging data. One of many examples is the International Press Telecommunications Council; it has defined an XML DTD allowing "news information to be transferred with markup and be easily transformed into an electronically publishable format." These vertical market standards will allow diverse applications to exchange data in unforeseen ways.

Distributed Applications

- **XML data need to be *processed*, *communicated* and *shared*.**
- **An application that communicates and processes XML between computers is in fact, a *distributed application*.**
- **Distributed applications are built out of parts.**

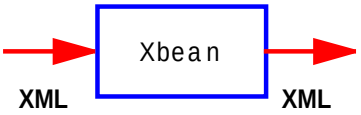
But just defining standard representations for exchanging data is insufficient. The data need to be integrated with existing applications and databases and processed by programs written in some programming language.

As soon as we communicate the XML data from one enterprise to another or even just one computer to another, we have in fact a distributed application. Distributed applications are tricky beasts.

Rather than structuring software that manipulates XML data as mammoth programs, *software component* technology allows developers to package smaller grained pieces of reusable functionality. *Java Beans* are software components that support the packaging, reuse, connection and customization of Java code. Design tools allow applications to be created by connecting and customizing existing Java Beans.

Xbean

- **Java Bean that consumes XML, processes it and passes it on.**



- **Inspired by IBM's XML Productivity Kit for Java:**
  - Distributed application focus
  - Open source project

So Xbeans are a good component technology for distributed applications that process XML in a data flow fashion. But rather than invent yet another component technology, Xbeans are Java Beans that process XML in a simple, standard fashion. (We will see the simple interfaces in a moment…)

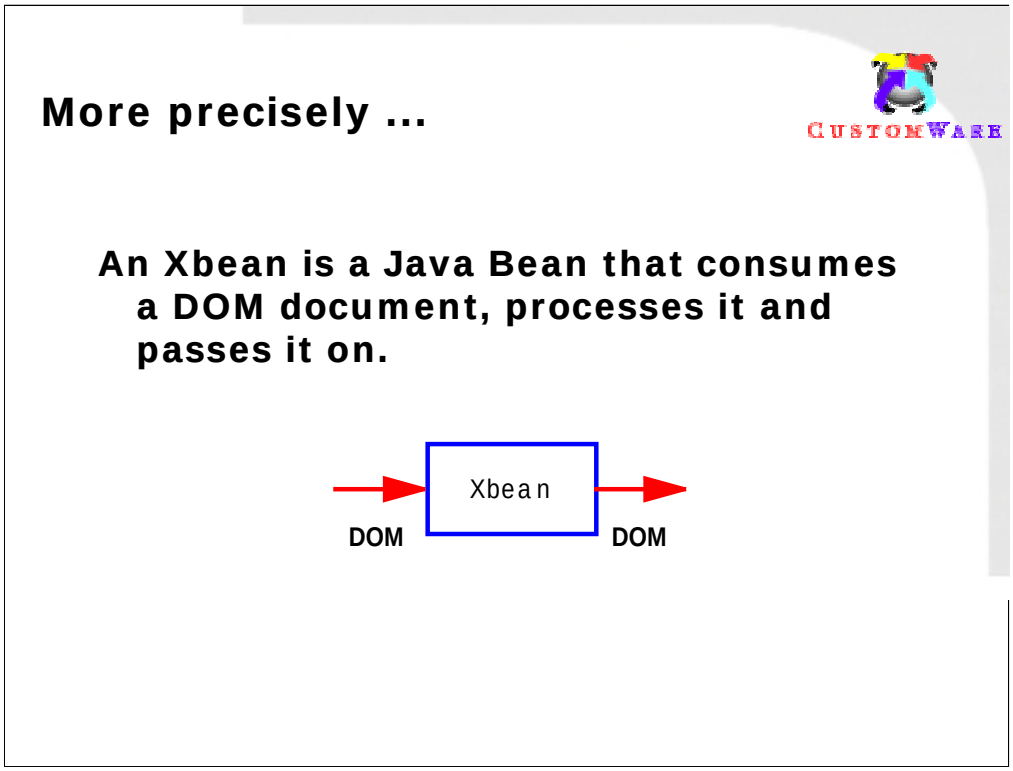
So what is an Xbean? An Xbean is a Java Bean that consumes XML as input, processes it in some fashion and then produces XML as output.

Document Object Model

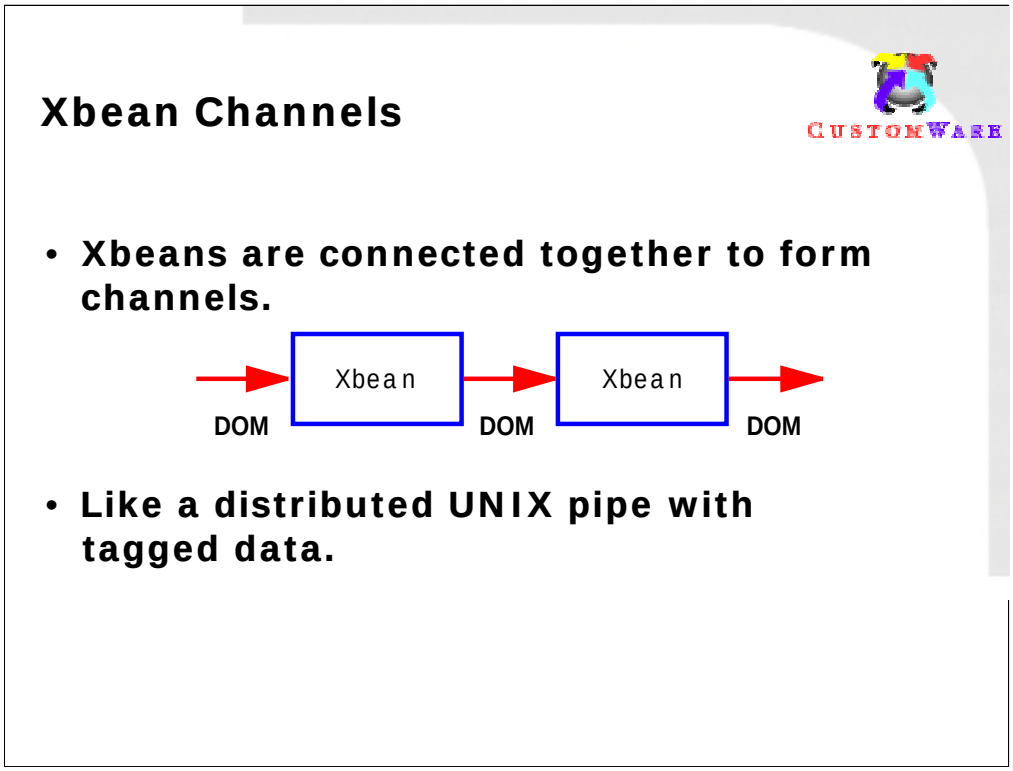
- Undesirable if every object processing XML had to parse it.
- W3C defined the Document Object Model (DOM).
- Standard application programmer's interface to XML data.
- Most XML parsers produce a DOM representation of the parsed XML.

