Calendars, Dates, and Times

Calendars, Dates, and Times

Most developers don't need to be concerned with the issues discussed here - the OFBiz framework is designed to handle regional issues automatically. If you are a framework developer, then read on...

OFBiz supports date and time localization by using the ICU4J classes and/or the corresponding `java.util.*` classes.

OFBiz uses the ICU4J library classes instead of the corresponding JRE classes for an important reason: keeping current with changing standards. Whenever regional standards change, the ICU4J library is updated - which keeps OFBiz's internationalization current.

Framework developers get themselves into trouble with date/time localization regularly. This doesn't have to be the case if a little time is invested in understanding the issues involved.

Developers should:

1. Understand the related Java classes `java.util.Date`java.util.Localejava.util.TimeZonejava.util.Calendar and `org.ofbiz.base.util.UtilDateTime`
2. Avoid making assumptions
3. Never use millisecond arithmetic

Also long ago Adrian suggested this 3 rules/use-cases which make much sense and helps to understand, more could be added...

1. Automatic server processes like the Job Scheduler should use the **server's timezone.**
2. Customer-facing processes should use a product **store's timezone.** Example: "Your order will be shipped by 4:30 PM PST today."
3. Calendar applications (anything work effort related) should use the **user-selected timezone**.

The `java.util.*` Classes

The `java.util.Date` class, and its subclass `java.sql.Timestamp` should be thought of as a constant. It is an immutable moment of time occurring in GMT. (The `Date` class has setXxxx methods, but they are all deprecated.) You cannot change a `Date`'s time zone - it is, and always will be, referenced to GMT.

If you want to localize a `Date` instance, you'll need a `Locale` and `TimeZone` instance. The OFBiz framework makes the user's locale and time zone available throughout most of the execution path - usually in a context `Map`. Call the `UtilDateTime.toDateTimeFormat` method (leave the String argument null) to get a `DateFormat` instance. When you call `DateFormat.format`, the class will compute the user's date and time using the **immutable, GMT-referenced** `Date` and the information provided by the `TimeZone` instance. The format of the date/time `String` is controlled by the `Locale` instance.

If you want to perform date/time arithmetic, you have two options: use the `UtilDateTime.adjustTimestamp` method, or create a `Calendar` instance by calling `UtilDateTime.toCalendar` and adjust the date using that instance.

Avoid Making Assumptions

Different cultures use different calendars. Different regions can have different calendar rules, even when using the same calendar. The `Calendar` class, when used properly, will accommodate all of these differences.

Assumptions to avoid:

- A year has 12 months
- The week number in your locale is the same week number in another locale
- The week always starts with Sunday
- Everyone uses a Gregorian calendar

Never Use Millisecond Arithmetic

Knowing that the `Date` class wraps a millisecond value makes it tempting to use millisecond arithmetic to perform date/time calculations. That approach will **always** fail - because it doesn't take into consideration different calendar systems, time zones, etc. OFBiz developers should never use millisecond arithmetic.