KEP: 2 Title: Modification of datetime type, introduction of 'tz' sub-tag Version: $Revision$ Last-Modified: $Date$ Author: Georg Greve <greve@kolabsys.com> Status: Draft Type: Design Content-Type: text/x-rst Created: 2010-11-16

Abstract

Kolab used to store all times in UTC and did not allow for time zone information. For recurring events in parts of the world with Daylight Saving Time (DST) regimes this means the time of the event is changing, as DST is defined in a **dynamic** offset towards UTC.

In order to achieve a recurring event that retains its local time across DST transitions, a client must know which time zone to use. The implicit assumption of older clients to always use local time zone is problematic, as explained in "Description of current client behaviour" below. So enabling time zone information for datetime fields is essential.

Some reference for background was provided on the Kolab format list [1] and there have been various proposals for resolution of the issue, including adding time zone information in a separate XML tag, along with DST details. [2] This document is based upon those discussions and draws inspiration from the various proposals, resulting in the described update which follows the approach of least invasive change with least burden on client implementors.

Update to the XML Format

All objects hold datetime in the form of creation and modification times. Consistent time handling across all object types and occurences of time objects is highly desirable. The following change therefore affects all Kolab object types.

Change of type: datetime

The type for datetime storage in Kolab XML is modified as follows:

- All datetime storage fields **MUST** be stored in the format described by *RFC3339: Date and Time on the Internet: Timestamps* [3].
- All such timestamps **MUST** be calculated in standard time, **NOT** in Daylight Saving Time (DST).
- All datetime storage fields **MAY** carry up to one 'tz' hierarchically nested sub-tag describing the time zone in the uniform naming convention designed by Paul Eggert, specifying time zones from the Olson database, a.k.a. tz database, a.k.a. zoneinfo database [4].

Examples of valid 'start-date' fields using datetime structures according to the above specification are

- <start-date>2010-01-31T11:27:21Z</start-date>
- <start-date>2010-01-31T11:27:21+00:00</start-date>
- <start-date>1996-12-19T16:39:57-08:00</start-date>
- <start-date>1996-12-19T16:39:57-08:00<tz>America/Sao_Paulo</tz></start-date>

Note: The first two examples are identical to the datetime type used in Kolab XML storage prior to this modification.

When time zone information is provided, a client **MUST** consider the event local to this time zone. Recurrence **MUST** then be calculated to keep the event at the same local time within that time zone, adjusting the time for the event accordingly for the client's local time zone.

When no time zone information is provided, a client **MUST** calculate recurrences strictly according to UTC.

When modifying objects, clients **MUST** preserve the original time zone used for storage.

When adding new objects, clients **SHOULD** default to the local time zone of the user, but SHOULD allow the user to select the time zone for storage and consequently recurrence calculation.

Upgrade Path

All clients already must preserve all tags they do not understand. So the newly introduced 'tz' tag must be preserved by older clients that do not understand it.

There will be some change of client behaviour as older events now displayed in newer clients now correctly start changing their local time when entering/leaving DST. This is preferable to current client behaviour as it will ensure all users get the same time for an event and are no longer going to miss each other.

Unavoidably, older clients will continue to display recurrence times incorrectly. This is neither an improvement nor a deterioration of the current situation.

Smart Upgrade Option

Clients **MAY** choose to use the 'product-id' and absence of 'tz' tag to identify which event was created by an older client, and silently update it to have recurrence behave in the way the user expects. When doing so, it is recommended to assume the client's local time zone was the one for which the recurrence should be stable in local time, as that was what clients were assuming and showing to the user thus far.

As this provides a substantial amount of work for clients, this is **NOT** a requirement.

Background notes on backwards-compatibility

This modification represents the least invasive change to redress the existing issues.

While it is preferable to have correct recurrence calculation, it would be even better if this change could take place without notice by the user. This would only become possible if clients had the chance to differentiate between objects according to the format prior to this update, and those afterwards. This would require either an additional field, which older clients could ignore, but newer clients MUST use to allow for differentiation, -- OR -- a change in object version number.

Because there are multiple datetime fields possible in any object, and additional use of such structures is conceivable in the future, such an additional field had to be introduced in a nested structure. Making its usage mandatory to gain certainty of whether an object was written by a new client seemed to add cruft without much benefit. A version change on the other hand would break older clients entirely, likely resulting in error messages or "disappearing" events for the user.

Both approaches would logically favour a "fix broken objects when you see them" policy, which is much more demanding and burdensome for client implementors, and more error prone than the "lazy update" path described above, which relies on the user to identify unwanted behaviour and then make a decision about how to "fix" the event.

Description of current client behaviour

Existing clients currently make the implicit assumption that the time was specified in and should be calculated against the local time zone of the client itself. This will lead to issues when a user is changing time zones, or when participants in multiple time zones are concerned. This behaviour could be confirmed with both Kontact and the Kolab Web Client Horde.

A weekly meeting is set for 11:00 every Wednesday in Zurich, Switzerland, starting on 23 June 2010. This gets translated stored in Kolab XML as 2010-06-23T09:00:00Z. On Wednesday 17 November 2010 Switzerland has switched out of DST, the local timezone is therefore UTC+1. If correctly interpreting the stored information, the meeting should now start at 10:00. At 09:50 the KDE Reminder Daemon correctly informs the user that the conference call is about to start in 10 minutes.

KDE Kontact however incorrectly displays the meeting as scheduled for 11:00. The same is true for the Kolab web client based on Horde for all versions of Kolab <= 2.2.4. This however is equivalent to 10:00 UTC. When adding another user in Sao Paulo, Brazil to the equation, the event is shown as taking place at 06:00 local time, or 08:00 UTC, due to the Brazilian summer time with an offset of UTC-3 that went into the assumption for the calculation of the recurrence. The result is that two users, while being presented with a data set that looks consistent, will miss each other by two hours.

Which other clients exhibit the same behaviour is unclear, but it seems there is no reasonable assumption that current behaviour correctly models any rational use case.

References

Copyright

---

1 Timezone / recurrence scheduling discussion for Kolab, Greve (http://kolab.org/pipermail/kolab-format/2010-October/001004.html)

2 Recurring events with timezone, Helwich (http://kolab.org/pipermail/kolab-format/2010-October/000999.html)

3 RFC3339: Date and Time on the Internet: Timestamps, Klyne, Newman (http://www.ietf.org/rfc/rfc3339.txt)

4 Wikipedia: Zoneinfo (https://secure.wikimedia.org/wikipedia/en/wiki/Zoneinfo)