To the end of accessing parsed XML data from different programming languages, the W3C has defined the Document Object Model (DOM) standard. The DOM is an application programmer's interface to XML data. It is available from many programming languages, including Java. Thus, Java programs can access XML data via the DOM API.

XML parsers produce a DOM representation of the XML document.



Xbeans consume and produce XML as DOM documents. That is, the data passed to Xbeans are not strings that need to be parsed by an XML parser, but an already parsed document object that is accessed via the w3c standard DOM API. As such, the previous graphic of an Xbean is not precise. This graphic shows the Xbean processing more precisely.

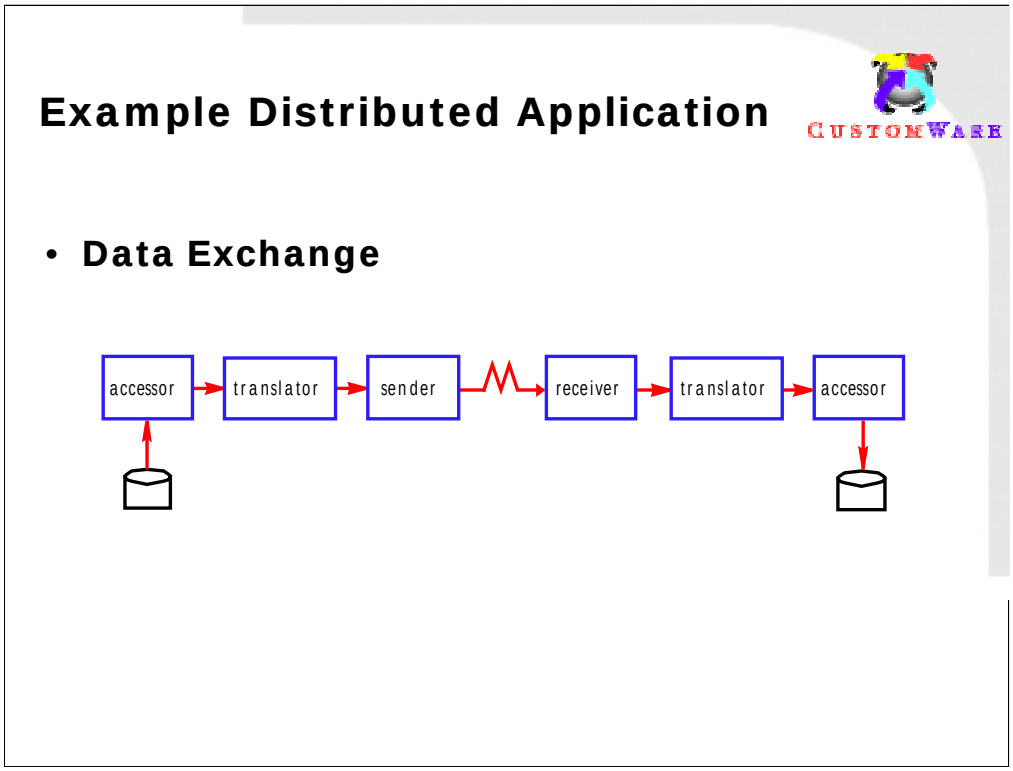


Xbeans are connected to each other forming a *channel*. Data flow from one bean to another in a channel. This is very similar to a UNIX pipe; typed XML data flow, rather than untyped bytes. Xbeans are also similar to CORBA event channels.

For now assume that only a single Xbean consumes the output of the previous; later we will describe an Xbean that acts as a parallelizer.

The Xbean paradigm is a very general data flow mechanism. XML describes structured data;

Java provides the computation and control on the data. As described in detail later, Xbeans are functionally composable simply by supporting a couple of minimal interfaces.



Enterprises want to exchange data. Industry specific standards efforts are defining XML Data Type Definitions (DTDs). These DTDs represent the semantics and format of the data to be exchanged.

Enterprises, however, have their data in their own databases defined by existing schema. That is, no two enterprises represent the same data in the same way. The idea is to access native data, translate it according to a standard DTD, transport it, translate it according to a native DTD and finally store it.

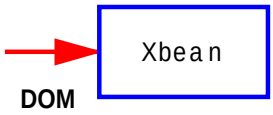
This slide illustrates a simple data exchange between enterprises using Xbeans. The blue boxes represents different Xbeans.

Each Xbean is configured appropriately. The *accessor* Xbean is configured to perform a particular SQL query and represent the result as an XML document. The *translator* Xbean is configured to translate the incoming XML document into an XML document that conforms to the agreed upon DTD for exchanging data. The *sender* and *receiver* Xbeans are configured to cooperate to transport the data.

At the enterprise that receives the data, a configured *translator* translates the data from the agreed upon DTD to a DTD that more closely matches the native schema. Finally the *accessor* is configured with an SQL query that stores the incoming data appropriately.

Sink Xbeans

- **An Xbean that consumes a DOM document.**



- **Sink Xbeans *implement* the DOMListener interface.**

An Xbean that implements the DOMListener interface is called a "sink Xbean", that is it receives XML data via that interface.

