The generic parser

1. The generic parser

The most basic part of JaxMeXS is its generic parser. This part is suited for arbitrary XML document types, for example configuration files and the like. The actual XML Schema parsers (the syntax parser and the logical parser) are sitting on top of the generic parser. If you aren't interested in the generic part, you may safely skip to the next section, which is basically self contained. However, some details are best described here.

The generic parser follows an idea from the Ant project: A SAX parser is responsible for the configuration of a bean. Any XML element is mapped to a Java bean. The XML elements attributes are mapped to properties of the bean and the child elements are mapped to other beans, which are part of the parent bean. If you know how to write an Ant task, you know how the generic parser works.

The generic parser is specified by the interface XsSAXParser and implemented by the class XsSAXParserImpl. There's few things to know about that. The most important parts are some other interfaces and classes which you might like to implement or extend:

1. The AttributeSetter interface is responsible for mapping XML attributes to bean properties.
2. The TextSetter handles character data contained in elements.
3. And, finally, the ChildSetter creates new beans for any child element.

2. The AttributeSetter interface

The AttributeSetter interface consists of a single method:

```
public void setAttribute(String pQName, String pNamespaceURI,
                         String pLocalName, String pValue) throws SAXException;
```

This method is invoked for any attribute that the SAX parser finds.

The AttributeSetters main idea is as follows: Suggest, that the attributes local name is `foo`. A method `setFoo(SomeClass pValue)` is searched. f such a method is found, the attribute value is converted into an instance of `SomeClass` and the method is invoked. More formally, the default implementation AttributeSetterImpl behaves as follows:

1. If the bean has a method `setAttribute(String, String, String,`

`String)`, it is invoked with the same arguments `pQName`, `pNamespaceURI`, `pLocalName`, `pValue`. If this method does not have the result types `boolean` or `Boolean`, or if the result is `true`, then the `AttributeSetterImpl` assumes that the property is set.

2. If the property is not set, and the bean has a method `setProperty(String)`, then this method is invoked with the attribute value.
3. Otherwise, if the bean has a method `setProperty(T)`, and the class `T` has either of a method `public static T valueOf(String)` or a constructor `public T(String)` (in that order), then the method `setProperty(T)` is invoked with the value obtained by an invocation of the method `valueOf()`, or the constructor, respectively. Note, that this applies in particular to the classes Long, Integer, Short, Byte, Double, Float, java.math.BigInteger, java.math.BigDecimal, java.io.File, and StringBuffer.
4. If the bean has a method `setProperty(boolean)`, the method will be invoked with the value `true` (the value specified in the XML file is either of `true`, or `1`) or `false` (the attribute has any other value).
5. If the bean has a method `setProperty(char)`, or `setProperty(Character)`, the method will be invoked with the first character of the value specified in the XML file. If the value contains zero or multiple characters, an IllegalArgumentException is thrown.
6. If the bean has either of the following methods, in that order:
   - `setProperty(long)`
   - `setProperty(int)`
   - `setProperty(short)`
   - `setProperty(byte)`
   - `setProperty(double)`
   - `setProperty(float)`

   then the attribute value is converted into the respective type and the method is invoked. An IllegalArgumentException is thrown, if the conversion fails.
7. If the bean has a method `java.lang.Class`, the `XsSAXParser` will interpret the value given in the XML file as a Java class name and load the named class from its class loader. If the class cannot be loaded, it will also try to use the current threads context class loader. An exception is thrown, if neither of the class loaders can load the class.

3. The TextSetter interface

The TextSetter interface is invoked from within the SAX ContentHandlers method `characters(char[] pBuffer, int pOffset, int pLen)`. It's task is to fill the bean with character data. Note, that the latter method may very well be called multiple times, even for a single character sequence in the XML file. For example, if the XML reader loads the XML file in blocks of 1024 characters and a block stops right within an elements character data, then it is valid behaviour to call the `character(char[], int, int)` method twice: Once for the first part, which sits at the end of the 1024 characters and once for the remaining part. The same holds for the `addText()` method:

```
public void addText(String pValue) throws SAXException;
```

The default implementation is TextSetterImpl, which behaves as follows:

1. If the bean has a method with the same signature `public void addText(String)`, then the method is invoked.
2. If the bean doesn't have such a method and the supplied text is all whitespace, then the text is ignored. Otherwise an exception is thrown.

4. Handling child elements

Embedded child elements are handled by the interface ChildSetter and its default implementation ChildSetterImpl. The interface exposes a single method:

```
public ContentHandler getChildHandler(String pQName, String pNamespaceURI,
                                      String pLocalName) throws SAXException;
```

The purpose of the method is to create a SAX handler for this element. If such a handler is returned, then it receives the SAX events generated for the element. The default implementation works as follows:

1. If the bean has a method with the same signature `public ContentHandler getChildHandler(String, String, String, String)`, then this method is invoked. A non-null result will be used as a ContentHandler for the child element.
2. If the bean doesn't have such a method or if the method returned null, and the local name is `foo`, then a method `public T createFoo()` is searched, with an arbitrary result type T. If there is such a method it is invoked and a new instance of XsSAXParserImpl is created to configure the bean.
3. An exception is thrown otherwise.