V2Features

XMLBeans Version 2 Plans

Overall XMLBeans Vision in relation to V2

XMLBeans is about providing access to the full power of XML to Java users in as user friendly manner as possible. XMLBeans Version 1 provides Java /XML binding that supports near 100% of XML Schema features as well as full access to the XML Infoset via the `XmlCursor` API. Version 2 is more evolutionary than revolutionary, it is a continuation of the XMLBeans Version 1 work which includes expanding on the core XMLBeans Java/XML binding features as well as the underlying XML Store. Perhaps the most significant feature that has deep architectural impact is DOM Level II support over the XML Store. As with Version 1, but more so with Version 2, XMLBeans can be viewed as a model for access XML in Java in general, both a Java/XML binding strategy for accessing XML when XML Schema(s) are available, as well as an API for direct access to the XML via DOM or `XmlCursor`.

The following section outlines goals for XMLBeans in Version 2.

Specific V2 Objectives

- **DOM Level II Support** - In Version 1 the only way to get a DOM Node for an `XmlObject` or `XmlCursor` was to use the `newDOMNode` API which copied from the XML Store into a new DOM Node. This is a fairly innefficient operation and the DOM Node was disconnected from the underlying XML Store so changes to the DOM Node were not reflected into the XML Store. There have been a significant number of requests for XMLBeans to support DOM natively since lots of tools (and developers) know how to work with DOM already and do not know how to work with `XmlCursor`. Also, SAAJ 1.2 now relies on an underlying DOM and having a SAAJ implementation on XMLBeans should be possible (since SAAJ represents a non-lossy SOAP Envelope Document). In Version 2 DOM will be implemented natively, meaning the tree within the XML Store will be able to represent DOM objects. Note that `XmlCursor` will remain fully supported so users will be able to switch between DOM, `XmlCursor`, and `XmlObject` (either untyped or typed).
- **Extensions** - In general XMLBeans generated interfaces have been pretty static, in large part due to the XMLBeans overall objective to correctly support the XML Schema type system (including the custome types defined in the schema) in Java. You can map target namespace/package and element/property names but that was about it. In Version 2 (this may be ported to Version 1 as well) you will be able to add custom functionality to generated XMLBeans interfaces/classes. To accomplish this you will be able to pass the Schema Compiler two things 1) an interface that defines the set of methods to implement and 2) a static handler which implements this functionality (it is debatable whether this should be static or instance based, there are arguments both ways). The underlying XMLBeans generated classes will implement the interface and for each method call out to the static handler. Note that this capability allows XMLBeans classes to *be* your interface, this could allow certain binding type strategies to sit on top of XMLBeans. For example, you could imagine an SDO (Service Data Objects) implementation on top of XMLBeans such that the SDO `DataObject` interface could be implemented by corresponding XMLBean(s). *7/14/04 - Note this feature has been implemented see this wiki page http://wiki.apache.org/xmlbeans/ExtensionInterfacesFeature for more information*.
- **Pre and Post methods** - In addition to being able to extend XMLBeans generated Java classes there have been many requests for event type methods. In Version 2 (may be implemented in Version 1 also) you will be able to specify *Pre* and* Post* methods that will be called before and after changes to the XMLBeans properties.* *7/14/04 - Note the first version of this feature has been implemented see this wiki page http://wiki.apache.org/xmlbeans/PrePostSetFeature for specifics*.
- **User Types** - Custom user types for simple values. See http://wiki.apache.org/xmlbeans/UserTypes for specifics.
- **Performance** - As in XMLBeans V1 performance is paramount V2. Since XMLBeans always loads an XML Store and then provides a binding view on to the store some amount of additional overhead, compared to other Java/XML binding frameworks that unmarshal directly to Java objects (but throw away XML Infoset info that doesn't fit into the Java objects). This makes the performance bar all the higher for XMLBeans so that the tradeoff for additional functionality and information is minimal. The primary focus for runtime performance is in the area of the XML Store although there are other factors to consider as well (for example parsing). Every effort will be made to make the XML Store as highly performant as possible. Benchmarks against related technologies need to be created (with source for the benchmarks available of course) and performance tests included in the test suite to assure that performance regressions do not occur in the development process.
- **Compilation Performance** - In XMLBeans V1 compilation, while very performant, was essentially fully batch oriented. If one minor change occurred to a single XML Schema amongst a whole set of XML Schemas that are submitted to the XMLBeans schema compiler the entire XMLBeans schema type system and generated classes were recreated. In other words XMLBeans has been unable to take advantage of compilation work that already occurred. For XMLBeans users with large numbers of XML Schemas, or IDE's integrating XMLBeans, compilation time can be problematic. In XMLBeans V2 major steps towards addressing this issue will be implemented. When doing an XMLBeans compile you should be able to pass in existing XMLBeans compiled artifacts (probably not a jar, most likely an exploded directory, or perhaps an in memory representation) and the XMLBeans compiler would do only the *incremental* work necessary to rebuild the type system and the java classes. In V2 this will likely show up as 1) only the java classes that change should be compiled and replaced and 2) incremental XML Schema compilation at the **namespace** level. Our opinion is that the doing incremental compilation at the **type** level is too low level and too difficult to implement (at least in V2)
- **Improved XQuery/XPath integration** - XMLBeans V1 has not implemented the `execQuery()` API (on `XmlCursor` and `XmlObject`) and the `selectPath()` API is integrated with Jaxen (XPath 1.0). The original proprietary XMLBeans that was donated to Apache (the basis for Apache XMLBeans V1) XMLBeans integrated with a proprietary BEA XQuery engine (note this implies XPath 2.0). In V2 we need to reexamine and clean up XMLBeans XQuery/XPath functionality. Since XMLBeans is store based a *major* advantage of XMLBeans should always be *great* XPath and XQuery (arguably XSLT as well ...) support. _ rem: Ideally we would talk BEA into open sourcing their XQuery engine <smile>_
- **JDK 1.5 Generics and Enumerations** - XMLBeans V2 will work with both JDK 1.4 and JDK 1.5. V2 will provide the option to take advantage of JDK 1.5 Generics and/or Enumerations in XMLBeans generated classes.
- **SAAJ 1.2 Support** - SAAJ defines a SOAP Envelope Document structure and in the latest spec release SAAJ 1.2 it is defined to be tightly integreated with DOM. In V2 it should be possible to create a SAAJ Envelope structure such that the SAAJ Nodes *are* the same objects as the underlying DOM Nodes. This would save a SAAJ implementation on top of DOM not to have a parrallel Node tree and should improve performance substantially. Note the goal is not to develop a SAAJ implementation within XMLBeans but to allow for a SAAJ 1.2 implementation to be effectively implemented on XMLBeans. SAAJ is a web services related technology and could have web services container specific functionality (logging, error handling, etc.) in it's implementation. XMLBeans V2 will have a SAAJ interface and the underlying XML Store tree nodes will implement the appropriate SAAJ interfaces and call back through the XMLBeans defined SAAJ interface. The goal is that any web services container could implement SAAJ 1.2 efficiently over XMLBeans.
- **Improved Error Handling** - In XMLBeans V1 error handling is ok but needs work. Errors should have codes. Errors should point accurately to the object in Error (location information needs be accurate). Error messages should be well written and consistent.