Interface supported by a Sink Xbean

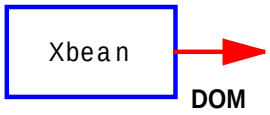
```
public interface DOMListener
   extends EventListener {

   public void documentReady(
                        DOMEvent evt)
            throws XbeansException;
}
```

Here's the DOMListener interface. It has a single documentReady operation. The argument is the DOM document. This interface conforms to JavaBean standards for events.

Source Xbeans

- **An Xbean that produces a DOM document.**



- **Source Xbeans *implement* the DOMSource interface and *use* the DOMListener interface.**

An Xbean that *implements* the *DOMSource* interface and *uses* the *DOMListener* interface is called a "source Xbean", that is it is a source of XML data.

Interface supported by a Source Xbean

```
public interface DOMSource {
  public void setDOMListener(
                 DOMListener
  DOMListener);
  public DOMListener getDOMListener();
}
```

- Most Xbeans are both sources and sinks.

Here's the DOMSource interface. It has operations for getting and setting the next Xbean in the chain. These operations correspond to the Java property DOMListener.

Generic Xbeans

- A generic Xbean is an Xbean that will operate on any DOM document.
- The Java implementation does not statically depend on a particular DTD.

Generic Xbeans process any kind of XML document using the DOM API. Many generic Xbeans are configured using the standard Java Bean mechanisms of property editors and customizers

Type Specific Xbeans

- An Xbean can be type specific.
- The Xbean checks the document type.
- The Java code is written to operate on a document supporting a particular DTD.

Xbeans can also be specific to a particular XML document type, that is it can be programmed to only work on XML documents whose type is known at compile time. While less general, they can still be part of a channel. Type specific Xbeans must check the type of the incoming XML document to ensure type integrity. Generic Xbeans can receive and process the output of a type specific Xbean.

Customized Xbeans

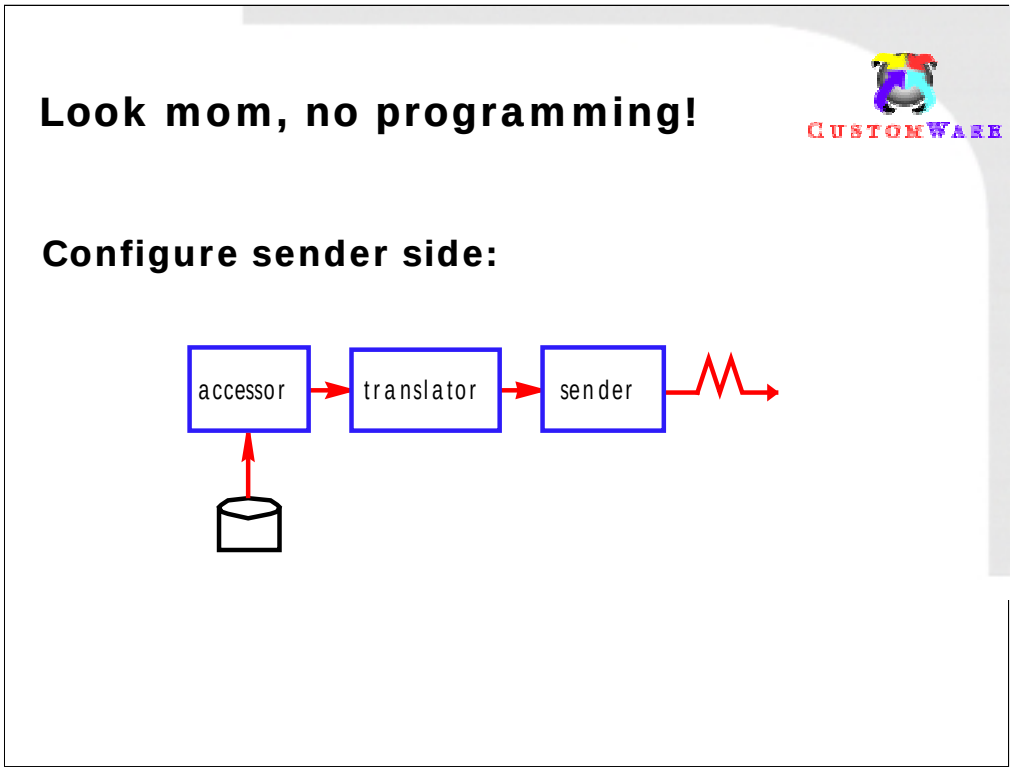
- **A customized Xbean is a generic Xbean that is customized at *design time* to operate on a specific type of XML document.**
- **The code is written generically but at runtime it operates on documents supporting particular DTDs.**

The most interesting and useful kind of Xbean is a customizable one. A customizable Xbean is written generically; that is, its code is written for any type of XML document.At design time, it is customized to operate on particular types of XML documents.

