Issues with date/time values

W3C XML Schema defines 9 different datatypes for various forms of time-related values. Most of these are not very useful for data exchange between applications. Consider gMonthDay, for instance, which represents a particular month and day of the month without regard to a year - although this could be useful for some specialized applications, most applications need full date specifications including a year. Three of the schema time-related datatypes are widely used for data exchange: dateTime, date, and time. The dateTime datatype is effectively the "core" type, with date and time types just trimmed versions of the dateTime representation.

Unfortunately, the dateTime/date/time datatypes all suffer from one severe limitation when it comes to their use for data exchange: Schema allows the values to be specified either in UTC time (or with a fixed offset from UTC, equivalent to UTC) or without reference to any time zone. The latter case is what usually causes problems for data exchange. Applications always want to work either with fully-specified date/time values, representing a particular instant of time which can be converted at will into any time zone, or with date/time values without reference to time zone (such as a birthday). Very few applications are written to work interchangably with both types of values.

In Java the standard date/time representation uses either a `java.util.Date` value or a `java.util.Calendar` value. The former corresponds to a fully-specified dateTime value in schema terms, since it is a specific millisecond in UTC. The latter has no schema equivalent, since it includes actual time zone information (rather than just an offset from GMT for a particular instant of time, as allowed by the schema representation). Various methods are used by different data binding tools to convert these two Java types to and from XML representations, but no method can correct the inherent differences.

Date/time conversions in JiBX

Because of these issues of date/time representations in schema, and the problems in matching the schema datatypes to Java types, JiBX supports working with both standard Java classes and Joda date/time classes. The Joda library provides a much richer set of representations for date/time values than standard Java, and the richer representations allow more ways of handling the conversions. This still doesn't correct for the fundamental flaws in the schema representation of date/times, but it at least allows you to easily control how the conversions are handled for your code.

The following table gives the full range of date/time formats built into the JiBX handling, including both default formats (as indicated) and formats you can specify by name for use in your bindings:

Date/time Formats

Type	Format Label	Conversion
`java.util.Date`	Date.default	Converts instances of `java.util.Date` to and from the schema dateTime

		representation (a text representation like "2000-03-21T01:33:00", with optional trailing fractional seconds, and difference from UTC). Since schema doesn't have any concept equivalent to Java time zones, this conversion always serializes times as UTC values (identified by a trailing "Z"). When deserializing times which do not include a "Z" or offset from UTC it treats the values as UTC. **Default**
`java.sql.Date`	SqlDate.default	Converts instances of `java.sql.Date` to and from the schema date representation (a text representation like "2000-03-21", with optional trailing time offset). Time zones and offsets are ignored by this conversion - any time offset is ignored when unmarshalling, and the output is generated without an offset or 'Z' UTC indicator when marshalling. **Default**
`java.sql.Time`	SqlTime.default	Converts instances of `java.sql.Time` to and from the schema time representation (a text representation like "01:33:00" with optional trailing fractional seconds and time offset). Time zones and offsets are ignored by this conversion - any time offset is ignored when unmarshalling, and the output is generated without an offset or 'Z' UTC indicator when marshalling. **Default**
`java.sql.Timestamp`	Timestamp.default	Converts instances of `java.sql.Timestamp` to and from the schema dateTime representation, just as the **Date:default** conversion does

		Date:default conversion does for `java.util.Date` instances. The only difference is that using the timestamp value permits greater precision in the time value represented, down to the nanosecond level. **Default**
`org.joda.time.LocalDate`	LocalDate.default	Converts instances of `org.joda.time.LocalDate` to and from the schema date representation, ignoring time zones (any time offset is ignored when unmarshalling, and the output is generated without an offset or 'Z' UTC indicator when marshalling). **Default**
`org.joda.time.DateMidnight`	DateMidnight.zoned	Converts instances of `org.joda.time.DateMidnight` to and from the schema date representation, including time zone (which is part of the `DateMidnight` structure). The local zone is used when deserializing a schema representation with no time zone information.
`org.joda.time.DateMidnight`	DateMidnight.local	Converts instances of `org.joda.time.DateMidnight` to and from the schema date representation, with the local zone used for all deserialized values (even if the schema representation includes time zone information) and no zone specified on output. **Default**
`org.joda.time.DateMidnight`	DateMidnight.UTC	Converts instances of `org.joda.time.DateMidnight` to and from the schema date representation, with the UTC zone used for all deserialized values (even if the schema representation includes time zone information) and zone

		zone information) and zone always specified as UTC on output (even if that doesn't match the zone information set on the `org.joda.time.DateMidnight` value).
`org.joda.time.LocalTime`	LocalTime.local	Converts instances of `org.joda.time.LocalTime` to and from the schema time representation, ignoring any time zone information when deserializing from a schema representation and leaving off any time zone information when serializing to a schema representation. **Default**
`org.joda.time.LocalTime`	LocalTime.UTC	Converts instances of `org.joda.time.LocalTime` to and from the schema time representation, ignoring any time zone information when deserializing from a schema representation and setting the time zone as UTC (with a 'Z' suffix) when serializing to a schema representation.
`org.joda.time.DateTime`	DateTime.zoned	Converts instances of `org.joda.time.DateTime` to and from the schema dateTime representation. When a schema dateTime value with time zone information included is deserialized, the time zone from the schema value is used for the constructed `DateTime` instance; if no time zone information is included, the local time zone is used. The actual zone set on the `DateTime` instance is used when serializing.
`org.joda.time.DateTime`	DateTime.UTC	Converts instances of `org.joda.time.DateTime` to

		and from the schema dateTime representation. The UTC time zone is used when deserializing a schema dateTime value with no time zone information supplied. Values are always adjusted to UTC and specified as UTC (using a 'Z' suffix) when serializing.
`org.joda.time.DateTime`	DateTime.local	Converts instances of `org.joda.time.DateTime` to and from the schema dateTime representation. Deserialized values always use the local time zone. When deserializing a schema dateTime value with time zone information included the value is converted to the local equivalent; if no time zone information is included in the schema value the local time zone is assumed. The actual zone set on the `DateTime` instance is used when serializing. **Default**
`org.joda.time.DateTime`	DateTime.strict-local	Like the **DateTime.local** conversion, above, but requires time zone information in schema dateTime values being deserialized (throwing an exception if no time zone information is present).
`org.joda.time.DateTime`	DateTime.strict-UTC	Like the **DateTime.UTC** conversion, above, but requires time zone information in schema dateTime values being deserialized (throwing an exception if no time zone information is present).

The actual implementations of the conversions for standard Java date/time classes are in the `org.jibx.runtime.Utility` class, while those for the Joda date/time classes are in the `org.jibx.runtime.JodaConvert` class. You can use the methods in these classes to construct your own custom combinations of serialization and deserialization handling.