Other Items To Be Researched/Considered

- **Schema to POJO/POJO to Schema** - XMLBeans generated classes are *tightly* coupled to the corresponding XML Schema(s). This is not necessarily a bad thing, it is what allows XMLBeans to fully support XML Schema functionality in a seamless and elegant manner. XML Beans versioning has essentially the same issues as XML Schema versioning. Consequently best practice with XMLBeans is to architecturally isolate XMLBeans through other layers using Data Access Object type patterns (using DAO informally here). Essentially this involves mapping the data in the XMLBean to a value object or business object and the rest of the codebase writes against this mapped object. Thus if the XML Schema changes it may be possible to change redo the mapping from the XMLBean to the value/business object. Currently with XMLBeans this is an exercise left up to the developer. It may be possible for XMLBeans to provide a *mapping object* that could facilitate a more *loosely coupled* architecture like this. This could entail a mapping file (or annotations) against an existing Java class that would allow the synchronization between the XMLBean and the corresponding class. There is still a good amount of research to be done here. It is possible this is more of a best practices document outlining a Data Access Object type pattern then an XMLBeans feature.
- **Canonicalization** - This is may be more research than committed feature. One of the more expensive aspects of XML Signature is the C14N processing. This involves reading the target XML nodes from the tree and *canonicalizing* the XML. It may be possible to improve this performance by supporting C14N directly in the XML Store (then again maybe not ..).
- **Hibernate integration** - There have been multiple discussions around using XMLBeans with Hibernate (http://nagoya.apache.org/eyebrowse/ReadMsg?listName=xmlbeans-user@xml.apache.org&msgNo=305). Ideally this would show up as some sort of plug-in into Hibernate that supports XMLBeans persistence effectively.
- **Eclipse and Netbeans plug in for XMLBeans** - XMLBeans supported plug-ins for these popular IDEs. Weblogic Workshop already has strong support.

Items we don't expect to get to

- **XmlObject Lossy Binding** - It seems feasible to generate XmlObject derived implementations of XMLBeans classes that do not require an underlying XML Store. Of course that would mean that a `newCursor()` call on `XmlObject` would get `UnsupportedOperationException`. The idea would be that an XMLBeans user may not initially need the functionality of an XML Store and could work completely through `XmlObject` (e.g., the binding layer). If at any point there is a need to have an underlying XML Store then there would be a compile option to have the XML Store available. This is primarily a performance optimization the generated java classes would still derive from `XmlObject` and have the same shape. A lot of research needs to be done here and it is possible this approach will not be feasible.
- **DOM Eventing** - DOM Eventing is challenging with the XML Store architecture and may not be feasible. XMLBeans will not address it in V2.