Xbeans are Java Beans

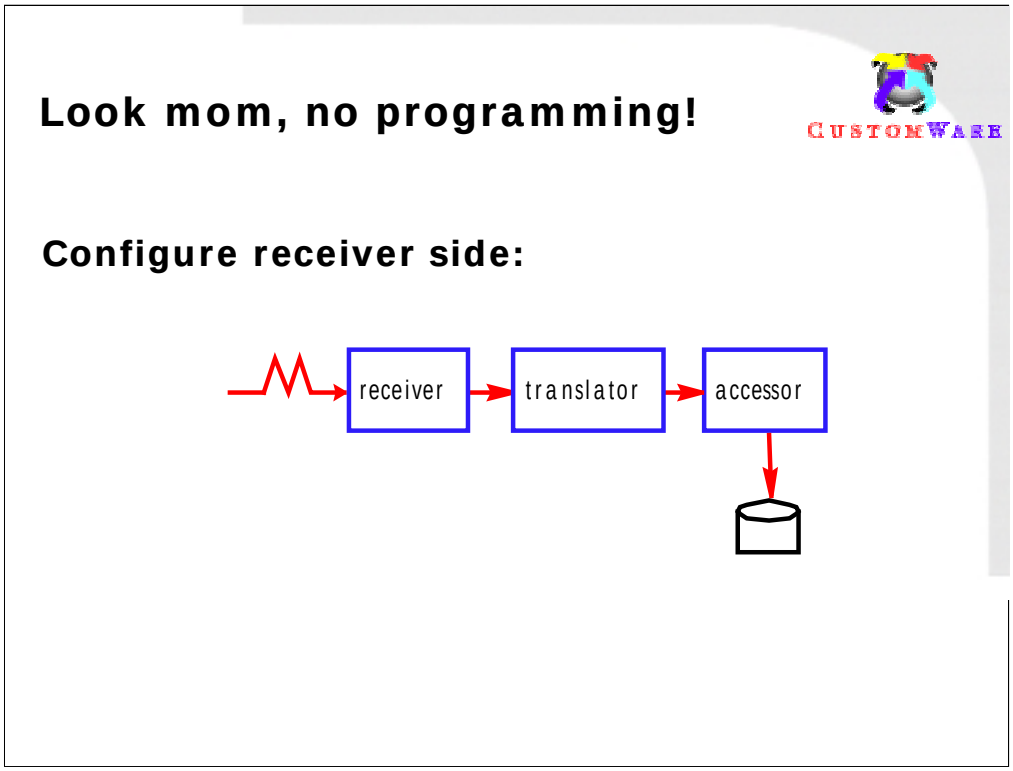
- **Xbean distributed applications can be created and configured in Java IDEs with Java Bean design tools:**
  - Visual Age
  - JBuilder
  - Café
  - NetBeans/Forte
  - FreeBuilder
  - ...

Xbeans can be configured using the standard Java Bean mechanisms of property editors and customizers. Using a Java Bean design tool, such as IBM's Visual Age for Java, Borland's JBuilder, Symantec's Visual Cafe, FreeBuilder or NetBeans/Forte, a developer can visually instantiate, customize and connect Xbeans. Complete distributed applications can be created, often without writing any code.



Given the right set of Xbeans and a Java Bean design tool, a simple distributed data exchange application can be built without any additional programming.

On the sending enterprise side, an accessor Xbean is configured to query a data source, a translator Xbean is configured to translate the data to an agreed upon DTD and the sender Xbean is configured with the identity of the receiver.



On the receiving enterprise side, the receiver Xbean is configured with its identity so that it will advertise its services at run time. Similarly, a translator Xbean is configured to translate the data from an agreed upon DTD to a format acceptable to the accessor Xbean. The accessor Xbean is configured to update the data source according to the receiving enterprise's schema.

Of course, other application specific logic can be added by adding other Xbeans. This shows the basic communication architecture for the data exchange application.

Some Example Xbeans

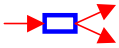
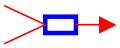
	Design time	Run time
data access	a mapping between result set and DTD	Produces (consumes) a DOM document according to the mapping
translator	a mapping between DTDs.	translates incoming document to a document of the other DTD.
sender receiver	sender: the id of the receiver receiver: its id	communicates XML over the wire.

Here is a summary of some useful Xbeans. Once again, recall that the properties set at design time can also be set at run time.

Some Example Xbeans ...

	Design time	Run time
viewer	presentation properties	visualizes the incoming DOM document
parser	the XML source	produces a DOM document
writer	the output	produces XML text from a DOM document

Some Example Xbeans ...

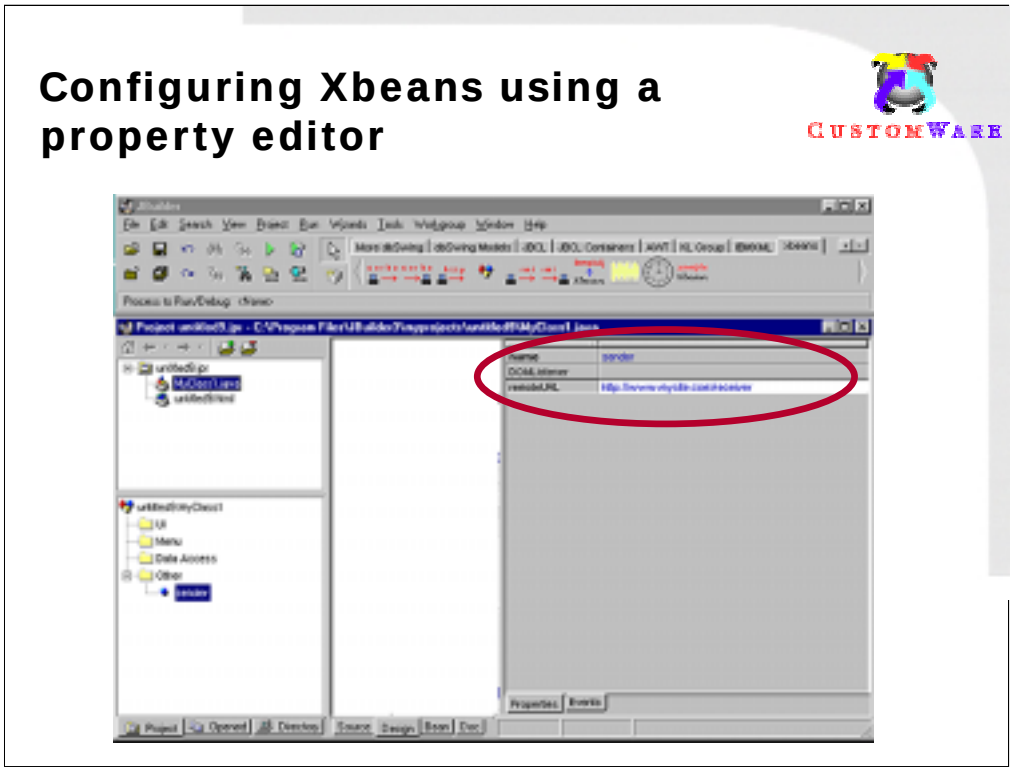
	Design time	Run time
parallelizer 	whether to copy or share a reference to outgoing DOM	creates multiple threads, shares or copies DOM
synchronizer 	criteria for choosing source DOM	receives multiple DOM documents and passes on one
logger	where output should go	produces a log record for each incoming DOM document

Some Example Xbeans ...

