Kepler - Bug #6893

DateToken value inconsistent

12/02/2015 04:33 PM - Daniel Crawl

Status:	New	**Start date:**	12/02/2015
Priority:	Low	**Due date:**	
Assignee:	Christopher Brooks	**% Done:**	20%
Category:	actors	**Estimated time:**	0.00 hour
Target version:	2.6.0	**Spent time:**	1.50 hour
Bugzilla-Id:			

Description

DateToken stores the date internally in _value and _calendar, but these fields are not kept consistent. For example, if you use ModifyDate to add 2 seconds, _value is not updated. See the attached test workflow.

Also, ModifyDate changes the state of the input token instead of creating a new one - I thought this was not allowed in actors.

History

#1 - 12/03/2015 02:48 PM - Christopher Brooks

- % Done changed from 0 to 20

The code is not too well documented, but I believe that one issue is that java.util.Calendar does not have sufficient precision to represent nanoseconds and microseconds. The DateToken seems to be constructed to try to handle this.

ModifyDate has a bug because it updates the _calendar field of the token, but does not update _value field. The DateToken.getValue() field returns the value of the _value field, which as not been updated.

In particular, ModifyDate.fire() contained code like:

```
if (unitString.equals("Year")) {
                token.getCalendarInstance().add(Calendar.YEAR, val);
            } else if (unitString.equals("Month")) {
                token.getCalendarInstance().add(Calendar.MONTH, val);
            } else if (unitString.equals("Day")) {
                token.getCalendarInstance().add(Calendar.DAY_OF_MONTH, val);
            } else if (unitString.equals("Hour")) {
                token.getCalendarInstance().add(Calendar.HOUR_OF_DAY, val);
            } else if (unitString.equals("Minute")) {
                token.getCalendarInstance().add(Calendar.MINUTE, val);
            } else if (unitString.equals("Second")) {
                token.getCalendarInstance().add(Calendar.SECOND, val);
            } else if (unitString.equals("Millisecond")) {
                token.getCalendarInstance().add(Calendar.MILLISECOND, val);
            } else if (unitString.equals("Microsecond")) {
                token.addMicroseconds(val);
            } else if (unitString.equals("Nanosecond")) {
                token.addNanoseconds(val);
            } else {
                throw new IllegalActionException(this, "The unit " + unitString
                        + " is not supported");
            }

output.send(0, token);
```

Only if the unitString is Microsecond or Nanosecond, then the _value field will be updated.

I don't have a good fix for this. I think this class tries to do two things (submillisecond support and string parsing) and does neither well.

My fix was to add a method to DateToken that updates _value and _calendar:

```
/** Set the time in milliseconds since January 1, 1970.
     *  @paramn The time as a long value.
     *  @see #getTimeInMilliseconds();
     */
    public void setTimeInMilliseconds(long newValue) {
        // FIXME: This is a poor design because we are exposing
```

```
        // _value with a setter.
        if (_precision == PRECISION_NANOSECOND) {
            _value = newValue * 1000000;
        } else if (_precision == PRECISION_MICROSECOND) {
            _value = newValue * 1000;
        } else if (_precision == PRECISION_MILLISECOND) {
            _value = newValue;
    } else if (_precision == PRECISION_SECOND) {
            _value = newValue / 1000;
        }
        _calendar.setTimeInMillis(newValue);
    }
```

I updated ModifyDate to use that. A better solution would be to rewrite this code using the Java 8 Date infrastructure.

Daniel wrote: | Also, ModifyDate changes the state of the input token instead of creating a new one - I thought this was not allowed in actors.

I looked at ModifyDate and in fire() it reads the input and creates a new DateToken:

```
public void fire() throws IllegalActionException {
        super.fire();
        if (input.hasToken(0)) {
            DateToken inputToken = (DateToken) input.get(0);

DateToken token = new DateToken(inputToken.getValue(),
                    inputToken.getPrecision(), inputToken.getTimeZone());
```

Later, the token gets modified in various ways. I agree that this is not optimal, but the code is not modifying the input token.

I checked in some code and the test. The test now passes.

Technically, this bug is solved, but there are such basic issues with DateToken that I'm fine with leaving it open and either retargetting it or lowering the priority.

#2 - 12/04/2015 09:18 AM - Daniel Crawl

- Priority changed from Normal to Low

Thanks for the fix, Christopher. It works now for my use case.

I'll leave the bug open and lower the priority.

Files

ModifyDate2.xml	30.3 KB	12/03/2015	Daniel Crawl