	Design time	Run time
timer		records current time when DOM document is processed
meter		records size of DOM document

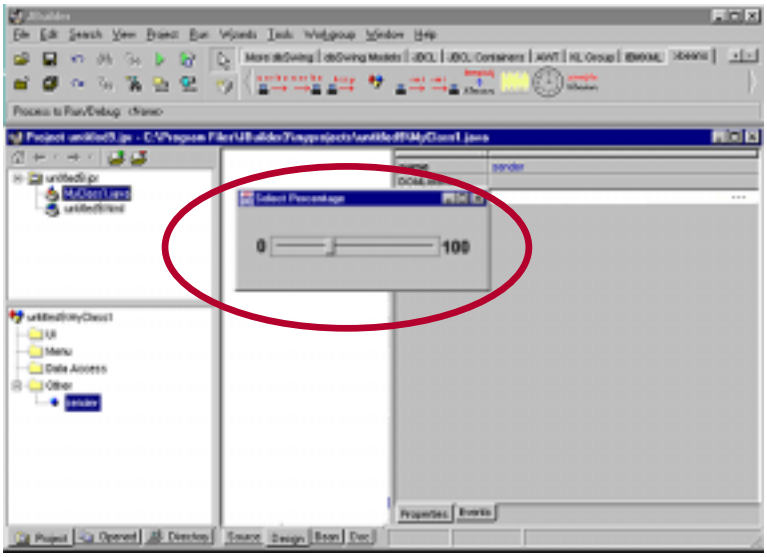
Plus many more yet to be invented Xbeans!

These are simply examples. Lots of generic and not so generic Xbeans can exist.

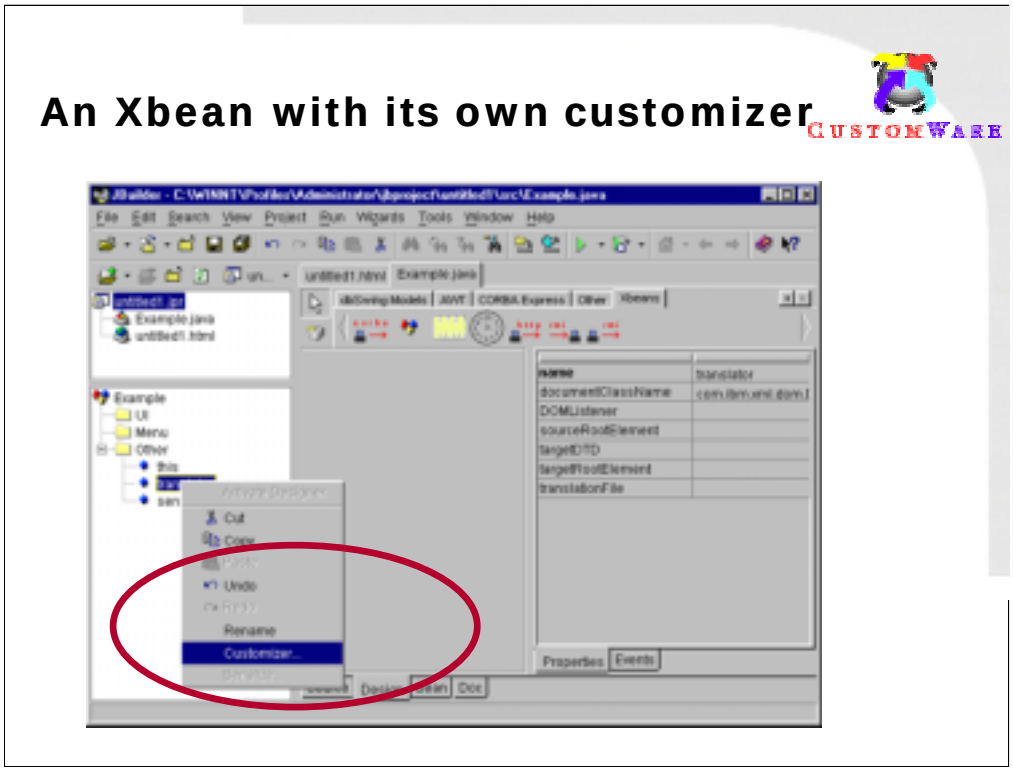


In the simplest case, the Java IDE's property editor can be used to set a property value at design time. This is an example of setting a property using Borland's JBuilder. Similar functionality exists in other Java IDEs.

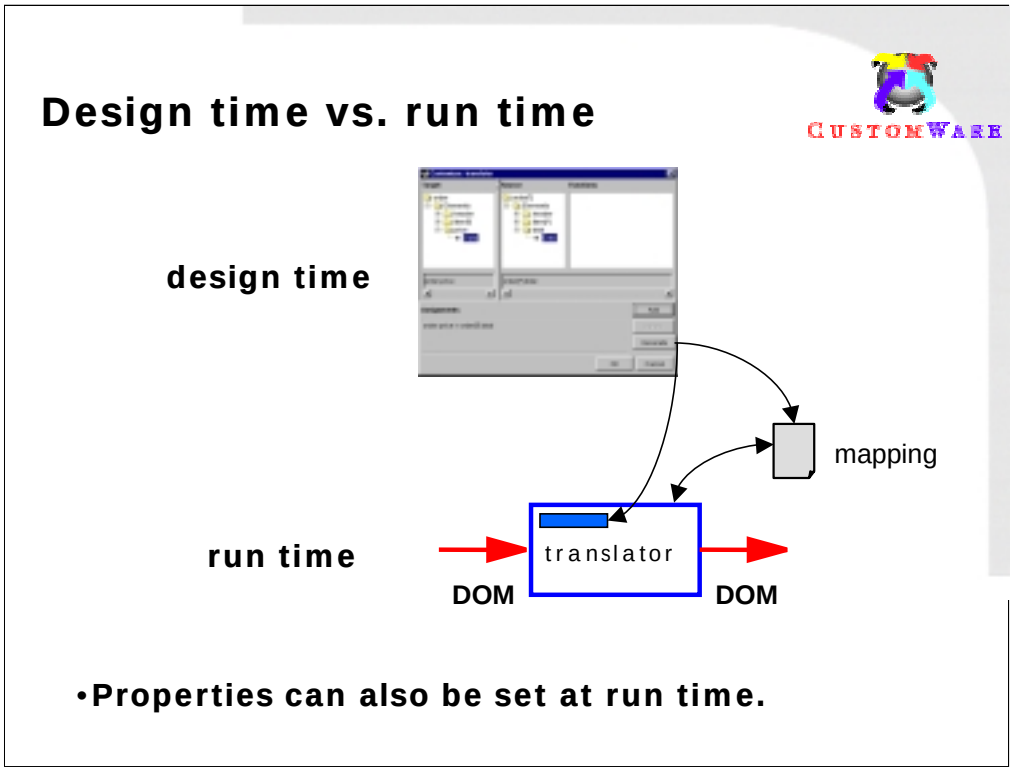
An Xbean with its own property editor



Some beans defined their own property editors. The editors can be invoked from the IDE to set a property value at design time. Again, this is an example of setting a property using Borland's JBuilder. Similar functionality exists in other Java IDEs.



A *customizer* is used when more sophisticated bean configuration is needed. For example, if several properties are set from a complex user interaction. Customizers can be invoked directly from the IDE at design time. Again, this is an example of invoking a customizer using Borland's JBuilder. Similar functionality exists in other Java IDEs.



This slide summarizes the difference between configuring the Xbean at design time and the Xbean processing at run time. The customizer is invoked at design time to configure the Xbean. The result of this is a mapping document which describes the translation. The customizer sets a property in the Xbean that specifies the mapping document. At run time, the translator Xbean processes the mapping document to configure itself. When the translator receives a DOM document it translates it according to the mapping.

Note that this configuration is not just limited to design time. At run time, the mapping property could be set on the translator Xbean. This would cause it to process a different mapping document and make different translations.

Why a translator Xbean?

- Assume we wish to exchange data.
- We agree upon a common format for the data we are going to exchange.
- This common format is specified by a DTD (or schema).
- Sender of the data probably does not have the data in that format.
- Receiver of the data probably does not want to manipulate and store the data in that format.

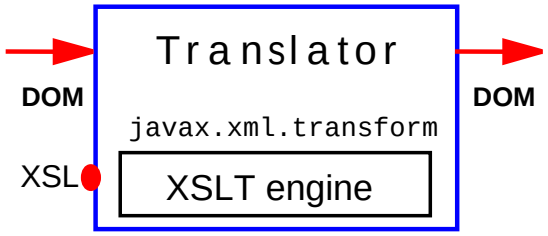
So, let's turn our attention to the translator Xbean. Why do we need a translator Xbean?

The assumption is that two or more parties that are going to exchange data do so by agreeing on a common format for the data. This common format is specified by a DTD or schema. There is little more the parties must agree upon.

Once there is an agreed upon format, everybody has a problem. The sender of the data probably does not have the data in that format. The data exists in the sender's own local format, probably in a database. Similarly, the receiver of the data does not want to manipulate and store the data in the agreed upon format.

The translator Xbean solves this problem. It allows the data to be mapped from one format to another.

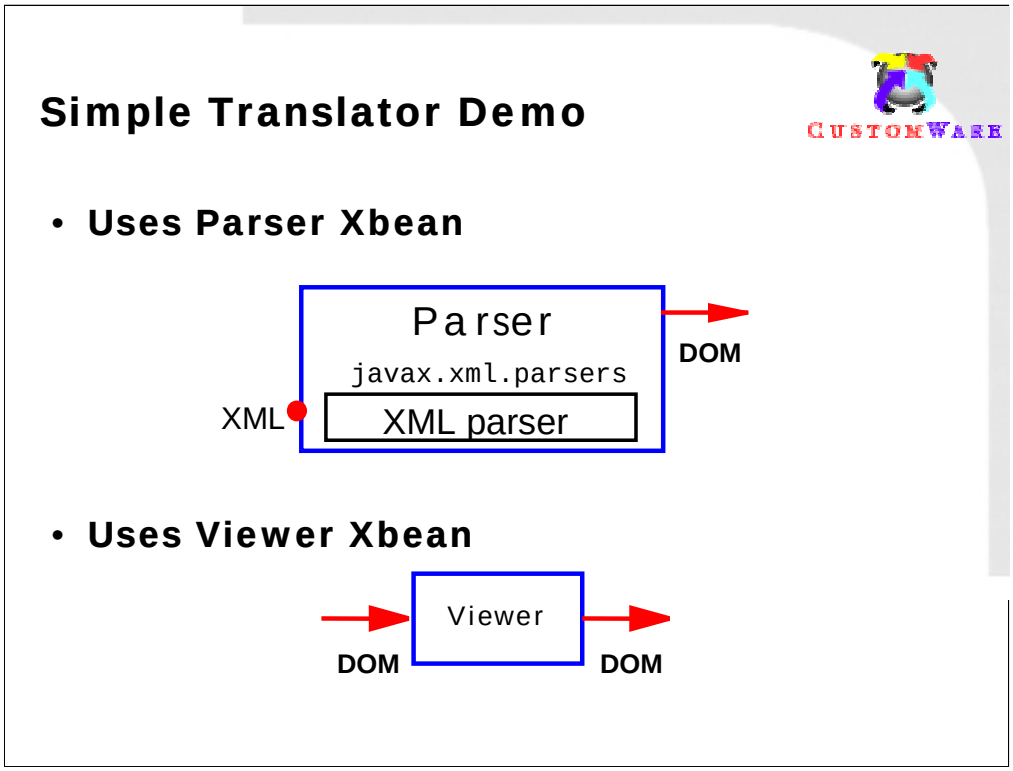
Translator Xbean



- **Can be configured to use any XSLT engine that supports TrAX, e.g. Xalan 2**
- **XSL property specifies the translation**
  - Set at design time by user OR
  - Set at design time as a result of running customizer.

At runtime, the translator Xbean receives a DOM and translates it to another DOM, according to an XSL translation. The translator Xbean uses an XSLT engine to accomplish this. Any XSLT engine that supports the TrAX interfaces (the javax.xml.transform package) will work.

The translator Xbean has an XSL property that specifies the translation to be done at runtime. This property can be set directly by the user or it can be set by the translator's customizer. If the customizer is used, then the XSL file will be generated.



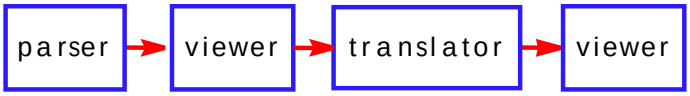
Let's see a demo of the translator Xbean. The demo also uses the parser Xbean and the viewer Xbean.

The parser Xbean has a property indicating the XML file to be parsed. The parser Xbean produces a DOM from the XML file.

The viewer Xbean receives a DOM, displays it in a Swing Tree and passes it on to the next Xbean in the channel.

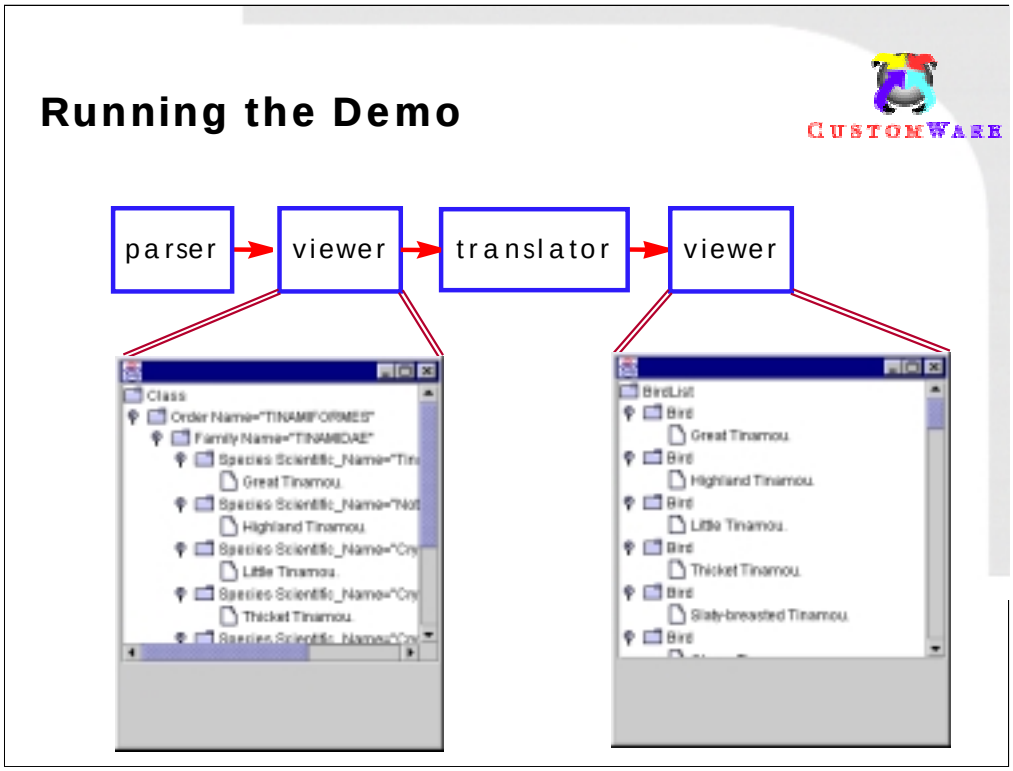
The viewer is handy for debugging Xbean channels. We will use it in the demo.

Simple Translator Demo



- **Parser configured with XML file, produces DOM.**
- **Viewer shows document as a tree.**
- **Translator configured by XSL file, translates from one DOM to another.**
- **Viewer shows resulting document as a tree.**

So the simple translator demo will parse an XML file, view the parsed document, translate it using the translator and then view the translated document.



In the talk the demo is given live. This is what you would see. First the birds.xml file is parsed and viewed by the viewer. Then it is translated to the agreed upon format and then the translated data is viewed.

This simple demo application is built without writing any Java code. The Xbeans are simply configured using a Java IDE into the application.

Generating XSL

- Writing XSL can be tedious and error prone.
- Requires XSL expertise.
- The translator's customizer can generate the XSL.
- Customizer is a GUI design tool for the translator.

The translator Xbean has a simpler usage model than what was just demonstrated. Rather than writing XSL, the translator's customizer can be run at design time to generate the XSL. The customizer is a GUI design tool for the translator.

Design Goals for the Customizer

- **Eliminate hand coding of XSL**
- **Handle DTD->DTD case**
- **Output will be XML (DOM)**
  - Not interested in producing HTML, text, pdf, etc.
- **Customizer not required**
  - Can set the translator's XSL property directly
- **Simplicity over expressive power**

We had certain design goals for the customizer. It's main goal is to eliminate the hand coding of XSL. It is geared to solving the DTD to DTD case. That is, the usage scenario where the parties exchanging data agree upon a particular DTD for the data.

The customizer is not meant to map user actions to any arbitrary XSL. For example, using XSL it is possible to produce raw text, HTML or other non-XML formats. Since the translator is used to translate XML to XML, there is no need to support XSL that generates non-XML output.

On the other hand, it is not required to use the customizer. A user can write XSL and configure the translator to use it.

Assignment model

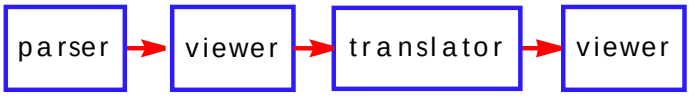
- Customizer supports an assignment model, rather than XSLT pattern matching.
- User navigates target and source DTDs and maps data from source to target.

The customizer supports an assignment model, rather than XSLT pattern matching. The user navigates target and source DTDs and maps data from source to target.

The demo will illustrate this.

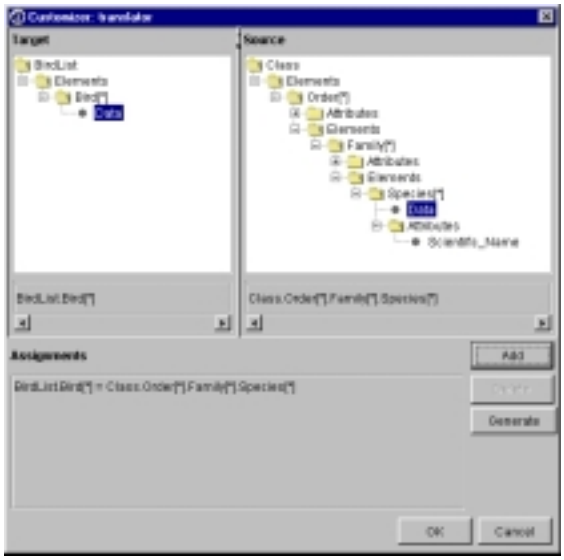
Demo with Customizer

- **Same demo structure but XSL is generated by customizer.**

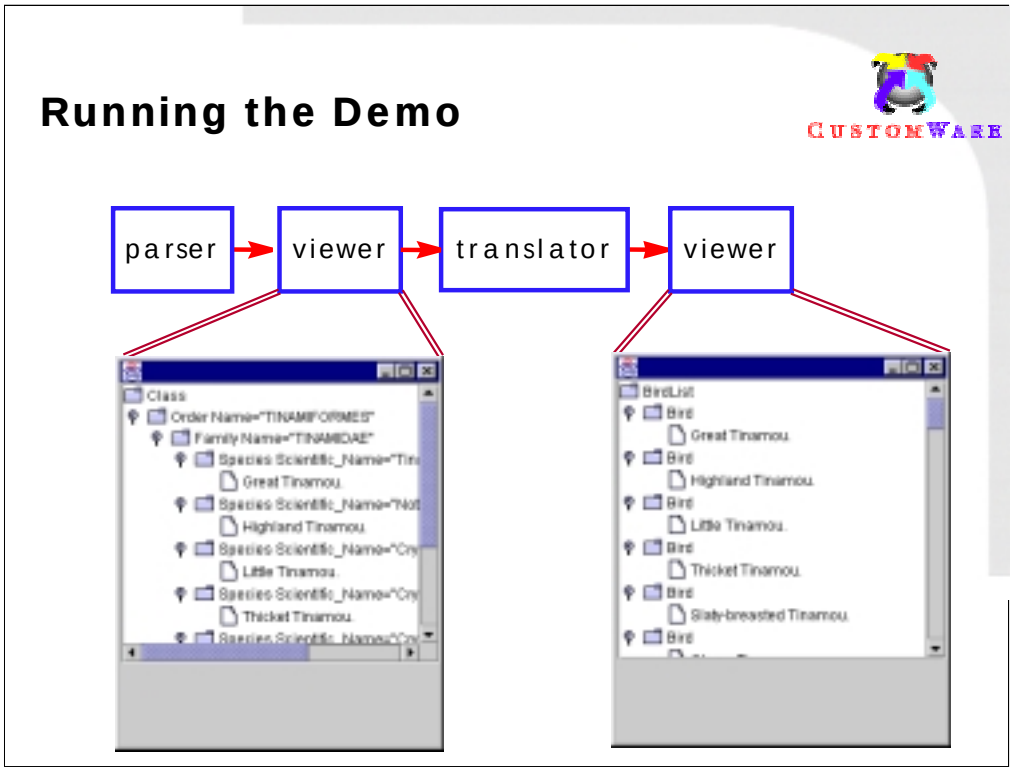


Again we are going to do the same demo but the translator will be customized using it's customizer. We will not write any XSL.

Running the Customizer



We have a target DTD and a source DTD. If we are sending the data then the target DTD is our agreed upon data exchange format. The user indicates that all data in the Species element in the source data will be assigned to the Bird element in the target data.



It’s the same demo as before. The result is the same.

Xbeans.org

- **Open source project to create a freely available repository of Xbeans.**
- **First release available:**
  - sender/receiver for http, rmi, corba
  - timer, memory meter
- **Looking for contributors:**

  www.xbeans.org

Xbeans.org is an open source project. The goal of Xbeans.org is to provide a rich repository of freely available Xbeans. The first version of Xbeans is available from the web site. Since Xbeans are clearly partitioned pieces of functionality with two well defined interfaces, independent and parallel development of Xbeans is greatly simplified. Unlike some open source projects, there is not a lot of coordination and project management required.

Xbean.org is looking for contributors of the Xbeans outlined here as well as any useful Xbean imaginable.

Xbeans Release Two

- **Goals**
  - Simplicity!
  - More functionality
  - No dependencies on proprietary code
    - Only depend on other open source projects
- **Significant Features**
  - all release 1 features
  - servlet receiver
  - better on the wire communication -- no Java serialization

The next release of Xbeans is going to be made available soon. The goals are to add significant features, remove the dependencies on non open-source code and to simplify the use of Xbeans even further.

The first release of Xbeans had dependencies on IBM and Borland code for parsing and CORBA. To use the first release, you were required to have those products. The next release will only depend on open-source code.

Release two …

- **Features continued …**
  - parser (wraps any javax.xml.parsers)
    - E.g. Xeres
  - translator based on xslt javax.xml.transform
    - Xalan
    - Wrapped xsltc perhaps
  - viewer
  - parallelizer
  - synchronizer
  - tests/sample programs
- **Available later this year**

Here are more features that will be included in release two.

Future release

- **Features**
  - Relational database accessor
  - HTML scraper
- **Xbean wrappers of other open-source code.**

Finally, future releases will support accessing relational databases and pulling data from web sites.

Xbeans in French pay phones?

CustomWare

france telecom

For French Overseas Territories:

- 0 + DOM code + correspondent's number

CustomWare

Questions